**See Also**

Object Type
Creating, Running, and Distributing Executable (.EXE) Files
Registering OLE Custom Controls
Trappable Errors for Windows 95 Custom Controls
Using Custom Properties Dialog Boxes
Visual Basic Custom Control Constants

## Object Type

In Visual Basic, a control's object type is used with the **TypeOf** keyword in an **If...Then...Else** statement. This is useful for creating a variable of that object type or determining the type of a control that is passed as an argument to an event (for example, the *source* argument of the DragDrop event). For more information on using a control's object type, search Help for the **If** keyword.

The object type, or class name, for each control is listed in the following table.

| Control | Object type |
|---|---|
| 3D check box | SSCheck |
| 3D command button | SSCommand |
| 3D frame | SSFrame |
| 3D group push button | SSRibbon |
| 3D option button | SSOption |
| 3D panel | SSPanel |
| Animated button | AniPushButton |
| Communications | MSComm |
| Data Outline | DataOutline |
| Gauge | Gauge |
| Graph | Graph |
| ImageList | ImageList |
| Key state | MhState |
| ListView | ListView |
| MAPI | MapiSession, MapiMessages |
| Masked edit | MaskEdBox |
| Multimedia MCI | MMControl |
| Outline | Outline |
| Picture clip | PictureClip |
| ProgressBar | ProgressBar |
| RichTextBox | RichTextBox |
| Slider | Slider |
| Spin button | SpinButton |
| SSTab | SSTab |
| StatusBar | StatusBar |
| TabStrip | TabStrip |
| ToolBar | ToolBar |
| TreeView | TreeView |

## Creating, Running, and Distributing Executable (.EXE) Files

To run your application under Microsoft Windows outside Visual Basic, create an executable (.EXE) file. You can create executable files for applications that use custom controls the same way you do for any other application.   There are a few issues to consider, however, when running such an application. See the following topics for more information.

Visual Basic Executable (.EXE) Files

Required Custom Control Files

## Visual Basic Executable (.EXE) Files

A custom control file is a DLL that is accessed both by Visual Basic and applications created by using Visual Basic. When you run an executable file that contains a custom control, the .OCX file associated with it must be on your system's path or in the same directory as the .EXE file. Otherwise, the application will not be able to find the code needed to create the control.

If a custom control can't be found, the Visual Basic run-time DLL generates the error message `File Not Found`. To distribute an application that uses custom controls, it is recommended that your installation procedure copy all required .OCX files into the user's Microsoft Windows \SYSTEM subdirectory.

You can freely distribute any application you create with Visual Basic to any Microsoft Windows user. (Visual Basic provides a Setup Wizard for writing your own application setups.) Users will need copies of the following:

- The Visual Basic run-time file (VBRUN40016DLL or VBRUN40032.DLL).
- Any .OCX files.
- Additional DLLs as required by your application or by custom controls.

## Required Custom Control Files

The files required by each custom control are listed in the following table.

| Control | Required files |
|---|---|
| 3D check box | THREED16.OCX (16 bit)<br>THREED32.OCX (32 bit) |
| 3D command button | THREED16.OCX (16 bit)<br>THREED32.OCX (32 bit) |
| 3D frame | THREED16.OCX (16 bit)<br>THREED32.OCX (32 bit) |
| 3D group push button | THREED16.OCX (16 bit)<br>THREED32.OCX (32 bit) |
| 3D option button | THREED16.OCX (16 bit)<br>THREED32.OCX (32 bit) |
| 3D panel | THREED16.OCX (16 bit)<br>THREED32.OCX (32 bit) |
| Animated button | ANIBTN16.OCX (16 bit)<br>ANIBTN32.OCX (32 bit) |
| Communications | MSCOMM16.OCX (16 bit)<br>MSCOMM32.OCX (32 bit) |
| Gauge | GAUGE16.OCX (16 bit)<br>GAUGE32.OXC (32 bit) |
| Graph | GRAPH16.OCX, GSW16.EXE,<br>GSWDLL16.DLL (16 bit)<br>GRAPH32.OCX, GSW32.EXE,<br>GSWDLL32.DLL (32 bit) |
| ImageList | COMCTL.OCX (32 bit only) |
| Key state | KEYSTA16.OCX (16 bit)<br>KEYSTA32.OCX (32 bit) |
| ListView | COMCTL.OCX (32 bit only) |
| MAPI * | MSMAPI16.OCX (16 bit)<br>MSMAPI32.OCX (32 bit) |
| Masked edit | MSMASK16.OCX (16 bit)<br>MSMASK32.OCX (32 bit) |
| Multimedia MCI ** | MCI16.OCX (16 bit)<br>MCI32.OCX (32 bit) |
| Outline | MSOUTL16.OCX (16 bit)<br>MSOUTL32.OCX (32 bit) |
| Picture clip | PICCLP16.OCX (16 bit)<br>PICCLP32.OCX (32 bit) |
| ProgressBar | COMCTL.OCX (32 bit only) |
| RichEdit | RICHTX32.OCX (32 bit only) |
| Slider | COMCTL.OCX (32 bit only) |
| Spin button | SPIN16.OCX (16 bit)<br>SPIN32.OCX (32 bit) |
| SSTab | COMCTL.OCX (32 bit only) |
| StatusBar | COMCTL.OCX (32 bit only) |
| TabStrip | COMCTL.OCX (32 bit only) |
| Toolbar | COMCTL.OCX (32 bit only) |
| Treeview | COMCTL.OCX (32 bit only) |

* Microsoft Mail for Windows electronic mail system required.

** Multimedia PC required.

## Registering OLE Custom Controls

When you install the Professional Edition, Visual Basic 4.0 automatically registers its OLE custom controls in the system registry. You are then able to use the custom controls at design time to build your applications.

If you plan to create a set-up program for your application, you'll need to include information on any OLE custom controls in the SETUP.LST file. For more information, see chapter 30, "Distributing Your Applications," in the *Programmer's Guide*.

The VB.LIC file, shipped in previous versions of Visual Basic, is not used for OLE Custom Controls.

**Note** It is a violation of your license agreement to copy and distribute any information from the Licenses section of the system registry.

## Using Custom Properties Dialog Boxes

When setting the properties of a custom control, you may need or prefer to use the control's custom properties dialog box. This dialog box provides an alternative to the list of properties in the Properties window for setting control properties at design time.

### Two Ways to Set Properties

The reason for the custom properties dialog box is that not all applications that use custom controls provide a Properties window like the one in Visual Basic. The dialog box provides an interface for setting key control properties regardless of the interface supplied by the hosting application.

For some control properties, you choose either of these two locations to set the property:

- The Properties window
- The custom properties dialog box

In some cases, the dialog box is the only way to set a property at design time. This is usually the situation when the interface needed to set a property doesn't work inside the Properties window. For example, assigning a series of images to an **ImageList** control involves more than typing the name of a file or choosing from a list.

### Finding the Dialog Box

Not all custom controls provide a custom properties dialog box. To see whether a control provides this dialog box, scroll the list of properties in the Properties Window to the top. If the list of properties contains the name (Custom), then the control provides the dialog box.

### Using the Dialog Box

After you choose the (Custom) entry in the Properties window, click the Properties button to display the control's custom properties dialog box, often presented as a tabbed dialog box. Chose the tab that contains the interface for setting the properties that you want to set.

After you make changes in one tab, you can often apply those changes immediately by clicking the Apply button (if provided). You can click other tabs to set other properties as needed. To approve all changes made in the dialog box, click the OK button. To return to the Properties window without changing any property settings, click the Cancel button.

Documents the SetupWizard application.   For information about the Setup Toolkit, see the Visual Basic Help file.

Documents Visual Basic for Windows.

Documents the Data Access application.

Documents the Data Manager application.

Tutorials for learning to use Visual Basic for Windows.

Documents Microsoft Support Services.

Lists the applications written in Visual Basic that demonstrate techniques discussed in the printed documentation.

Documents the custom controls provided with the Professional Edition.

Documents the Crystal Reports application.

Documents the segmented hypergraphic editor for creating hotspots within graphics for use in authoring Help files.

Documents the installation tools for ODBC.

Documents the ODBC driver for SQL Server databases.

Documents the VisData sample application.

Documents Windows functions as used in the C programming language.

Documents the Code Profiler add-in.

Documents Remote Automation, the Component Manager, Remote Data Objects (RDO), and the RemoteData control provided with the Enterprise Edition.

Documents the SourceSafe add-in for administrators.

Documents the SourceSafe add-in for users.

## Text Files

Microsoft Visual Basic 4.0 includes additional information in the following files:

| Text File | Description |
|---|---|
| APILOD.TXT | Describes how to use the API Text Viewer. |
| LABELS.TXT | Contains information about mailing labels. |
| PACKING.LST | Lists all files on the distribution disks provided with Visual Basic. |
| VB4DLL.TXT | Contains additional information about developing dynamic link libraries (DLLs) to use with Visual Basic. |
| WIN31API.TXT | Contains procedure, constant, and type declarations for 16-bit versions of Windows API functions. |
| WIN32API.TXT | Contains symbolic constants for 32-bit versions of Windows API functions. |
| WINMMSYS.TXT | Contains procedure, constant, and type declarations for Windows 3.1 multimedia API functions. |

## 3D Check Box Control

Properties Methods Events Constants Error Messages

### Description

The 3D check box control emulates the standard Visual Basic check box control, which displays an option that can be turned on or off. In addition, this control allows you to align three-dimensional text to the right or left of the check box.

### File Name

THREED16.OCX, THREED32.OCX

### Class Name

SSCheck

### Remarks

The 3D check box has several custom properties that allow you to adjust the three-dimensional appearance of the control. When you draw a 3D check box on a form, the custom property settings for the control are saved and used as a template for the next 3D check box that you create.

Since the three-dimensional gray scale look requires a background color of light gray, the BackColor property is not available with this control. In Visual Basic, this control should be used on forms that have the BackColor property set to light gray (&H00C0C0C0&).

### Bound Properties

The 3D check box has three bound properties: DataChanged, DataField, and DataSource. This means that it can be linked to a data control and display field values for the current record in the recordset. The 3D check box can only be bound to a field that is of a boolean data type. The 3D check box control can also write out values to the recordset.

When the value of the field referenced by the DataField property is read, it is converted to a Value property value, if possible. If the field value is NULL, then the Value property is set to 2, which means the check box is grayed.

For more information on using bound controls, refer to Chapter 22, "Accessing Databases With the Data Control," in the *Programmer's Guide*.

---

**Distribution Note** When you create and distribute applications that use the 3D check box control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Wizard included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

---

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *Alignment | Font | Height | Parent |
| Caption | *Font3D | HelpContextID | TabIndex |
| Container | FontBold | hWnd | TabStop |
| DataChanged | FontItalic | Index | Tag |
| DataField | FontName | Left | Top |
| DataSource | FontSize | MouseIcon | Value |
| DragIcon | FontStrikethru | MousePointer | Visible |
| DragMode | FontUnderline | Name | WhatsThisHelpID |
| Enabled | ForeColor | Object | Width |

Value is the default value of the control.

**Note** The DragIcon, DragMode, HelpContextID, Index, and Parent properties are only available in Visual Basic. The Name property is the same as the CtlName property in Visual Basic 1.0.

The DataChanged, DataField, and DataSource properties are bound properties and are only available in Visual Basic 3.0.

## Events

All of the events for this control are listed in the following table. Events that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *Click | GotFocus | KeyUp | MouseMove |
| DragDrop | KeyDown | LostFocus | MouseUp |
| DragOver | KeyPress | MouseDown | |

**Note** The DragDrop and DragOver events are only available in Visual Basic.

**Methods**

All of the methods for this control are listed in the following table.

| | | |
|---|---|---|
| Drag | Refresh | ZOrder |
| Move | SetFocus | ShowWhatsThis |

**Note** The **Drag** and **ZOrder** methods are only available in Visual Basic.

## 3D Command Button Control

Properties Methods Events Constants Error Messages

The 3D command button control emulates the standard Visual Basic command button control, which performs a task when the user either clicks the button or presses a key. In addition, this control can display a three-dimensional caption as well as a bitmap or icon. A variable bevel width allows the button to appear raised off the screen.

**File Name**

THREED16.OCX, THREED32.OCX

**Class Name**

SSCommand

**Remarks**

The 3D command button has several custom properties that allow you to adjust the three-dimensional appearance of the control. When you draw a 3D command button on a form, the custom property settings for the control are saved and used as a template for the next 3D command button that you create.

Since the three-dimensional gray scale look requires a background color of light gray, the BackColor property is not available with this control. In Visual Basic, this control should be used on forms that have the BackColor property set to light gray (&H00C0C0C0&).

---

**Distribution Note** When you create and distribute applications that use the 3D command button control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

---

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *AutoSize | *Font3D | hWnd | *RoundedCorners |
| *BevelWidth | FontBold | Index | TabIndex |
| Cancel | FontItalic | Left | TabStop |
| Caption | FontName | MouseIcon | Tag |
| Container | FontSize | MousePointer | Top |
| Default | FontStrikethru | Name | Value |
| DragIcon | FontUnderline | Object | Visible |
| DragMode | ForeColor | *Outline | WhatsThisHelpID |
| Enabled | Height | Parent | Width |
| Font | HelpContextID | *Picture | |

Value is the default value of the control.

---

**Note** The DragIcon, DragMode, HelpContextID, Index, and Parent properties are only available in Visual Basic. The Name property is the same as the CtlName property in Visual Basic 1.0.

---

## Events

All of the events for this control are listed in the following table.

| | | | |
|---|---|---|---|
| Click | GotFocus | KeyUp | MouseMove |
| DragDrop | KeyDown | LostFocus | MouseUp |
| DragOver | KeyPress | MouseDown | |

**Note** The DragDrop and DragOver events are only available in Visual Basic.

**Methods**

All of the methods for this control are listed in the following table.

| | | |
|---|---|---|
| Drag | Refresh | ZOrder |
| Move | SetFocus | ShowWhatsThis |

**Note** The **Drag, SetFocus,** and **ZOrder** methods are only available in Visual Basic.

## Picture Property, 3D Command Button Control

Example

Specifies a bitmap or an icon to display on the command button. This property is write-only at design time.

### Syntax

[*form*.]*CommandButton3d*.**Picture**[ = *picture*]

### Remarks

The following table lists the Picture property settings for the 3D command button control.

| Setting | Description |
|---|---|
| (none) | (Default) No picture. |
| (bitmap) or (icon) | Designates a graphic to display. You can load the graphic from the Properties window at design time. |

In Visual Basic, you can load a graphic at design time from the Properties window. At run time, you can set this property by using the **LoadPicture** function on a bitmap or icon, or you can use Clipboard methods such as **GetData, SetData,** and **GetFormat** with the nontext Clipboard formats **vbCFBitmap** and **vbCFDIB**, as defined in the object library in the Object Browser.

If you set the Picture property at design time, the graphic is saved and loaded with the form. If you create an executable file, the file contains the image. When you load a graphic at run time, the graphic is not saved with the application. Use the **SavePicture** function to save a graphic from a form or picture box into a file.

**Note** This control can display bitmaps (.BMP) and icons (.ICO), but not Windows metafiles (.WMF). At run time, you can set the Picture property to any other object's DragIcon, Icon, Picture, or Image property, or you can assign it the graphic returned by the **LoadPicture** function. You can only assign the Picture property directly.

### Data Type

**Integer**

## Picture Property Example, 3D Command Button Control

The following example pastes a bitmap from the Clipboard onto a command button. To try this example, create a form with a command button, and then, in another application, copy a picture onto the Clipboard, switch to Visual Basic, and run this example.

**Note** The picture must be on the Clipboard in bitmap form.

```
Private Sub Form_Click ()
   Const vbCFBitmap = 2
   Command3D1.Picture = Clipboard.GetData(vbCFBitmap)
End Sub
```

## 3D Frame Control

Properties    Methods    Events    Constants    Error Messages

The 3D frame control emulates the standard Visual Basic frame control, which provides a graphical or functional grouping of controls. The 3D frame control also allows the use of three-dimensional text (right, left, or centered in the frame), and the frame itself can appear raised or inset.

**File Name**

THREED16.OCX, THREED32.OCX

**Class Name**

SSFrame

**Remarks**

The 3D frame has several custom properties that allow you to adjust the three-dimensional appearance of the control. When you draw a 3D frame on a form, the custom property settings for the control are saved and used as a template for the next 3D frame that you create.

Since the three-dimensional gray scale look requires a background color of light gray, the BackColor property is not available with this control. In Visual Basic, this control should be used on forms that have the BackColor property set to light gray (&H00C0C0C0&).

---

**Distribution Note** When you create and distribute applications that use the 3D frame control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

---

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *Alignment | FontBold | hWnd | *ShadowColor |
| Caption | FontItalic | Index | *ShadowStyle |
| Container | FontName | Left | TabIndex |
| DragIcon | FontSize | MouseIcon | Tag |
| DragMode | FontStrikethru | MousePointer | Top |
| Enabled | FontUnderline | Name | Visible |
| Font | ForeColor | Object | WhatsThisHelpID |
| *Font3D | Height | Parent | Width |

Caption is the default value of the control.

**Note**   The Align, DragIcon, DragMode, HelpContextID, Index, and Parent properties are only available in Visual Basic. The Name property is the same as the CtlName property in Visual Basic 1.0.

## Events

All of the events for this control are listed in the following table.

| | | | |
|---|---|---|---|
| Click | DragDrop | MouseDown | MouseUp |
| DblClick | DragOver | MouseMove | |

**Note** The DragDrop and DragOver events are only available in Visual Basic.

## Methods

All of the methods for this control are listed in the following table.

| | | | |
|---|---|---|---|
| Drag | Refresh | ShowWhatsThis | ZOrder |
| Move | | | |

**Note** The **Drag** and **ZOrder** methods are only available in Visual Basic.

## ShadowStyle Property, 3D Frame Control

Determines whether the frame appears inset or raised.

**Syntax**

[*form*.]*Frame3d*.**ShadowStyle**[ = *color%*]

**Remarks**

The following table lists the ShadowStyle property settings for the 3D frame control.

| Setting | Description |
|---|---|
| 0 | (Default) Inset. Frame appears inset into the form. |
| 1 | Raised. Frame appears raised off the form. |

**Data Type**

**Integer** (Enumerated)

## 3D Group Push Button Control

Properties Methods Events Constants Error Messages

The 3D group push button control is a push button that turns its state on and off when clicked. Individual 3D group push buttons can be used in groups to emulate the functionality of the tool bar in Microsoft Excel spreadsheets or the ribbon in Microsoft Word for Windows word processing program. This control has a Picture property to which a bitmap graphic can be assigned.

### File Name

THREED16.OCX, THREED32.OCX

### Class Name

SSRibbon

### Remarks

The buttons on the 3D group push button control look similar to command buttons, but they behave more like option buttons; that is, depressing one button within a button group automatically raises the previously depressed button. You group buttons using the GroupNumber property. The GroupAllowAllUp property also allows all 3D group push buttons in a group to be in the up position.

The button has three picture properties: PictureUp, PictureDn, and PictureDisabled. The PictureDisabled property determines which graphic is displayed when the button is in the disabled state. You can specify both PictureUp and PictureDn properties, or you can specify the up bitmap only, in which case the 3D group push button will either dither, invert, or use the unchanged up bitmap when displaying the button in the down position. You choose the type of change with the PictureDnChange property.

---

**Note** If the BevelWidth property is set to 1 or 2 rather than 0, the bitmap that you specify is only for the area inside the bevels. The 3D group push button takes care of drawing the bevels and offsetting the bitmap down and to the right when it is pressed. However, you may set the BevelWidth property to 0 and incorporate the button shading for the up and down positions in your pictures.

---

Unlike most three-dimensional controls, the 3D group push button has a BackColor property. The BackColor property defaults to light gray, but it can be changed to match the background color of the bitmap that is placed on it. In this way a bitmap with a dominant background color can appear to be part of the button. Note that the BackColor property only affects the area inside the 3D group push button's beveled edges. The edges are always shaded with white and dark gray.

The 3D group push button has several custom properties that allow you to adjust the three-dimensional appearance of the control. When you draw a 3D group push button on a form, the custom property settings for the control are saved and used as a template for the next 3D group push button that you create.

---

**Distribution Note** When you create and distribute applications that use the 3D group push button control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

---

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *AutoSize | *GroupNumber | Name | *RoundedCorners |
| BackColor | Height | Object | TabIndex |
| *BevelWidth | HelpContextID | *Outline | Tag |
| Container | hWnd | Parent | Top |
| DragIcon | Index | *PictureDisabled | Value |
| DragMode | Left | *PictureDn | Visible |
| Enabled | MouseIcon | *PictureDnChange | WhatsThisHelpID |
| *GroupAllowAllUp | MousePointer | *PictureUp | Width |

Value is the default value of the control.

**Note** The Align, DragIcon, DragMode, HelpContextID, Index, and Parent properties are only available in Visual Basic. Name is the same as the CtlName property in Visual Basic 1.0.

## Events

All of the events for this control are listed in the following table. Events that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *Click | DragOver | MouseMove | MouseUp |
| DragDrop | MouseDown | | |

**Note** The DragDrop and DragOver events are only available in Visual Basic.

**Methods**

All of the methods for this control are listed in the following table.

| | | | |
|---|---|---|---|
| Drag | Move | Refresh | Zorder |
| ShowWhatsThis | | | |

**Note** The **Drag** and **ZOrder** methods are only available in Visual Basic.

## GroupAllowAllUp Property, 3D Group Push Button Control

Determines whether all buttons in a logical group can be in the up position.

**Syntax**

[*form.*]*GroupPushButton.***GroupAllowAllUp**[ = {**True | False**}]

**Remarks**

The following table lists the GroupAllowAllUp property settings for the 3D group push button control.

| Setting | Description |
|---|---|
| **True** | (Default) All buttons in the current logical group may be in the up position. |
| **False** | At least one button in the current logical group must be depressed. |

The setting of the GroupAllowAllUp property for a button in one group has no effect on any other group.

If the GroupAllowAllUp property is set to **False,** no check will be made by the 3D group push button control to ensure that at least one button is depressed when the form on which the button resides is loaded. It is up to you to set the initial state of the Value property for one of the buttons in the group to **True** (depressed).

**Note** When the GroupAllowAllUp property is set for a button in a logical group, the GroupAllowAllUp property is automatically set to the same value for all the other buttons in the group. Use the GroupNumber property to create logical groups of 3D group push buttons.

**Data Type**

**Integer** (Boolean)

## GroupNumber Property, 3D Group Push Button Control

Sets or returns the GroupNumber associated with the 3D group push button.

**Syntax**

[*form*.]*GroupPushButton*.**GroupNumber**[ = *group%*]

**Remarks**

The following table lists the GroupNumber property settings for the 3D group push button control.

| Setting | Description |
|---|---|
| 0 | The button is not part of a logical grouping and as such can be turned on and off (by means of code or a mouse click) independently of any other group push buttons on the form. |
| 1− 99 | (Default = 1) The button is a member of a logical grouping of 3D group push buttons (that is, other buttons on the same form with the same GroupNumber property setting). |

The GroupNumber property only has a grouping effect on buttons that are siblings, that is, buttons with the same parent. For example, in Visual Basic, you could consider two buttons placed directly on a form siblings, and you can use their GroupNumber property to group them. Then, if you place a third button in a frame control on the same form, the third button would not be a sibling of the first two, even though they are all on the same form.

This property defaults to 1, and all sibling buttons form a group.

If this property is set to 0, the button will operate independently. It will turn its state on or off when clicked.

It is possible to set up multiple logical groups on a single form, frame, panel, or picture box by varying the GroupNumber property. All siblings with the same GroupNumber will operate as a group.

**Note** There are two types of groups. The first type requires that at least one button in the group be depressed (it operates like an option button group); the other type allows all buttons to be up. Refer to the GroupAllowAllUp property for details.

**Data Type**

**Integer**

## PictureDisabled Property, 3D Group Push Button Control

Specifies a bitmap to display on the 3D group push button when it is disabled. This property is write-only at design time.

**Syntax**

[*form*.]*GroupPushButton*.**PictureDisabled**[ = *picture*]

**Remarks**

The following table lists the PictureDisabled property settings for the 3D group push button control.

| Setting | Description |
|---|---|
| (none) | (Default) No bitmap is specified for display when the button is disabled. |
| (bitmap) | Designates a graphic to display on the button when it is disabled. You can load the graphic from the Properties window at design time. |

This graphic is only displayed if the 3D group push button is disabled, that is, its Enabled property is set to **False**. Setting this property is optional. If you do not set this property, the button will display the graphic specified for the PictureUp property.

In Visual Basic, you can load a graphic at design time from the Properties window. At run time, you can set this property by using the **LoadPicture** function on a bitmap or, you can use Clipboard methods such as **GetData**, **SetData**, and **GetFormat** with the nontext Clipboard formats **vbCFBitmap** and **vbCFDIB**, as defined in the Visual Basic (VB) object library in the Object Browser.

When setting the Picture property at design time, the graphic is saved and loaded with the form. If you create an executable file, the file contains the image. When you load a graphic at run time, the graphic is not saved with the application. Use the **SavePicture** statement to save a graphic from a form or picture box into a file.

**Note** At run time, you can set the Picture property to any other object's Picture or Image property, or you can assign it the graphic returned by the **LoadPicture** function. The Picture property can only be assigned directly.

**Data Type**

**Integer**

## PictureDn Property, 3D Group Push Button Control

Specifies a bitmap to display on the button when it is in the depressed or down position. This property is write-only at design time.

### Syntax

[*form*.]*GroupPushButton*.**PictureDn**[ = *picture*]

### Remarks

The following table lists the PictureDn property settings for the 3D group push button control.

| Setting | Description |
|---|---|
| (none) | (Default) No bitmap is specified for display when the button is down. When the button is down, the PictureUp bitmap is displayed modified, as determined by the PictureDnChange property. |
| (bitmap) | Designates a graphic to display on the button when it is down. You can load the graphic from the Properties window at design time. |

This bitmap is displayed only if the button is in the down state; that is, the Value property is **True**. It is not necessary to assign a bitmap to this property; if this property is set to none, the 3D group push button automatically creates the bitmap to be displayed when the button is in the down position. See the PictureDnChange property for an explanation of the options available when you want to have the 3D group push button create the down bitmap.

If the BevelWidth property is set to 1 or 2 rather than 0, the bitmap that you specify is only for the area inside the bevels. The 3D group push button takes care of drawing the bevels and offsetting the bitmap down and to the right when it is pressed. However, you may set the BevelWidth property to 0 and incorporate button shading for the up and down positions in your pictures.

You can load a graphic at design time from the Properties window. At run time, you can set this property by using the **LoadPicture** function on a bitmap, or you can use Clipboard methods such as **GetData**, **SetData**, and **GetFormat** with the nontext Clipboard formats **vbCFBitmap** and **vbCFDIB** as defined in the Visual Basic (VB) object library in the Object Browser.

When setting the Picture property at design time, the graphic is saved and loaded with the form. If you create an executable file, the file contains the image. When you load a graphic at run time, the graphic is not saved with the application. Use the **SavePicture** statement to save a graphic from a form or picture box into a file.

**Note** At run time, the Picture property can be set to any other object's Picture or Image property, or you can assign it the graphic returned by the **LoadPicture** function. The Picture property can only be assigned directly.

### Data Type

**Integer**

## PictureDnChange Property, 3D Group Push Button Control

Determines how the PictureUp bitmap is used to create the PictureDn bitmap if a PictureDn bitmap is not specified.

### Syntax

[*form*.]*GroupPushButton*.**PictureDnChange**[ = *setting%*]

### Remarks

The following table lists the PictureDnChange property settings for the 3D group push button control.

| Setting | Description |
|---|---|
| 0 | PictureUp bitmap unchanged. |
| 1 | (Default) Dither PictureUp bitmap. Create a copy of the up bitmap and change every other pixel that is in the BackColor color to white. This has the effect of lightening that color (for example, light gray will appear to be a lighter shade of gray). |
| 2 | Invert PictureUp bitmap. |

When using setting 1 with large bitmaps, due to the overhead of dithering the bitmap, there is a slight time lag the first time the button is pressed. If the time lag is unacceptable, use one of the other settings, or specify a PictureDn bitmap.

### Data Type

**Integer** (Enumerated)

## PictureUp Property, 3D Group Push Button Control

Specifies a bitmap to display on the button when it is in the up position. This property is write-only at design time.

**Syntax**

[*form*.]*GroupPushButton*.**PictureUp**[ = *picture*]

**Remarks**

The following table lists the PictureUp property settings for the 3D group push button control.

| Setting | Description |
|---|---|
| (none) | (Default) No bitmap is specified for display when the button is in the up position. |
| (bitmap) | Designates a graphic to display on the button when it is up. You can load the graphic from the Properties window at design time. |

This bitmap is displayed if the button is in the up state; that is, the Value property is **False**. If the PictureDn property is set to none, you can also use the PictureUp to create the bitmap to be displayed when the button is in the down position. See the PictureDnChange property for an explanation of the options available when you choose to have the 3D group push button create the down bitmap.

You can load a graphic at design time from the Properties window. At run time, you can set this property by using the **LoadPicture** function on a bitmap, or you can use Clipboard methods such as **GetData**, **SetData**, and **GetFormat** with the nontext Clipboard formats **vbCFBitmap** and **vbCFDIB** as defined in the Visual Basic (VB) object library in the Object Browser.

When setting the Picture property at design time, the graphic is saved and loaded with the form. If you create an executable file, the file contains the image. When you load a graphic at run time, the graphic is not saved with the application. Use the **SavePicture** statement to save a graphic from a form or picture box into a file.

**Note** At run time, you can set the Picture property to any other object's Picture or Image property, or you can assign it the graphic returned by the **LoadPicture** function. The Picture property can only be assigned directly.

**Data Type**

**Integer**

## 3D Option Button Control

Properties Methods Events Constants Error Messages

The 3D option button control emulates the standard Visual Basic option button control, which displays an option that can be turned on or off. This control also allows you to align three-dimensional text to the right or left of the option button.

**File Name**

THREED16.OCX, THREED32.OCX

**Class Name**

SSOption

**Remarks**

The 3D option button has several custom properties that allow you to adjust the three-dimensional appearance of the control. When you draw a 3D option button on a form, the custom properties for the control are remembered and used as a template for the next 3D option button that you create.

Since the three-dimensional gray scale look requires a background color of light gray, the BackColor property is not available with this control. In Visual Basic, this control should be used on forms that have the BackColor property set to light gray (&H00C0C0C0&).

---

**Distribution Note** When you create and distribute applications that use the 3D option button control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

---

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *Alignment | FontItalic | Index | Tag |
| Caption | FontName | Left | Top |
| Container | FontSize | MouseIcon | Value |
| DragIcon | FontStrikethru | MousePointer | Visible |
| DragMode | FontUnderline | Name | WhatsThisHelpID |
| Enabled | ForeColor | Object | Width |
| Font | Height | Parent | |
| *Font3D | HelpContextID | TabIndex | |
| FontBold | hWnd | TabStop | |

Value is the default value of the control.

**Note** The DragIcon, DragMode, HelpContextID, Index, and Parent properties are only available in Visual Basic. The Name property is the same as the CtlName property in Visual Basic 1.0.

## Events

All of the events for this control are listed in the following table. Events that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *Click | DragOver | KeyDown | KeyUp |
| *DblClick | GotFocus | KeyPress | LostFocus |
| DragDrop | | | |

**Note** The DragDrop and DragOver events are only available in Visual Basic.

**Methods**

All of the methods for this control are listed in the following table.

| | | |
|---|---|---|
| Drag | Refresh | ZOrder |
| Move | SetFocus | ShowWhatsThis |

**Note** The **Drag**, **SetFocus**, and **ZOrder** methods are only available in Visual Basic.

## DblClick Event, 3D Option Button Control

Occurs when the user presses and then releases a mouse button, then presses it again over an option button. You can trigger the DblClick event in code by setting the control's Value property to **True**.

### Syntax

**Private Sub** *OptionButton3d_***DblClick** (*Value* **As Integer**)

### Remarks

This event is the same as the standard Visual Basic DblClick event, except that the control's *Value* is passed as an argument. When the option button is selected, *Value* = **True**. When it is not selected, *Value* = **False**.

## 3D Panel Control

Properties    Methods    Events    Constants    Error Messages

You can use the 3D panel control to display plain or three-dimensional text on a three-dimensional background, to group other controls on a three-dimensional background as an alternative to the frame control, or to lend a three-dimensional appearance to standard controls such as list boxes, combo boxes, scroll bars, and so on.

### File Name

THREED16.OCX, THREED32.OCX

### Class Name

SSPanel

### Remarks

The 3D panel is a three-dimensional rectangular area of variable size that can be as large as the form itself or just large enough to display a single line of text. It can present status information in a dynamically colored circle or bar with or without showing percent. (See the FloodShowPct property.)

While you can create some dramatic effects with the 3D panel, the control only has four basic visual properties: OuterBevel, InnerBevel, BevelWidth, and BorderWidth. By combining these properties in different ways, you can generate interesting backgrounds for text and controls.

Unlike most 3D controls, the 3D panel has a BackColor property. It defaults to light gray but can be changed to any color you choose. When used sparingly, the BackColor property can give presentation panels additional impact without getting in the way of the form's usefulness.

Like frames, 3D panels can have other controls placed on them.

The 3D panel has several custom properties that allow you to adjust the three-dimensional appearance of the control. When you draw a 3D panel on a form, the custom property settings for the control are saved and used as a template for the next 3D panel that you create.

### Bound Properties

The 3D panel has three bound properties: DataChanged, DataField, and DataSource. This means that it can be linked to a data control and display field values for the current record in the recordset. The 3D panel control can also write out values to the recordset.

When the value of the field referenced by the DataField property is read, it is converted to a Caption property string, if possible. If the recordset is updatable, the string is converted to the data type of the field.

For more information on using bound controls, refer to Chapter 22, "Accessing Databases With the Data Control," in the *Programmer's Guide*.

---

**Distribution Note**  When you create and distribute applications that use the 3D panel control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Wizard included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

---

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| Align | DataSource | FontName | Negotiate |
| *Alignment | DragIcon | FontSize | Object |
| *AutoSize | DragMode | FontStrikethru | *Outline |
| BackColor | Enabled | FontUnderline | Parent |
| *BevelInner | *FloodColor | ForeColor | *RoundedCorners |
| *BevelOuter | *FloodPercent | Height | *ShadowColor |
| *BevelWidth | *FloodShowPct | hWnd | TabIndex |
| *BorderWidth | *FloodType | Index | Tag |
| Container | Font | Left | Top |
| DataChanged | *Font3D | MouseIcon | Visible |
| DataField | FontBold | MousePointer | WhatsThisHelpID |
| | FontItalic | Name | Width |

Caption is the default value of the control.

**Note** The DragIcon, DragMode, HelpContextID, Index, and Parent properties are only available in Visual Basic. The Name property is the same as the CtlName property in Visual Basic 1.0.

The DataChanged, DataField, and DataSource properties are bound properties and are only available in Visual Basic 3.0.

**Events**

All of the events for this control are listed in the following table.

| | | | |
|---|---|---|---|
| Click | DragDrop | MouseDown | MouseUp |
| DblClick | DragOver | MouseMove | |

**Note** The DragDrop and DragOver events are only available in Visual Basic.

## Methods

All of the methods for this control are listed in the following table.

| | | | |
|---|---|---|---|
| Drag | Move | Refresh | Zorder |
| ShowWhatsThis | | | |

**Note** The **Drag** and **ZOrder** methods are only available in Visual Basic.

## BevelInner Property, 3D Panel Control

Determines the style of the inner bevel of the panel.

**Syntax**

[*form*.]*Panel3d*.**BevelInner**[ = *setting%*]

**Remarks**

The following table lists the BevelInner property settings for the 3D panel control.

| Setting | Description |
|---|---|
| 0 | (Default) None. No inner bevel is drawn. |
| 1 | Inset. The inner bevel appears inset on the screen. |
| 2 | Raised. The inner bevel appears raised off the screen. |

Use this property with the BevelOuter, BorderWidth, and BevelWidth properties.

**Data Type**

**Integer** (Enumerated)

## BevelOuter Property, 3D Panel Control

Determines the style of the outer bevel of the panel.

**Syntax**

[*form*.]*Panel3d*.**BevelOuter**[ = *setting%*]

**Remarks**

The following table lists the BevelOuter property settings for the 3D panel control.

| Setting | Description |
|---|---|
| 0 | None. No outer bevel is drawn. |
| 1 | Inset. The outer bevel appears inset on the screen. |
| 2 | (Default) Raised. The outer bevel appears raised off the screen. |

Use this property with the BevelInner, BorderWidth, and BevelWidth properties.

**Data Type**

**Integer** (Enumerated)

## BorderWidth property, 3D Panel Control

Sets or returns the width of the border, which is the distance between the outer and inner bevels of the panel.

**Syntax**

[*form*.]*Panel3d*.**BorderWidth**[ = *width%*]

**Remarks**

The setting for this property determines the number of pixels between the inner and outer bevels that surround the panel.

Border width can be set to a value between 0 and 30, inclusive.

Use this property in conjunction with the BevelInner, BevelOuter, and BevelWidth properties.

**Data Type**

**Integer**

## FloodColor Property, 3D Panel Control

Sets or returns the color used to paint the area inside the panel's inner bevel when the 3D panel is used as a status or progress indicator (that is, when the FloodType property setting is other than none).

### Syntax

[*form*.]*Panel3d*.**FloodColor**[ = *color&*]

### Remarks

The FloodColor property has the same range of settings as standard Visual Basic color settings.

| Setting | Description |
|---|---|
| Normal RGB colors | In Visual Basic, specified by using the Color palette, the RGB scheme, or **QBColor** functions in code. |
| System default colors | In Visual Basic, specified with system color constants listed in the object library in the Object Browser. |

Use this property with FloodPercent, FloodShowPct, and FloodType to cause the panel to display a colored status bar indicating the degree of completion of a task.

At design time you can set this property by entering a hexadecimal value in the Settings box or by clicking the three dots that appear at the right of the Settings box. Clicking this button displays a dialog box that allows you to select a FloodColor setting from a palette of colors similar to the Visual Basic Color Palette window.

**Note** The FloodColor property defaults to bright blue: RGB (0, 0, 255). The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower three bytes, from least to most significant, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF).

### Data Type

**Long**

## FloodPercent Property, 3D Panel Control

Example

Sets or returns the percentage of the painted area inside the panel's inner bevel when the panel is used as a status or progress indicator (that is, FloodType property setting other than none). This property is not available at design time.

### Syntax

[*form*.]*Panel3d*.**FloodPercent**[ = *percent%*]

### Remarks

The FloodPercent property can be set to an integer value between 0 and 100.

Use this property in conjunction with FloodColor, FloodShowPct, and FloodType to cause the panel to display a colored status bar, indicating the degree of completion of a task.

### Data Type

**Integer**

## FloodPercent Example, 3D Panel Control

### Visual Basic Example

The following example shows how the FloodPercent property updates the display of a panel status bar.

```
Private Sub Command1_Click ()
   Panel3d1.FloodPercent = 0       ' Init status
   Panel3d1.FloodType = 1          ' Left to right
   ' Do some long running process and update status bar at 10%
   ' intervals.
   For I% = 1 To 10
      DoLongRunningProcess
      Panel3d1.FloodPercent = I% * 10
      a% = DoEvents()              ' Let Windows do other operations.
   Next I%
End Sub
```

## FloodShowPct Property, 3D Panel Control

Determines whether the current setting of the FloodPercent property will be displayed in the center of the panel when the panel is used as a status or progress indicator (that is, FloodType property setting is other than none).

### Syntax

[*form*.]*Panel3d*.**FloodShowPct**[ = {**True** | **False**}]

### Remarks

The following table lists the FloodShowPct property settings for the 3D panel control.

| Setting | Description |
|---|---|
| **True** | (Default) The current setting of the FloodPercent property will be displayed. |
| **False** | The current setting of the FloodPercent property will not be displayed. |

### Data Type

**Integer** (Boolean)

## FloodType Property, 3D Panel Control

Determines if and how the panel is used as a status or progress indicator.

**Syntax**

[*form*.]*Panel3d*.**FloodType**[ = *setting%*]

**Remarks**

The following table lists the FloodType property settings for the 3D panel control.

| Setting | Description |
|---|---|
| 0 | (Default) None. Panel has no status bar capability and the caption (if any) is displayed. |
| 1 | Left to right. Panel will be painted in a color, which is specified by the FloodColor property, from the left inner bevel to the right as the FloodPercent property increases. |
| 2 | Right to left. Panel will be painted in a color, which is specified by the FloodColor property, from the right inner bevel to the left as the FloodPercent property increases. |
| 3 | Top to bottom. Panel will be painted in a color, which is specified by the FloodColor property, from the top inner bevel downward as the FloodPercent property increases. |
| 4 | Bottom to top. Panel will be painted in a color, which is specified by the FloodColor property, from the bottom inner bevel upward as the FloodPercent property increases. |
| 5 | Widening circle. Panel will be painted in a color, which is specified by the FloodColor property, from the center outward in a widening circle as the FloodPercent property increases. |

**Note** If the FloodType setting is a value other than 0, the panel caption (if any) will not be displayed.

**Data Type**

**Integer** (Enumerated)

## Animated Button Control

Properties Methods Events Constants Error Messages

The animated button control is a flexible button control that allows you to use any icon, bitmap, or metafile to define your own button controls. Control types include animated buttons, multistate buttons, and animated check boxes.

**File Name**

ANIBTN16.OCX, ANIBTN32.OCX

**Class Name**

AniPushButton

**Remarks**

Each animated button can contain zero or more images and an optional text caption. An animated button can be thought of as a series of frames that are displayed in sequence.

You can use the Picture property to load images into the animated button control. The Frame property indicates which picture is currently accessible through the Picture property. In other words, the Frame property is an index of the array of images in the control.

The images are displayed within the control's border. The default is to display the images in the center of the control, but you can use the PictureXpos and PictureYpos properties to position the image within the control. You can also use the PictDrawMode property to scale the image to the exact size of the control or to adjust the control to the size of your image.

The Caption text can be displayed next to the images or on the images, depending on the TextPosition property.

**Distribution Note** When you create and distribute applications that use the animated button control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

**Animation Cycles and Button Types**

The following table shows how you can use frame sequences to implement various types of animated buttons.

| Button type | Cycle | Description |
|---|---|---|
| Animated | 0 | When the left mouse button is clicked, half of the frames are displayed in order. When the button is released, the remaining frames are displayed in order, returning to the first frame. |
| Multistate | 1 | Each frame specifies a particular state. When the left button is clicked, it automatically switches to the next state and displays the appropriate frame. |
| 2-state animated | 2 | When the left button is clicked, frames are displayed in sequential order until the middle frame appears, and the state is changed to 2 (that is, checked).<br>When the button is clicked again, the remaining frames are displayed, returning to the first frame. The state is changed back to 1. |
| Enhanced button | 0 | An animated button with only two frames. |
| Enhanced check box | 1 | A multistate button with two frames. |

It is possible to pass Clipboard images directly into animated button frames. When loading frames, it is also possible to pass Windows metafiles; images are scaled to the control and then converted into bitmaps.

**Note** The animated button control is generally used to create small- to medium-sized buttons. However, the control is capable of holding large bitmaps. Bitmaps and icons held in an animated button control use few Windows resources. The data is stored in global memory in a private format and does not use Windows bitmap or icon resource handles. The animated button control is a useful tool for archiving bitmaps or icons.

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| BackColor | Font | hWnd | *SpecialOp |
| BorderStyle | FontBold | Index | *Speed |
| Caption | FontItalic | Left | TabIndex |
| *CCBfileLoad | FontName | MouseIcon | TabStop |
| *CCBfileSave | FontSize | MousePointer | Tag |
| *ClearFirst | FontStrikethru | Name | *TextPosition |
| *ClickFilter | FontUnderline | Object | *TextXpos |
| Caption | ForeColor | Parent | *TextYpos |
| *Cycle | *Frame | *PictDrawMode | Top |
| DragIcon | Height | *Picture | *Value |
| DragMode | HelpContextID | *PictureXpos | Visible |
| Enabled | *HideFocusBox | *PictureYpos | WhatsThisHelpID |
| | | | Width |

Value is the default value of the control.

**Note** The Name property is the same as the CtlName property in Visual Basic 1.0.

**Events**

All of the events for this control are listed in the following table. Events that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *Click | DragOver | KeyDown | KeyUp |
| DragDrop | GotFocus | KeyPress | LostFocus |

**Methods**

All of the methods for this control are listed in the following table. For documentation of the methods not unique to this control, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

| | | |
|---|---|---|
| Drag | Refresh | ZOrder |
| Move | SetFocus | ShowWhatsThis |

## CCBfileLoad Property, Animated Button Control

Loads image and animated button property information from files previously saved with the CCBfileSave property. This property is write-only.

**Syntax**

[*form*.]*AniButton.***CCBfileLoad** = *filename$*

**Remarks**

All animated button files have the extension .CCB.

CCB files save only image information and animated button property information. Except for the BorderStyle property, information for standard properties is not saved in these files. If you want to save all of the information for an animated button control, place it on a form and save the form. In App Studio, place the control on a dialog and save the dialog. You can also copy controls using the Clipboard.

You can type the name of the file directly or click the ellipsis (...) to the right of the Settings box to open a CCBfileLoad dialog box.

Animated button CCB files are fully compatible with Desaware's Custom Control Factory and can be used to transfer frame sequences to and from Custom Control Factory controls.

**Data Type**

**String**

## CCBfileSave Property, Animated Button Control

Saves information for an animated button control in a file. This property is write-only.

**Syntax**

[*form*.]*AniButton.***CCBfileSave** = *filename$*

**Remarks**

The name of the CCB file to save is indicated by the placeholder *filename$.* All animated button files have the extension .CCB.

You can save image and property information into CCB files that can then be distributed or used to build a library of animated button controls. These files save only image and animated button property information. Except for the BorderStyle property, information for standard properties is not saved in the CCB files. If you want to save all of the information for an animated button control, place it on a form and save the form. In App Studio, place the control on a dialog and save the dialog. You can also use the Clipboard to copy controls.

You can type in the name of the file directly or click the ellipsis (...) to the right of the Settings box to open a CCBfileSave dialog box.

Animated button CCB files are fully compatible with Desaware's Custom Control Factory and can be used to transfer frame sequences to and from Custom Control Factory controls.

**Data Type**

**String**

## ClearFirst Property, Animated Button Control

Determines whether the control is cleared between frames.

**Syntax**

[*form.*]*AniButton.***ClearFirst**[ = {**True | False**}]

**Remarks**

Normally, button controls are animated by drawing a new frame right on top of a previous frame. This produces a smooth animation effect when either the image is stable or changes are gradual.

If you animate an image with large changes (for example, if an object is moving rapidly), an illusion of tearing may occur when part of the old image and part of the new image are on the screen at the same time.

Setting ClearFirst to **True** causes the control to be cleared between frames. This eliminates the tearing effect; however, it does tend to cause increased flicker between frames. Try the control both ways to determine which produces the best effect.

The following table lists the ClearFirst property settings for the animated button control.

| Setting | Description |
|---|---|
| **False** | (Default) ClearFirst feature disabled. |
| **True** | ClearFirst feature enabled. |

**Data Type**

**Integer** (Boolean)

## ClickFilter Property, Animated Button Control

Determines what part of the animated button control detects a mouse click.

**Syntax**

[*form*.]*AniButton.***ClickFilter**[ = *setting%*]

**Remarks**

The following table lists the ClickFilter property settings for the animated button control.

| Setting | Description |
|---|---|
| 0 | (Default) Mouse clicks are detected anywhere in the control. |
| 1 | Mouse clicks must be on either the caption text or the actual image frame in order to be detected. |
| 2 | Mouse clicks must be on the image frame in order to be detected. |
| 3 | Mouse clicks must be on the caption text in order to be detected. |

All mouse clicks on parts of the window that are not specified will be ignored. The animated button invokes a Click event when a mouse click is detected.

**Data Type**

**Integer** (Enumerated)

## Cycle Property, Animated Button Control

Controls the animation cycle and differentiates between animated, multistate, and 2-state animated buttons.

### Syntax

[*form*.]*AniButton*.**Cycle**[ = *setting%*]

### Remarks

The following table lists the Cycle property settings for the animated button control.

| Setting | Description |
|---|---|
| 0 | (Default) Plays one half of the frame sequence when the user chooses (clicks) the button. Plays the rest of the frame sequence when the button is released. Returns to the first frame. |
| 1 | Jumps to the next frame in the sequence when the button is released. Increments the Value property at this time. This implements a one-frame-per-state multistate button. Clicking the button when the button is set to the last frame (last state) causes the button to return to the first frame (first state). |
| 2 | Plays one half of the frame sequence when the user chooses (clicks) the button for the first time. This sets the Value property to 2 (from 1). When the button is clicked again, the remaining frames will be played and the button will return to frame 1. At this time the Value property will be set back to 1. This implements a 2-state animated button. |

The Cycle property affects only the display sequence of images. The Click event occurs when the mouse button is released. Pressing the SPACEBAR when a button has the focus causes the button to be selected and released (as if it were clicked by the mouse).

### Data Type

**Integer** (Enumerated)

## Frame Property, Animated Button Control

Example

Indicates the current frame.

### Syntax

[*form*.]*AniButton.***Frame**[ = *setting%*]

### Remarks

The frame property has the following effects:

- The current frame is the frame displayed while in design mode.
- The current frame is the frame that can be accessed using the Picture property (in both design and run modes under program control).

The Frame property has no effect on the appearance of the control at run time. It still can be set to choose the frame to set or retrieve using the Picture property.

The Frame property can have the values one through the number of frames plus one. The argument *setting%* is the number of the individual frame that is displayed in design mode and that can be accessed in both design and run mode.

### Data Type

**Integer**

## Frame Property Example, Animated Button Control

The following example shows how to determine the number of frames in an animated button control at run time.

```
Private Sub Form_Click ()
   Dim a%, done%
   ' This will hold the frame number.
   a% = 1
   ' This flag tells us when done.
   done% = 0
   On Error GoTo foundprop
   Do
      ' Buttons CtlName property here.
      AniButton1.frame = a%
      ' Done. a% contains the number of
      ' the frame that caused the error.
      If done% Then Exit Do
      a% = a% + 1
   Loop While - 1
   ' Calculate the actual number of images.
   ' a% - 1 is the empty trailing frame.
      a% = a% - 1
   Exit Sub
   FoundProp:
      done% = -1
      Resume Next
End Sub
```

## HideFocusBox Property, Animated Button Control

Normally, when an animated button has the focus, a dotted-line rectangle appears around the caption (or around the image if no caption is present).

There are occasions, however, when the focus rectangle might interfere with the animation. To prevent the focus rectangle from appearing, set this property to **True**.

### Syntax

[*form.*]*AniButton.***HideFocusBox**[ = {**True | False**}]

### Remarks

The following table lists the HideFocusBox property settings for the animated button control.

| Setting | Description |
|---|---|
| **False** | (Default) Focus rectangle appears when the control has the focus. |
| **True** | Focus rectangle is hidden when the control has the focus. |

### Data Type

**Integer** (Boolean)

## PictDrawMode Property, Animated Button Control

Defines how the image frame is drawn within the control. It is possible for any given image frame (bitmap or icon) to be smaller or larger than the control.

### Syntax

[*form*.]*AniButton.***PictDrawMode**[ = *setting%*]

### Remarks

The following table lists the PictDrawMode property settings for the animated button control.

| **Setting** | **Description** |
|---|---|
| 0 | (Default) Positions the image according to the values in the PictureXpos and PictureYpos properties and places the caption according to the TextPosition property value. These properties control the X and Y position on a scale of 0 to 100. |
| 1 | Automatically controls the sizing mode. The animated button control is sized to fit the largest image frame or the caption, whichever is largest. |
| 2 | Stretches the image to fit. The image frame is expanded or contracted to fill the current size of the control. In this mode, the caption (if present) is always printed as if the TextPosition property were set to 0 (that is, displayed on top of the image). |

### Data Type

**Integer** (Enumerated)

## Picture Property, Animated Button Control

You can use this property to set and get the image frames in the control. In design mode, you can click the ellipsis (...) to the right of the Settings box to open the Load Picture dialog box.

You can use this property to transfer images between forms and picture controls and the animated button control. This is done by assignment in the same way that images can be transferred using the Picture property in forms and picture controls. For example:

```
Form.Picture = Anibutton1.Picture.
```

The image frame that is accessed with this property is always the image specified by the Frame property.

## PictureXpos Property, Animated Button Control

Controls the horizontal placement of the image in the control.

**Syntax**

[*form*.]*AniButton.***PictureXpos**[ = *setting%*]

**Remarks**

The value of this property can vary from 0 to 100, inclusive. The value represents the percentage placement from the upper-left corner of the control. Thus, a value of 0 places the image at the upper-left corner of the control; a value of 100 places it at the lower-right corner of the control. The default value is 50. Refer to the TextPosition property for details on how the behavior of this property may be modified by the positioning of the caption.

**Data Type**

**Integer**

## PictureYpos Property, Animated Button Control

Controls the vertical placement of the image in the control.

**Syntax**

[*form*.]*AniButton.***PictureYpos**[ = *setting%*]

**Remarks**

The value of this property can vary from 0 to 100, inclusive. The value represents the percentage placement from the upper-left corner of the control. Thus, a value of 0 places the image at the upper-left corner of the control; a value of 100 places it at the lower-right corner of the control. The default value is 50. Refer to the TextPosition property for details on how the behavior of this property may be modified by the positioning of the caption.

**Data Type**

**Integer**

## SpecialOp Property, Animated Button Control

Triggers special operations on the part of the animated button control. A special operation is triggered by assigning a value to this property at run time. This property is not available at design time and is write-only at run time.

### Syntax

[*form*.]*AniButton.***SpecialOp** = *setting%*

### Remarks

The following table lists the SpecialOp property settings for the animated button control.

| **Setting** | **Description** |
|---|---|
| 1 | Simulates a click. The control behaves exactly as if it had been clicked. The control receives the focus and the form is activated if necessary. This option will not work if the button's Enabled property is **False**. This option has no effect if the control's Visible property is set to **False**. |
| Any other value | No effect. No error is reported. |

### Data Type

**Integer**

## Speed Property, Animated Button Control

Specifies the approximate delay, in milliseconds, between frames.

**Syntax**

[*form.*]*AniButton.***Speed**[ = *setting%*]

**Remarks**

Enter a value between 0 and 32767, inclusive. The default value is 0.

Larger numbers slow down the animation speed, and using very large numbers with this property significantly impacts system performance. For best results, choose values below 100.

**Data Type**

**Integer**

## TextPosition Property, Animated Button Control

Controls the position of the caption in the control. By doing so, it also influences the position of the image.

**Syntax**

[*form*.]*AniButton*.**TextPosition**[ = *setting%*]

**Remarks**

The following table lists the TextPosition property settings for the animated button control.

| Setting | Description |
|---|---|
| 0 | (Default) Caption is positioned within the control based on the TextXpos and TextYpos properties. The image is positioned according to the PictDrawMode, PictureXpos, and PictureYpos properties. |
| 1 | Image is placed at the left of the control. The TextXpos property positions the caption within the space between the rightmost position of the image and the rightmost position of the control. The vertical position is determined the same as when the TextPosition property is 0. |
| 2 | Image is placed at the right of the control. The TextXpos property positions the caption within the space between the leftmost position of the image and the leftmost position of the control. The vertical position is determined the same as when the TextPosition property is 0. |
| 3 | Image is placed at the bottom of the control. The TextYpos property positions the caption within the space between the top of the image and the top of the control. The horizontal position is determined the same as when the TextPosition property is 0. |
| 4 | Image is placed at the top of the control. The TextYpos property positions the caption within the space between the bottom of the image and the bottom of the control. The horizontal position is determined the same as when the TextPosition property is 0. |

**Note** When the PictDrawMode property is 2, the image and caption positions are the same as when the TextPosition property is 0.

**Data Type**

**Integer** (Enumerated)

## TextXpos Property, Animated Button Control

Controls the horizontal placement of the text caption.

**Syntax**

[*form.*]*AniButton.***TextXpos**[ = *setting%*]

**Remarks**

The value of this property can vary from 0 to 100, inclusive. The value represents the percentage placement from the upper-left corner of the caption area in the control. Thus, a value of 0 places the caption at the upper-left corner of the caption area; a value of 100 places it at the lower-right corner of the caption area. The default value is 50.

The caption area refers to the part of the control reserved for the text caption. This depends on which setting you use for the TextPosition property, as described in the following table.

| Setting | Description |
|---|---|
| 0 | Caption area is entire control. Caption overlays any images in the control. |
| 1 | Caption placed to the right of the image. |
| 2 | Caption placed to the left of the image. |
| 3 | Caption placed above the image. |
| 4 | Caption placed below the image. |

**Data Type**

**Integer** (Enumerated)

## TextYpos Property, Animated Button Control

Controls the vertical placement (TextYpos) of the text caption.

**Syntax**

[*form*.]*AniButton.***TextYpos**[ = *setting%*]

**Remarks**

The value of this property can vary from 0 to 100, inclusive. The value represents the percentage placement from the upper-left corner of the caption area in the control. Thus, a value of 0 places the caption at the upper-left corner of the caption area; a value of 100 places it at the lower-right corner of the caption area. The default value is 50.

The caption area refers to the part of the control reserved for the text caption. This depends on which setting you use for the TextPosition property, as described in the following table.

| Setting | Description |
|---|---|
| 0 | Caption area is entire control. Caption overlays any images in the control. |
| 1 | Caption placed to the right of the image. |
| 2 | Caption placed to the left of the image. |
| 3 | Caption placed above the image. |
| 4 | Caption placed below the image. |

**Data Type**

**Integer** (Enumerated)

## Value Property, Animated Button Control

Indicates the state of a 2-state or multistate animated button. Refer to the Cycle property for how this property works for the different button and animation modes.

**Syntax**

[*form*.]*AniButton*.**Value**[ = *setting%*]

**Remarks**

This property can be retrieved to determine the current frame number of an animated button control. When the Cycle property is set to 1, you can use the Value property to specify the frame of the cycle you want to display.

When the Value property of a control is changed, the display may not be updated until subsequent events have occurred (such as the **DoEvents()** function).

Setting the Value property of a control does not cause a Click event to occur.

**Data Type**

**Integer** (Enumerated)

## Click Event, Animated Button Control

Occurs when the user presses and then releases a mouse button over an animated button.

**Syntax**

**Private Sub** *AniButton_***Click** ( )

**Remarks**

This event is the same as the standard Visual Basic Click event, except that it is not generated when the user presses Enter. You can use a KeyPress event to detect when the user presses Enter.

## Communications Control

Properties Events Functions Example Constants Error Messages

The communications control provides serial communications for your application by allowing the transmission and reception of data through a serial port.

**File Name**

MSCOMM16.OCX, MSCOMM32.OCX

**Class Name**

MSComm

**Remarks**

The communications control provides the following two ways for handling communications:

- Event-driven communications is a very powerful method for handling serial port interactions. In many situations you want to be notified the moment an event takes place, as when a character arrives or a change occurs in the Carrier Detect (CD) or Request To Send (RTS) lines. In such cases, you would use the communications control's OnComm event to trap and handle these communications events. The OnComm event also detects and handles communications errors. For a list of all possible events and communications errors, see the CommEvent property.
- You can also poll for events and errors by checking the value of the CommEvent property after each critical function of your program. This may be preferable if your application is small and self-contained. For example, if you are writing a simple phone dialer, it may not make sense to generate an event after receiving every character, because the only characters you plan to receive are the OK response from the modem.

Each communications control you use corresponds to one serial port. If you need to access more than one serial port in your application, you must use more than one communications control. The port address and interrupt address can be changed from the Windows Control Panel.

Although the communications control has many important properties, there are a few that you should be familiar with first.

| Properties | Description |
|---|---|
| CommPort | Sets and returns the communications port number. |
| Settings | Sets and returns the baud rate, parity, data bits, and stop bits as a string. |
| PortOpen | Sets and returns the state of a communications port. Also opens and closes a port. |
| Input | Returns and removes characters from the receive buffer. |
| Output | Writes a string of characters to the transmit buffer. |

**Distribution Note** When you create and distribute applications that use the communications control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

## Communications Control Example

The following simple example shows how to perform basic serial port communications:

```
Private Sub Form_Load ()
   ' Use COM1.
   Comm1.CommPort = 1
   ' 9600 baud, no parity, 8 data, and 1 stop bit.
   Comm1.Settings = "9600,N,8,1"
   ' Tell the control to read entire buffer when Input is used.
   Comm1.InputLen = 0
   ' Open the port.
   Comm1.PortOpen = True
   ' Send the attention command to the modem.
   Comm1.Output = "AT" + Chr$(13)
   ' Wait for data to come back to the serial port.
   Do
      Dummy = DoEvents()
   Loop Until Comm1.InBufferCount >= 2
   ' Read the "OK" response data in the serial port.
   InString$ = Comm1.Input
   ' Close the serial port.
   Comm1.PortOpen = False
End Sub
```

## Properties

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *Break | *DSRTimeout | Left | *PortOpen |
| *CDHolding | *DTREnable | Name | *RThreshold |
| *CDTimeout | *Handshaking | *NullDiscard | *RTSEnable |
| *CommEvent | *InBufferCount | Object | *Settings |
| *CommID | *InBufferSize | *OutBufferCount | *SThreshold |
| *CommPort | Index | *OutBufferSize | Tag |
| *CTSHolding | *Input | *Output | Top |
| *CTSTimeout | *InputLen | Parent | |
| *DSRHolding | *Interval | *ParityReplace | |

Input is the default value of the control.

**Note** The Name property is the same as the CtlName property in Visual Basic 1.0.

### Events

All of the events for this control are listed in the following table. Events that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

*OnComm

### Functions

All of the functions for this control are listed in the following table. Functions that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | |
|---|---|
| *ComInput | *ComOutput |

## Break Property, Communications Control

Example

Sets or clears the break signal state. This property is not available at design time.

### Syntax

[*form*.]*MSComm*.**Break**[ = {**True | False**}]

### Remarks

The following table lists the Break property settings for the communications control.

| Setting | Description |
|---|---|
| **True** | Sets the break signal state. |
| **False** | Clears the break signal state. |

When set to **True**, the Break property sends a break signal. The break signal suspends character transmission and places the transmission line in a break state until you set the Break property to **False**.

Typically, you set the break state for a short interval of time, and *only* if the device with which you are communicating requires that a break signal be set.

### Data Type

**Integer** (Boolean)

Close Copy Print

## Break Example, Communications Control

The following example shows how to send a break signal for a tenth of a second:

```
' Set the Break condition.
Comm1.Break = True
' Set duration to 1/10 second.
Duration! = Timer + .1
' Wait for the duration to pass.
Do Until Timer > Duration!
   Dummy = DoEvents()
Loop
' Clear the Break condition.
Comm1.Break = False
```

## CDHolding Property, Communications Control

Determines whether the carrier is present by querying the state of the Carrier Detect (CD) line. Carrier Detect is a signal sent from a modem to the attached computer to indicate that the modem is online. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MSComm*.**CDHolding**

### Remarks

The following table lists the CDHolding property settings for the communications control.

| Setting | Description |
|---|---|
| **True** | Carrier Detect line is high. |
| **False** | Carrier Detect line is low. |

When the Carrier Detect line is high (CDHolding = **True**) and the CDTimeout number of milliseconds has passed, the communications control sets the CommEvent property to **comCDTO** (Carrier Detect Timeout Error), and generates the OnComm event.

**Note** It is especially important to trap a loss of the carrier in a host application, such as a bulletin board, because the caller can hang up (dropping the carrier) at any time.

The Carrier Detect is also known as the Receive Line Signal Detect (RLSD).

See the CDTimeout property for information on trapping this condition using the OnComm event.

### Data Type

**Integer** (Boolean)

## CDTimeout Property, Communications Control

Sets and returns the maximum amount of time (in milliseconds) that the control waits for the Carrier Detect (CD) signal before timing out. This property indicates timing out by setting the CommEvent property to **comCDTO** (Carrier Detect Timeout Error) and generating the OnComm event.

### Syntax

[*form*.]*MSComm*.**CDTimeout**[ = *milliseconds&*]

### Remarks

**Note** The 32-bit version of this control (MSCOMM32.OCX) doesn't support this property.

When the Carrier Detect line is low (CDHolding = **False**) and CDTimeout number of milliseconds has passed, the communications control sets the CommEvent property to **comCDTO** (Carrier Detect Timeout Error) and generates the OnComm event.

Refer to the CDHolding property for more information on detecting the presence of a carrier.

### Data Type

**Long**

## CommEvent Property, Communications Control

Returns the most recent communication event or error. This property is not available at design time and is read-only at run time.

**Syntax**

[*form*.]*MSComm*.**CommEvent**

**Remarks**

Although the OnComm event is generated whenever a communication error or event occurs, the CommEvent property holds the numeric code for that error or event. To determine the actual error or event that caused the OnComm event, you must reference the CommEvent property.

The code returned by the CommEvent property is one of the settings of the following communication errors or events, as specified in the object library in the Object Browser.

Communications errors include the following settings.

| Setting | Value | Description |
|---|---|---|
| **comBreak** | 1001 | A Break signal was received. |
| **comCTSTO** | 1002 | Clear To Send Timeout. The Clear To Send line was low for CTSTimeout number of milliseconds while trying to transmit a character. |
| **comDSRTO** | 1003 | Data Set Ready Timeout. The Data Set Ready line was low for DSRTimeout number of milliseconds while trying to transmit a character. |
| **comFrame** | 1004 | Framing Error. The hardware detected a framing error. |
| **comOverrun** | 1006 | Port Overrun. A character was not read from the hardware before the next character arrived and was lost. If you get this error under Windows version 3.0, decrease the value of the Interval property. For more details, refer to the Interval property. |
| **comCDTO** | 1007 | Carrier Detect Timeout. The Carrier Detect line was low for CDTimeout number of milliseconds while trying to transmit a character. Carrier Detect is also known as the Receive Line Signal Detect (RLSD). |
| **comRxOver** | 1008 | Receive Buffer Overflow. There is no room in the receive buffer. |
| **comRxParity** | 1009 | Parity Error. The hardware detected a parity error. |
| **comTxFull** | 1010 | Transmit Buffer Full. The transmit buffer was full while trying to queue a character. |

Communications events include the following settings.

| Setting | Value | Description |
|---|---|---|
| **comEvSend** | 1 | There are fewer than SThreshold number of characters in the transmit buffer. |
| **comEvReceive** | 2 | Received RThreshold number of characters. This event is generated continuously until you use the Input property to remove the data from the receive buffer. |
| **comEvCTS** | 3 | Change in Clear To Send line. |
| **comEvDSR** | 4 | Change in Data Set Ready line. This event is only fired when DSR changes from ■1 to 0. |
| **comEvCD** | 5 | Change in Carrier Detect line. |
| **comEvRing** | 6 | Ring detected. Some UARTs (universal asynchronous receiver-transmitters) may not support this event. |
| **comEvEOF** | 7 | End Of File (ASCII character 26) character received. |

**Data Type**

**Integer**

## CommID Property, Communications Control

Returns a handle that identifies the communications device. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MSComm*.**CommID**

### Remarks

This is the value returned by the Windows API **OpenComm** function and used by the internal communications routines in the Windows API.

### Data Type

**Integer**

## CommPort Property, Communications Control

Sets and returns the communications port number.

### Syntax

[*form*.]*MSComm*.**CommPort**[ = *portNumber%*]

### Remarks

You can set *portNumber* to any number between 1 and 99 at design time (the default is 1). However, the communications control generates error 68 (Device unavailable) if the port does not exist when you attempt to open it with the PortOpen property.

**Warning** You must set the CommPort property before opening the port.

### Data Type

**Integer**

## CTSHolding Property, Communications Control

Determines whether you can send data by querying the state of the Clear To Send (CTS) line. Typically, the Clear To Send signal is sent from a modem to the attached computer to indicate that transmission can proceed. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MSComm*.**CTSHolding**

### Remarks

The following table lists the CTSHolding property settings for the communications control.

| Setting | Description |
|---|---|
| **True** | Clear To Send line high. |
| **False** | Clear To Send line low. |

When the Clear To Send line is low (CTSHolding = **False**) and the CTSTimeout number of milliseconds has passed, the communications control sets the CommEvent property to **comCTSTO** (Clear To Send Timeout) and invokes the OnComm event.

The Clear To Send line is used in RTS/CTS (Request To Send/Clear To Send) hardware handshaking. The CTSHolding property gives you a way to manually poll the Clear To Send line if you need to determine its state.

For more information on handshaking protocols, see the Handshaking property.

### Data Type

**Integer** (Boolean)

## CTSTimeout Property, Communications Control

Sets and returns the number of milliseconds to wait for the Clear To Send signal before setting the CommEvent property to **comCTSTO** and generating the OnComm event.

### Syntax

[*form*.]*MSComm*.**CTSTimeout**[ = *milliseconds&*]

### Remarks

**Note** The 32-bit version of this control (MSCOMM32.OCX) doesn't support this property.

When the Clear To Send line is low (CTSHolding = **False**) and the CTSTimeout number of milliseconds has passed, the communications control sets the CommEvent property to **comCTSTO** (Clear To Send Timeout) and generates the OnComm event.

See the CTSHolding property, which gives you a means to manually poll the Clear To Send line.

### Data Type

**Long**

## DSRHolding Property, Communications Control

Determines the state of the Data Set Ready (DSR) line. Typically, the Data Set Ready signal is sent by a modem to its attached computer to indicate that it is ready to operate. This property is not available at design time.

### Syntax

[*form*.]*MSComm*.**DSRHolding**[ = *setting*]

### Remarks

The following table lists the DSRHolding property settings for the communications control.

| Setting | Description |
|---|---|
| **True** | Data Set Ready line high. |
| **False** | Data Set Ready line low. |

When the Data Set Ready line is high (DSRHolding = **True**) and the DSRTimeout number of milliseconds has passed, the communications control sets the CommEvent property to **comDSRTO** (Data Set Ready Timeout) and invokes the OnComm event.

This property is useful when writing a Data Set Ready/Data Terminal Ready handshaking routine for a Data Terminal Equipment (DTE) machine.

### Data Type

**Integer** (Boolean)

## DSRTimeout Property, Communications Control

Sets and returns the number of milliseconds to wait for the Data Set Ready (DSR) signal before setting the CommEvent property to **comDSRTO** and generating the OnComm event.

### Syntax

[*form*.]*MSComm*.**DSRTimeout**[ = *milliseconds&*]

### Remarks

---

**Note** The 32-bit version of this control (MSCOMM32.OCX) doesn't support this property.

---

When the Data Set Ready line is high (DSRHolding = **True**) and the DSRTimeout number of milliseconds has passed, the communications control sets the CommEvent property to **comDSRTO** (Data Set Ready Timeout) and generates the OnComm event.

This property is useful when writing a Data Set Ready/Data Terminal Ready handshaking routine for a DTE machine.

See the DSRHolding property, which allows you to manually poll the Data Set Ready line.

### Data Type

**Long**

## DTREnable Property, Communications Control

Determines whether to enable the Data Terminal Ready (DTR) line during communications. Typically, the Data Terminal Ready signal is sent by a computer to its modem to indicate that the computer is ready to accept incoming transmission.

### Syntax

[*form*.]*MSComm*.**DTREnable**[ = {**True | False**}]

### Remarks

The following table lists the DTREnable property settings for the communications control.

| Setting | Description |
|---|---|
| **True** | Enable the Data Terminal Ready line. |
| **False** | (Default) Disable the Data Terminal Ready line. |

When DTREnable is set to **True**, the Data Terminal Ready line is set to high (on) when the port is opened, and low (off) when the port is closed. When DTREnable is set to **False**, the Data Terminal Ready always remains low.

**Note** In most cases, setting the Data Terminal Ready line to low hangs up the telephone.

### Data Type

**Integer** (Boolean)

## Handshaking Property, Communications Control

Sets and returns the hardware handshaking protocol.

**Syntax**

[*form*.]*MSComm*.**Handshaking**[ = *protocol%*]

**Remarks**

Handshaking refers to the internal communications protocol by which data is transferred from the hardware port to the receive buffer. When a character of data arrives at the serial port, the communications device has to move it into the receive buffer so that your program can read it. If there is no receive buffer and your program is expected to read every character directly from the hardware, you will probably lose data because the characters can arrive very quickly.

A handshaking protocol insures that data is not lost due to a buffer overrun, in which case data arrives at the port too quickly for the communications device to move the data into the receive buffer.

Valid protocols are listed in the following table.

| Setting | Value | Description |
|---|---|---|
| **comNone** | 0 | (Default) No handshaking. |
| **comXOnXOff** | 1 | XON/XOFF handshaking. |
| **comRTS** | 2 | RTS/CTS (Request To Send/Clear To Send) handshaking. |
| **comRTSXOnXOff** | 3 | Both Request To Send and XON/XOFF handshaking. |

**Data Type**

**Integer**

## InBufferCount Property, Communications Control

Returns the number of characters waiting in the receive buffer. This property is not available at design time.

**Syntax**

[*form*.]*MSComm*.**InBufferCount**[ = *count%*]

**Remarks**

InBufferCount refers to the number of characters that have been received by the modem and are waiting in the receive buffer for you to take them out. You can clear the receive buffer by setting the InBufferCount property to 0.

---

**Note** Do not confuse this property with the InBufferSize property − InBufferSize reflects the total size of the receive buffer.

---

**Data Type**

**Integer**

## InBufferSize Property, Communications Control

Sets and returns the size of the receive buffer in bytes.

### Syntax

[*form*.]*MSComm*.**InBufferSize**[ **=** *numBytes%*]

### Remarks

InBufferSize refers to the total size of the receive buffer. The default size is 1024 bytes. Do not confuse this property with the InBufferCount property − InBufferCount reflects the number of characters currently waiting in the receive buffer.

---

**Note** Note that the larger you make the receive buffer, the less memory you have available to your application. However, if your buffer is too small, it runs the risk of overflowing unless handshaking is used. As a general rule, start with a buffer size of 1024 bytes. If an overflow error occurs, increase the buffer size to handle your application's transmission rate.

---

### Data Type

**Integer**

## Input Property, Communications Control

Example

Returns and removes a string of characters from the receive buffer. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MSComm*.**Input**

### Remarks

The InputLen property determines the number of characters that are read by the Input property. Setting InputLen to 0 causes the Input property to read the entire contents of the receive buffer.

### Data Type

**String**

## Input Example, Communications Control

This example shows how to retrieve data from the receive buffer:

```
' Retrieve all available data.
Comm1.InputLen = 0
' Check for data.
If Comm1.InBufferCount Then
   ' Read data.
   InString$ = Comm1.Input
End If
```

## InputLen Property, Communications Control

Example

Sets and returns the number of characters the Input property reads from the receive buffer.

### Syntax

[*form*.]*MSComm*.**InputLen**[ = *numChars%*]

### Remarks

The default value for the InputLen property is 0. Setting InputLen to 0 causes the communications control to read the entire contents of the receive buffer when Input is used.

If InputLen characters are not available in the receive buffer, the Input property returns a zero-length string (""). The user can optionally check the InBufferCount property to determine if the required number of characters are present before using Input.

This property is useful when reading data from a machine whose output is formatted in fixed-length blocks of data.

### Data Type

**Integer**

## InputLen Example, Communications Control

This example shows how to read 10 characters of data:

```
' Specify a 10 character block of data.
Comm1.InputLen = 10
' Read data.
CommData$ = Comm1.Input
```

## Interval Property, Communications Control

Sets the interval, in milliseconds, for polling the hardware port for data under Windows version 3.0.

### Syntax

[*form*.]*MSComm*.**Interval**[ = *milliseconds&*]

### Remarks

The default value for the Interval property is 1000 (1 second).

You only need this property for applications that run under Windows graphical environment version 3.0, because the communications control has to manually poll the hardware port for data at a given interval. However, under Windows operating system version 3.1 this is not necessary, and you don't need to use the Interval property.

### Data Type

**Long**

## NullDiscard Property, Communications Control

Determines whether null characters are transferred from the port to the receive buffer.

**Syntax**

[*form*.]*MSComm*.**NullDiscard**[ = {**True** | **False**}]

**Remarks**

The following table lists the NullDiscard property settings for the communications control.

| Setting | Description |
|---|---|
| **True** | Null characters are *not* transferred from the port to the receive buffer. |
| **False** | (Default) Null characters are transferred from the port to the receive buffer. |

A null character is defined as ASCII character 0, Chr$(0).

**Data Type**

**Integer** (Boolean)

## OutBufferCount Property, Communications Control

Returns the number of characters waiting in the transmit buffer. You can also use it to clear the transmit buffer. This property is not available at design time.

### Syntax

[*form*.]*MSComm*.**OutBufferCount**[ = **0**]

### Remarks

You can clear the transmit buffer by setting the OutBufferCount property to 0.

---

**Note** Do not confuse the OutBufferCount property with the OutBufferSize property ■ OutBufferSize reflects the total size of the transmit buffer.

---

### Data Type

**Integer**

## OutBufferSize Property, Communications Control

Sets and returns the size, in characters, of the transmit buffer.

**Syntax**

[*form*.]*MSComm*.**OutBufferSize**[ = *NumBytes%*]

**Remarks**

OutBufferSize refers to the total size of the transmit buffer. The default size is 512 bytes. Do not confuse this property with the OutBufferCount property bmc emdash.bmp} OutBufferCount reflects the number of bytes currently waiting in the transmit buffer.

---

**Note** The larger you make the transmit buffer, the less memory you have available to your application. However, if your buffer is too small, you run the risk of overflowing unless you use handshaking. As a general rule, start with a buffer size of 512 bytes. If an overflow error occurs, increase the buffer size to handle your application's transmission rate.

---

**Data Type**

**Integer**

## Output Property, Communications Control

Example

Writes a string of characters to the transmit buffer. This property is not available at design time.

### Syntax

[*form*.]*MSComm*.**Output**[ = *outString$*]

### Data Type

**String**

## Output Example, Communications Control

The following example shows how to send every character the user types to the serial port:

```
Private Sub Form_KeyPress (KeyAscii As Integer)
   Comm1.Output = Chr$(KeyAscii)
End Sub
```

## ParityReplace Property, Communications Control

Sets and returns the character that replaces an invalid character in the data stream when a parity error occurs.

### Syntax

[*form*.]*MSComm*.**ParityReplace**[ = *char$*]

### Remarks

The *parity bit* refers to a bit that is transmitted along with a specified number of data bits to provide a small amount of error checking. When you use a parity bit, the communications control adds up all the bits that are set (having a value of 1) in the data and tests the sum as being odd or even (according to the parity setting used when the port was opened).

By default, the control uses a question mark (?) character for replacing invalid characters. Setting ParityReplace to an empty string ("") disables replacement of the character where the parity error occurs.   The OnComm event is still fired and the CommEvent property is set to comRXParity.

### Data Type

**String**

## PortOpen Property, Communications Control

Example

Sets and returns the state of the communications port (open or closed). This property is not available at design time.

### Syntax

[*form*.]*MSComm*.**PortOpen**[ = {**True** | **False**}]

### Remarks

The following table lists the PortOpen property settings for the communications control.

| Setting | Description |
|---|---|
| **True** | Port is opened. |
| **False** | Port is closed. |

Setting the PortOpen property to **True** opens the port. Setting it to **False** closes the port and clears the receive and transmit buffers. The communications control automatically closes the serial port when your application is terminated.

Make sure that the CommPort property is set to a valid port number before opening the port. If the CommPort property is set to an invalid port number when you try to open the port, the communications control generates error 68 (Device unavailable).

In addition, your serial port device must support the Settings property. If the Settings property contains communications settings that your hardware does not support, your hardware may not work correctly.

If either the DTREnable or the RTSEnable properties is set to **True** before the port is opened, the properties are set to **False** when the port is closed. Otherwise, the DTR and RTS lines remain in their previous state.

### Data Type

**Integer** (Boolean)

Close Copy Print

## PortOpen Example, Communications Control

The following example opens communications port number 1 at 2400 baud with no parity checking, 8 data bits, and 1 stop bit:

```
Comm1.Settings = "2400,n,8,1"
Comm1.CommPort = 1
Comm1.PortOpen = True
```

## RThreshold Property, Communications Control

Sets and returns the number of characters to receive before the communications control sets the CommEvent property to **comEvReceive** and generates the OnComm event.

### Syntax

[*form*.]*MSComm*.**RThreshold**[ = *numChars%*]

### Remarks

Setting the RThreshold property to 0 (the default) disables generating the OnComm event when characters are received.

Setting RThreshold to 1, for example, causes the communications control to generate the OnComm event every time a single character is placed in the receive buffer.

### Data Type

**Integer**

## RTSEnable Property, Communications Control

Determines whether to enable the Request To Send (RTS) line. Typically, the Request To Send signal that requests permission to transmit data is sent from a computer to its attached modem.

### Syntax

[*form*.]*MSComm*.**RTSEnable**[ = {**True | False**}]

### Remarks

The following table lists the RTSEnable property settings for the communications control.

| Setting | Description |
|---|---|
| **True** | Enables the Request To Send line. |
| **False** | (Default) Disables the Request To Send line. |

When RTSEnable is set to **True**, the Request To Send line is set to high (on) when the port is opened, and low (off) when the port is closed.

The Request To Send line is used in RTS/CTS hardware handshaking. The RTSEnable property allows you to manually poll the Request To Send line if you need to determine its state.

For more information on handshaking protocols, see the Handshaking property.

### Data Type

**Integer** (Boolean)

## Settings Property, Communications Control

Example

Sets and returns the baud rate, parity, data bit, and stop bit parameters.

### Syntax

[*form*.]*MSComm*.**Settings**[ = *paramString$*]

### Remarks

If *paramString$* is not valid when the port is opened, the communications control generates error 380 (Invalid property value).

*ParamString$* is composed of four settings and has the following format:

```
"BBBB,P,D,S"
```

Where BBBB is the baud rate, P is the parity, D is the number of data bits, and S is the number of stop bits. The default value of *paramString$* is:

```
"9600,N,8,1"
```

The following table lists the valid baud rates.

| Setting |
|---|
| 110 |
| 300 |
| 600 |
| 1200 |
| 2400 |
| 9600 (Default) |
| 14400 |
| 19200 |
| 38400 (reserved) |
| 56000 (reserved) |
| 128000 (reserved) |
| 256000 (reserved) |

The following table describes the valid parity values.

| Setting | Description | |
|---|---|---|
| E | Even | |
| M | Mark | |
| N | (Default) | None |
| O | Odd | |
| S | Space | |

The following table lists the valid data bit values.

| Setting | |
|---|---|
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | (Default) |

The following table lists the valid stop bit values.

| Setting | |
|---|---|
| 1 | (Default) |

1.5
2

**Data Type**

**String**

Close Copy Print

**Settings Example, Communications Control**

The following example sets the control's port to communicate at 2400 baud with no parity checking, 8 data bits, and 1 stop bit:

```
Comm1.Settings = "2400,N,8,1"
```

## SThreshold Property, Communications Control

Sets and returns the minimum number of characters allowable in the transmit buffer before the communications control sets the CommEvent property to **comEvSend** and generates the OnComm event.

### Syntax

[*form*.]*MSComm*.**SThreshold**[ = *numChars%*]

### Remarks

Setting the SThreshold property to 0 (the default) disables generating the OnComm event for data transmission events. Setting the SThreshold property to 1 causes the communications control to generate the OnComm event when the transmit buffer is completely empty.

If the number of characters in the transmit buffer is less than *numChars%*, the CommEvent property is set to **comEvSend**, and the OnComm event is generated. The **comEvSend** event is only fired once, when the number of characters crosses the SThreshold. For example, if SThreshold equals five, the **comEvSend** event occurs only when the number of characters drops from five to four in the output queue. If there are never more than SThreshold characters in the output queue, the event is never fired.

### Data Type

**Integer**

## OnComm Event, Communications Control

Example

The OnComm event is generated whenever the value of the CommEvent property changes, indicating that either a communications event or an error occurred.

### Syntax

**Private Sub** *MSComm_***OnComm ()**

### Remarks

The CommEvent property contains the numeric code of the actual error or event that generated the OnComm event. Note that setting the RThreshold or SThreshold properties to 0 disables trapping for the **comEvReceive** and **comEvSend** events, respectively.

Close Copy Print

## OnComm Event Example, Communications Control

The following example shows how to handle communications errors and events. You can insert code to handle a particular error or event after its **Case** statement.

```
Private Sub Comm_OnComm ()
   Select Case Comm1.CommEvent
   ' Errors
      Case comBreak            ' A Break was received.
      ' Code to handle a BREAK goes here.
      Case comCDTO             ' CD (RLSD) Timeout.
      Case comCTSTO            ' CTS Timeout.
      Case comDSRTO            ' DSR Timeout.
      Case comFrame            ' Framing Error
      Case comOverrun          ' Data Lost.
      Case comRxOver           ' Receive buffer overflow.
      Case comRxParity         ' Parity Error.
      Case comTxFull           ' Transmit buffer full.
   ' Events
      Case comEvCD             ' Change in the CD line.
      Case comEvCTS            ' Change in the CTS line.
      Case comEvDSR            ' Change in the DSR line.
      Case comEvRing           ' Change in the Ring Indicator.
      Case comEvReceive        ' Received RThreshold # of chars.
      Case comEvSend           ' There are SThreshold number of
                               ' characters in the transmit buffer.
   End Select
End Sub
```

## ComInput Function, Communications Control

Returns and removes a string of characters from the receive buffer.

### Syntax

**ComInput(ByVal** *hWnd* **As Integer,** *lpData* **As Any, ByVal** *cbData* **As Integer) As Integer**

| Parameter | Type | Description |
|---|---|---|
| *hWnd* | **HWND** | Window handle of the control. |
| *lpData* | **LPSTR** | Long pointer to the start of the data buffer. |
| *cbData* | **int** | The length of *lpData* in bytes. |

### Remarks

This function is equivalent to the Input property.

In Visual Basic 1.0, the Input and Output properties are defined as HSZ (null-terminated string) data types. This means that if an application attempts to retrieve a string with an embedded Null character from the receive buffer, the resulting string is truncated at the embedded Null character. The **ComInput** function can retrieve strings from the receive buffer that have embedded Null characters.

### Return Value

Number of bytes received.

## ComOutput Function, Communications Control

Writes a string of characters to the transmit buffer.

**Syntax**

**ComOutput(ByVal** *hWnd* **As Integer,** *lpData* **As Any, ByVal** *cbData* **As Integer) As Integer**

| Parameter | Type | Description |
|---|---|---|
| *hWnd* | **HWND** | Window handle of the control. |
| *lpData* | **LPSTR** | Long pointer to the start of the data buffer. |
| *cbData* | **int** | The length of *lpData* in bytes. |

**Remarks**

This function is equivalent to the Output property.

In Visual Basic 1.0, the Input and Output properties are defined as HSZ (null-terminated string) data types. This means that if an application attempts to send a string with an embedded Null character to the transmit buffer, the resulting string is truncated at the embedded Null character. The **ComOutput** function can send strings to the transmit buffer that have embedded Null characters.

**Return Value**

Number of bytes sent.

## Gauge Control

Properties    Methods    Events    Constants

The gauge control creates user-defined gauges with a choice of linear (filled) or needle styles.

**File Name**

GAUGE16.OCX, GAUGE32.OCX

**Class Name**

Gauge

**Remarks**

This control is useful for thermometers, fuel gauges, percent-complete indicators, or any other type of analog gauge.

---

**Note**  When you use bitmaps or icons in the gauge control and specify those bitmaps in the Picture property at design time, the bitmaps become a part of your form. This means you do not have to distribute them separately. On the other hand, if you use LoadPicture to add bitmaps or icons at run time, then the bitmaps must be present at run time.

---

The Style property defines the type of gauge to be displayed. The default setting is 0 (horizontal linear).

The control's fill area is defined by the InnerTop, InnerBottom, InnerRight, and InnerLeft properties. The default values for these properties create a fill area that covers most of the control. Therefore, when you define a bitmap for the control, only the edges of the bitmap are displayed. To display the bitmap, either set the Style property to 2 or 3 (semicircular or full needle, respectively) or resize the fill area of the control.

When the Style property is either 0 or 1 (indicating a linear gauge), the BackColor and ForeColor properties define the colors of the fill area. The Min, Max, and Value properties determine how the colors are used to fill this area. For example, if Min is 0, Max is 100, and Value is 25, then 25% of the fill area will be drawn with the ForeColor, and 75% will be drawn with the BackColor.

---

**Distribution Note**  When you create and distribute applications that use the gauge control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

---

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| AutoSize | Index | MousePointer | Tag |
| *BackColor | *InnerBottom | Name | Top |
| Container | *InnerLeft | *NeedleWidth | *Value |
| DragIcon | *InnerRight | Object | Visible |
| DragMode | *InnerTop | Parent | WhatsThisHelpID |
| Enabled | Left | *Picture | *Width |
| *ForeColor | *Max | *Style | hWnd |
| *Height | *Min | TabIndex | |
| HelpContextID | MouseIcon | TabStop | |

Value is the default value of the control.

**Note** Name is the equivalent of the CtlName property in Visual Basic 1.0.

### Events

All of the events for this control are listed in the following table. Events that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *Change | DragOver | KeyUp | MouseUp |
| Click | GotFocus | LostFocus | |
| DblClick | KeyDown | MouseDown | |
| DragDrop | KeyPress | MouseMove | |

### Methods

All of the methods for this control are listed in the following table.

| | | |
|---|---|---|
| Drag | Refresh | ZOrder |
| Move | SetFocus | ShowWhatsThis |

## BackColor Property, Gauge Control

Sets or returns the color used to erase the area created by the InnerTop, InnerLeft, InnerBottom, and InnerRight properties.

### Syntax

[*form*.]*Gauge*.**BackColor**[ = *color&*]

### Remarks

BackColor has no effect on gauges with Style = 2 (semicircular needle), or Style = 3 (full needle) when you assign the control's Picture property to a bitmap.

### Data Type

**Long**

## ForeColor Property, Gauge Control

Sets the color used to fill the area defined by the InnerTop, InnerLeft, InnerBottom, and InnerRight properties.

**Syntax**

[*form*.]*Gauge*.**ForeColor**[ = *color&*]

**Remarks**

This property only affects gauges with Style = 0 or 1 (horizontal bar or vertical bar, respectively).

**Data Type**

**Long**

## Height, Width Properties, Gauge Control

Determines the height and width of the gauge control.

**Syntax**

[*form*.]*Gauge*.**Height**[ = *setting%*]
[*form*.]*Gauge*.**Width**[ = *setting%*]

**Remarks**

You cannot resize a gauge control unless the AutoSize property is set to **False**.

**Data Type**

**Integer**

## InnerBottom Property, Gauge Control

Sets or returns the distance from the bottom edge of the gauge control used to display the changeable portion of the gauge.

### Syntax

[*form.*]*Gauge.***InnerBottom**[ = *pixels%*]

### Remarks

This property, expressed in terms of pixels, must be greater than zero. InnerBottom is relative to the bottom edge of the control.

Needle gauges adjust the needle within this area, while fill gauges completely blot out this area to fill the proportionate parts with two colors.

### Data Type

**Integer**

## InnerLeft Property, Gauge Control

Sets or returns the distance from the left edge of the gauge control used to display the changeable portion of the gauge.

**Syntax**

[*form.*]*Gauge.***InnerLeft**[ = *pixels%*]

**Remarks**

This property, expressed in terms of pixels, must be greater than zero. InnerLeft is relative to the Top property of the control.

Needle gauges adjust the needle within this area, while fill gauges completely blot out this area to fill the proportionate parts with two colors.

**Data Type**

**Integer**

## InnerRight Property, Gauge Control

Sets or returns the distance from the right edge of the gauge control used to display the changeable portion of the gauge.

**Syntax**

[*form.*]*Gauge.***InnerRight**[ = *pixels%*]

**Remarks**

This property, expressed in terms of pixels, must be greater than zero. InnerRight is relative to the right edge of the control.

Needle gauges adjust the needle within this area, while fill gauges completely blot out this area to fill the proportionate parts with two colors.

**Data Type**

**Integer**

## InnerTop Property, Gauge Control

Sets or returns the distance from the top edge of the gauge control used to display the changeable portion of the gauge.

### Syntax

[*form.*]*Gauge.***InnerTop**[ = *pixels%*]

### Remarks

This property, expressed in terms of pixels, must be greater than zero. InnerTop is relative to the Top property.

Needle gauges adjust the needle within this area, while fill gauges completely blot out this area to fill the proportionate parts with two colors.

### Data Type

**Integer**

## Max Property, Gauge Control

An integer value (0 ■ 32767) that sets or returns the maximum number that the Value property can take on. The default value is 100.

### Syntax

[*form*.]*Gauge*.**Max**[ = *setting%*]

### Remarks

If you attempt to set the Value property to a value greater than the Max property, it is adjusted to the value of the Max property.

### Data Type

**Integer**

## Min Property, Gauge Control

An integer value (0 ■ 32767) that sets or returns the minimum number that the Value property can take on. The default value is zero.

### Syntax

[*form*.]*Gauge*.**Min**[ = *setting%*]

### Remarks

If you attempt to set the Value property to a value less than the Min property, it is adjusted to the value of the Min property.

### Data Type

**Integer**

## NeedleWidth Property, Gauge Control

Sets or returns the width, in pixels, of the needle on needle-style gauges. The range is 0 to 32767.

**Syntax**

[*form.*]*Gauge.***NeedleWidth**[ = *width%*]

**Data Type**

**Integer**

## Picture Property, Gauge Control

Specifies a bitmap to display on the gauge.

### Syntax

[*form*.]*Gauge*.**Picture**[ = *picture*]

### Remarks

The following table lists the Picture property settings for the gauge control.

| Setting | Description |
|---|---|
| (none) | (Default) No bitmap specified for the gauge. |
| (bitmap) | Designates a graphic to display on the gauge. You can load the graphic from the Properties window at design time. |

Several bitmaps for the gauge control are located in the \BITMAPS\GAUGE subdirectory. The style you choose for a gauge must be compatible with the bitmap or the graphic will not be drawn properly.

**Note** This control can display bitmap (.BMP) and icon (.ICO) files.

You can load a graphic at design time from the Properties window. When you set the Picture property at design time, the graphic is saved and loaded with the form. If you create an executable file, the .EXE file contains the image.

You can set this property at run time by using the **LoadPicture** function on a bitmap or icon, or you can use Clipboard methods such as **GetData, SetData,** and **GetFormat** with the nontext Clipboard formats **vbCFBitmap** and **vbCFDIB**, as defined in the object library in the Object Browser. When you load a graphic at run time, the graphic is not saved with the application. Use the **SavePicture** statement to save a graphic from a form or picture box into a file.

**Note** At run time, either you can set the Picture property to any other object's Picture or Image property, or you can assign it the graphic returned by the LoadPicture function. You can only assign the Picture property directly.

### Data Type

**Integer**

## Style Property, Gauge Control

Sets or returns the type of gauge.

**Syntax**

[*form*.]*Gauge*.**Style**[ = *setting%*]

**Remarks**

The following table lists the Style property settings for the gauge control.

| Setting | Description |
|---|---|
| 0 | (Default) Horizontal linear gauge with fill. |
| 1 | Vertical gauge with fill. |
| 2 | Semicircular needle gauge. |
| 3 | Full circle needle gauge. |

The semicircular needle gauge places the needle base in the bottom center of the area defined by the Inner*portion* properties. The needle length is calculated so that the needle is never drawn outside of this area. When Value = Min, the needle will point 90 degrees to the left. When Value = Max, the needle will point 90 degrees to the right. When Value = (Min + Max)/2, the needle points straight up.

The full-circle needle gauge places the needle base in the center of the area defined by the Inner*portion* properties. The needle length is calculated so that the needle will never be drawn outside of this area. When Value = Min or Value = Max, the needle points 90 degrees to the left. Setting the Value property between Min and Max will point the needle to a proportionate point on the circle, moving clockwise.

**Data Type**

**Integer** (Enumerated)

## Value Property, Gauge Control

Sets or returns the current position of the gauge. See the Style property for more details.

**Syntax**

[*form*.]*Gauge*.**Value**[ = *setting%*]

**Remarks**

If you attempt to set the Value property to a value less than the Min property, it is adjusted to the value of the Min property. If you attempt to set the Value property to a value greater than the Max property, it is adjusted to the value of the Max property.

**Data Type**

**Integer**

## Change Event, Gauge Control

Occurs when the control's Value property changes.

**Syntax**

**Private Sub** *Gauge_***Change**( )

## Graph Control

See Also Properties Methods Events Constants Error Messages

The graph control allows you to design graphs interactively on your forms. At run time, you can send new data to the graphs and draw them, print them, copy them onto the Clipboard, or change their styles and shapes. The following is a typical graph control:

**File Name**

GRAPH16.OCX, GRAPH32.OCX

**Class Name**

Graph

**Remarks**

The graph control acts as a link between your application and the Graphics Server graphing and charting library.

At design time, the graph control has an automatic redraw capability. Every time you change a property, the control redraws the graph so that you can see the effects of the change. You can enter data for the graph either at design time or at run time. At run time, when graph is given new data and style options, it combines these new values with your design-time values.

As a design aid, the graph control automatically generates random data at design time to give you an idea of what your graph will look like.

---

**Distribution Note** When you create and distribute applications that use the graph control, you should install the appropriate files in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

---

**See Also**

Property Types and Arrays
Graph Control Extended Version
Graph Types and Negative Values

## Property Types and Arrays, Graph Control

Example

The following table describes array properties for the graph control.

| Property | Description |
|---|---|
| GraphData | Values to be graphed (this is a two-dimensional array when there are multiple data sets). |
| ColorData | Colors of bars, pie slices, lines, and so on. |
| ExtraData | Extra style options (for example, which pie slices to explode). |
| LabelText | Labels. |
| LegendText | Legends. |
| PatternData | Pattern and line styles. |
| SymbolData | Symbols for lines, legends, and so on. |
| XPosData | X-variable data for scatter graphs. |

Array properties are controlled through two simple properties: ThisSet and ThisPoint. ThisSet is the index for the data you entered with the GraphData property. ThisPoint references the individual data points for the set specified by the ThisSet property. Both have a minimum value of 1.

For example, if you set ThisSet to 1, ThisPoint to 5, and LabelText to "Friday," the fifth label of the first data set is set to the text string "Friday."

The AutoInc property, when set to 1 (on), automatically increments ThisPoint and ThisSet every time you enter an array property value.

At run time, when you dynamically create a new instance of a control array, you must reassign all data associated with array properties.

The overall dimensions of the arrays are determined by the properties NumSets and NumPoints. ThisSet and ThisPoint cannot exceed NumSets and NumPoints, respectively, and the AutoInc property functions monitor their current values. NumSets and NumPoints also determine what the graphs look like. For example, if you want to graph three data sets, each containing ten points, set NumSets to 3 and NumPoints to 10, and then enter the GraphData values.

DataReset is another property associated with arrays. It allows you to clear all the data held in any or all of the array properties. For example, if you haven't set any LabelText strings, the graph control labels your graph 1, 2, 3, and so on. Deleting all your labels individually would have the effect of displaying no labels (that is, labels exist but they are all null). Using DataReset sets the LabelText strings back to their original numeric values of 1, 2, 3, and so on.

■

## Property Types and Arrays Example, Graph Control

At design time, to enter a data set of five points, set the AutoInc property to 1 (on), select the GraphData property in the Properties window, and enter the following five values, pressing ENTER between each number. For example:

**10** ENTER
**9** ENTER
**8** ENTER
**7** ENTER
**6** ENTER

Other information about graphs, such as labels and legends, can be entered in the same manner.

To change the values of a graph at run time, you write code. The following two examples would cause the same property value changes as in the previous example:

```
 ' Example 1
 Graph1.AutoInc = 1
 Graph1.GraphData = 10
 Graph1.GraphData = 9
 Graph1.GraphData = 8
 Graph1.GraphData = 7
 Graph1.GraphData = 6
 Graph1.DrawMode = 2

 ' Example 2
 Graph1.AutoInc = 1
 For I% = 1 To 5
    Graph1.GraphData = 11 ■ i%
Next I%
Graph1.DrawMode = 2
```

## Graph Types and Negative Values, Graph Control

Certain graph types cannot handle negative data meaningfully. They are the following:

- Pie charts (2D & 3D).
- Stacked Bar graphs.
- Gantt charts.
- Area graphs.
- Polar graphs.

For these graphs, negative data is forced to a positive number, however the data is not permanently changed. Changing to a graph type for which negative values are meaningful restores the original data.

**Properties**

The following table lists the properties for this control. Properties that apply *only* to this control, or that require special consideration, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *AutoInc | *Foreground | *LegendStyle | TabStop |
| *Background | *GraphCaption | *LegendText | Tag |
| BorderStyle | *GraphData | *LineStats | *ThickLines |
| *BottomTitle | *GraphStyle | Name | *ThisPoint |
| *ColorData | *GraphTitle | *NumPoints | *ThisSet |
| Container | *GraphType | *NumSets | *TickEvery |
| *CtlVersion | *GridStyle | Object | *Ticks |
| *DataReset | Height | *Palette | Top |
| DragIcon | HelpContextID | Parent | Visible |
| DragMode | hWnd | *PatternData | WhatsThisHelpID |
| *DrawMode | *ImageFile | *PatternedLines | Width |
| *DrawStyle | Index | *Picture | *XPosData |
| Enabled | *IndexStyle | *PrintStyle | *YAxisMax |
| *ExtraData | *LabelEvery | *QuickData | *YAxisMin |
| *FontFamily | *Labels | *RandomData | *YAxisPos |
| *FontSize | *LabelText | *SeeThru | *YAxisStyle |
| *FontStyle | Left | *SymbolData | *YAxisTicks |
| *FontUse | *LeftTitle | TabIndex | |

QuickData is the default value of the control.

**Note** Name is equivalent to the CtlName property in Visual Basic 1.0.

**Events**

All of the events for this control are listed in the following table.

| | | | |
|---|---|---|---|
| Click | DragOver | KeyPress | MouseDown |
| DblClick | GotFocus | KeyUp | MouseMove |
| DragDrop | KeyDown | LostFocus | MouseUp |

**Methods**

All of the methods for this control are listed in the following table.

| | | | |
|---|---|---|---|
| **Drag** | **Refresh** | **SetFocus** | **ShowWhatsThis** |
| | | | **ZOrder** |

## AutoInc Property, Graph Control

Example

Allows the properties specific to arrays to be set without manually incrementing the ThisPoint counter from ThisPoint = 1 to ThisPoint = NumPoints.

When NumSets > 1, AutoInc goes through all the points and sets them consecutively from ThisPoint = 1 to ThisPoint = NumPoints and from ThisSet = 1 to ThisSet = NumSets.

**Syntax**

[*form*.]*Graph*.**AutoInc**[ = *setting%*]

**Remarks**

The following table lists the AutoInc property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | Off |
| 1 | (Default) On |

When AutoInc is set to a new value (0 or 1), ThisPoint and ThisSet are both reinitialized to 1.

If you set the AutoInc property to 1 (on), when you switch from setting one of the array properties to setting a different one, both ThisPoint and ThisSet are reinitialized to 1.

AutoInc only changes ThisPoint and ThisSet when you set data values. When you get or use data values, ThisPoint and ThisSet are unaffected.

The AutoInc property works for all the properties specific to arrays:

- ColorData
- ExtraData
- GraphData
- LabelText
- LegendText
- PatternData
- SymbolData
- XPosData

**Data Type**

**Integer**

■

## AutoInc Example, Graph Control

```
Graph1.ThisSet = 1
For I% = 1 to Graph1.NumSets
   Graph1.ThisPoint = 1
   For J% = 1 to Graph1.NumPoints
      Graph1.GraphData = J%*I%
      If Graph1.ThisPoint < Graph1.NumPoints Then
         Graph1.ThisPoint = Graph1.ThisPoint + 1
      End If
   Next J%
   If Graph1.ThisSet < Graph1.NumSets Then
      Graph1.ThisSet = Graph1.ThisSet + 1
   End If
Next I%
Graph1.DrawMode = 2
```

Using the AutoInc property, the preceding code may be rewritten as:

```
Graph1.AutoInc = 1
For I% = 1 To (Graph1.NumSets * Graph1.NumPoints)
   Graph1.GraphData = Graph1.ThisPoint * Graph1.ThisSet
Next I%
Graph1.DrawMode = 2
```

It is not possible to use ThisPoint or ThisSet as counters in **For** statements. Visual Basic does not allow it.

## Background Property, Graph Control

Selects the background color of the graph.

**Syntax**

[*form*.]*Graph*.**Background**[ = *color%*]

**Remarks**

The following table lists the Background property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | Black |
| 1 | Blue |
| 2 | Green |
| 3 | Cyan |
| 4 | Red |
| 5 | Magenta |
| 6 | Brown |
| 7 | Light gray |
| 8 | Dark gray |
| 9 | Light blue |
| 10 | Light green |
| 11 | Light cyan |
| 12 | Light red |
| 13 | Light magenta |
| 14 | Yellow |
| 15 | (Default) White |

When you change the background color, the colors for the components of the graph are automatically selected. However, you may change the Foreground and the ColorData properties.

**Data Type**

**Integer** (Enumerated)

## BottomTitle Property, Graph Control

Example

Places the text string that you provide at the bottom of the graph, parallel to the horizontal axis.

### Syntax

[*form*.]*Graph*.**BottomTitle**[ = *string$*]

### Remarks

This property is ignored for Pie charts.

### Data Type

**String**

■

## BottomTitle Example, Graph Control

The following code places the title, "Title," at the bottom of a graph (Graph1) when you click a command button and no title currently exists. If the BottomTitle property does have a value, when you click the command button, the title will become blank. To try this example, paste the code into the Declarations section of a form that contains a command button and a graph.

```
Private Sub Command1_Click ()
   Graph1.RandomData = 1
   If Graph1.BottomTitle = "" Then
      Graph1.BottomTitle = "Title"
   Else
      Graph1.BottomTitle = ""
   End If
   Graph1.DrawMode = 2
End Sub
```

## ColorData Property, Graph Control

Selects the colors for each of the data sets on the graph. For pie charts and for bar graphs with NumSets = 1, you should specify a color for each point rather than for each set.

**Syntax**

[*form*.]*Graph*.**ColorData**[ **=** *setting%*]

**Remarks**

The following table lists the ColorData property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Black |
| 1 | Blue |
| 2 | Green |
| 3 | Cyan |
| 4 | Red |
| 5 | Magenta |
| 6 | Brown |
| 7 | Light gray |
| 8 | Dark gray |
| 9 | Light blue |
| 10 | Light green |
| 11 | Light cyan |
| 12 | Light red |
| 13 | Light magenta |
| 14 | Yellow |
| 15 | White |

Once you select one color, colors should be selected for all sets or they are shown in black.

Since this is an array property, the array element is determined by the current value of the ThisPoint property.

When you enter data, you can use the AutoInc property. If you set the AutoInc property to 1 (on), every time you set a new value, the ThisPoint counter is automatically incremented.

**Data Type**

**Integer** (Enumerated)

## CtlVersion Property, Graph Control

Gives the current release of your graph control. This property is read-only.

**Syntax**

[*form*.]*Graph*.**CtlVersion**

**Data Type**

**String**

## DataReset Property, Graph Control

Allows you to remove any or all of the array information that has been supplied to the graph control.

### Syntax

[*form*.]*Graph*.**DataReset**[ = *setting%*]

### Remarks

The following table lists the DataReset property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) None |
| 1 | GraphData |
| 2 | ColorData |
| 3 | ExtraData |
| 4 | LabelText |
| 5 | LegendText |
| 6 | PatternData |
| 7 | SymbolData |
| 8 | XPosData |
| 9 | All Data |

The All Data option resets all the data and text arrays.

When you reset an array, you reset it to the original empty state. All properties are set to their default values.

### Data Type

**Integer** (Enumerated)

## DrawMode Property, Graph Control

Defines the drawing mode for the graph control.

### Syntax

[*form*.]*Graph*.**DrawMode**[ = *mode%*]

### Remarks

The following table lists the DrawMode property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | No Action |
| 1 | Clear |
| 2 | Draw |
| 3 | Blit |
| 4 | Copy |
| 5 | Print |
| 6 | Write |

DrawMode property values 0 through 3 are recorded when a graph is saved to disk. These values remain the same between design mode and run mode. DrawMode property values 4, 5, and 6 are transient values that trigger the specified actions.

At design time, when you change a property value, the graph is automatically redrawn to show the effect of the change. At run time, the graph is only redrawn when you set DrawMode to 2 (Draw) or 3 (Blit). This allows you to change as many property values as you want before displaying the graph. However, when the form containing a graph is first displayed, the graph is automatically displayed according to the current DrawMode value.

| Setting | Action |
|---|---|
| 0 | The control is left blank; the graph will not appear. When you want the graph to appear, reset DrawMode to 2. |
| 1 | No graph is drawn, but the background of the control is set to the color specified by the Background property. If there is graph caption text, it is displayed in the center of the control. |
| 2 | (Default)   At design time, this redraws your graph every time you change a property. At run time, resetting DrawMode to 2 causes the graph to be redrawn. |
| 3 | There is a brief pause, and then the graph appears all at once. In this mode, the Graphics Server builds a hidden bitmap of the graph and then displays it using the Windows API **BitBlit** function. This mode is useful if you want to draw a graph, update it with new data, and then instantaneously display the updated graph. |
| 4 | The image of the graph is copied onto the Clipboard in either bitmap or metafile format. If DrawMode is set to 3 (Blit), it is in bitmap format; otherwise, it is in metafile format. |
| 5 | A high-quality image of the graph can be printed without the form. For more information, see the PrintStyle property. |
| 6 | The image of the graph is written to disk as a bitmap (.BMP) or metafile (.WMF). For this option to work, the ImageFile property must be set to provide a name for the file. If DrawMode is set to 3 (Blit), a bitmap is created; otherwise, a metafile is created. |

### Data Type

**Integer** (Enumerated)

## DrawStyle Property, Graph Control

If the setting is monochrome, this property sets the background to white and all colors to black. If no PatternData, SymbolData, or GraphStyle properties have been set, DrawStyle supplies default patterns and symbols.

### Syntax

[*form*.]*Graph*.**DrawStyle**[ = *style%*]

### Remarks

The following table lists the DrawStyle property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | Monochrome |
| 1 | (Default) Color |

### Data Type

**Integer** (Enumerated)

## ExtraData Property, Graph Control

Example

The ExtraData property has two purposes:

- To explode pie chart segment(s).
- To specify the color of the sides of a three-dimensional bar chart.

### Syntax

[*form*.]*Graph*.**ExtraData**[ = *setting%*]

### Remarks

The ExtraData property settings for pie charts are listed in the following table.

| Setting | Description |
|---|---|
| 0 | (Default) Not exploded |
| 1 | Exploded |

**Note** With pie charts, when the AutoInc property is set to 1, setting the ExtraData property cycles automatically through the set of pie slices, exploding each slice in turn. To explode a single slice, set AutoInc to 0, set the ThisPoint property to the datapoint you wish to explode, and finally set the ExtraData property to 1.

For three-dimensional bar charts, the ExtraData property settings are described in the following table.

| Setting | Description |
|---|---|
| 0 | (Default) Black |
| 1 | Blue |
| 2 | Green |
| 3 | Cyan |
| 4 | Red |
| 5 | Magenta |
| 6 | Brown |
| 7 | Light gray |
| 8 | Dark gray |
| 9 | Light blue |
| 10 | Light green |
| 11 | Light cyan |
| 12 | Light red |
| 13 | Light magenta |
| 14 | Yellow |
| 15 | White |

Since this is an array property, the array element you set is determined by the current value of the ThisPoint property.

When you enter data, you can use the AutoInc property. If you set the AutoInc property to 1 (on), every time you set a new value, the ThisPoint counter is automatically incremented.

### Data Type

**Integer** (Enumerated)

■

## ExtraData Example, Graph Control

The following code explodes the segments from the center of a three-dimensional pie chart. To try this example, paste the code into the Form_Load event procedure of a form that contains a graph (Graph1).

```
Private Sub Form_Load ()
   For I% = 1 to 4
      Graph1.GraphData = I%
   Next I%
   ThisPoint = 2
   Graph1.ExtraData = 1
   ThisPoint = 4
   Graph1.ExtraData = 1
   Graph1.DrawMode = 2
   Graph1.GraphType = 2
End Sub
```

## FontFamily Property, Graph Control

Selects the font family in which the text specified by the FontUse property is displayed.

### Syntax

[*form*.]*Graph*.**FontFamily**[ = *setting%*]

### Remarks

The following table lists the FontFamily property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Roman |
| 1 | Swiss |
| 2 | Modern |

The graph control specifies font families rather than type faces to avoid having to list all the available fonts, which may vary from one computer to another. A font of the requested generic type (Roman, Swiss, or Modern) is always available, regardless of the Windows configuration used on your computer.

### Data Type

**Integer** (Enumerated)

## FontSize Property, Graph Control

Determines the approximate font size in which the text specified by the FontUse property is displayed.

**Syntax**

[*form*.]*Graph*.**FontSize**[ = *setting%*]

**Remarks**

Enter a value between 50 and 500, inclusive. This value is the percentage of the system font size. The default depends on the setting of the FontUse property.

| FontUse setting | FontSize default |
|---|---|
| 0 (graph title) | 200% |
| 1 (other titles) | 150% |
| 2 (labels) | 100% |
| 3 (legend) | 100% |

FontSize acts as a starting point rather than an absolute setting; the text is reduced, if necessary, to fit into the available space.

**Data Type**

**Integer**

## FontStyle Property, Graph Control

Determines the style in which the text specified by the FontUse property is displayed.

**Syntax**

[*form*.]*Graph*.**FontStyle**[ = *setting%*]

**Remarks**

The following table lists the FontStyle property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) |
| 1 | Italic |
| 2 | Bold |
| 3 | Bold italic |
| 4 | Underlined |
| 5 | Underlined italic |
| 6 | Underlined bold |
| 7 | Underlined bold italic |

**Data Type**

**Integer** (Enumerated)

## FontUse Property, Graph Control

Determines to which text on a graph you will apply the settings for the FontFamily, FontSize, and FontStyle properties.

### Syntax

[*form*.]*Graph*.**FontUse**[ = *setting%*]

### Remarks

The following table lists the FontUse property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Graph title |
| 1 | Other titles |
| 2 | Labels |
| 3 | Legend |
| 4 | All text |

After you select a text type using FontUse, select the font family, size, and style for that type by setting the FontFamily, FontSize, and FontStyle properties. You can use setting 4 (all text) to make all of your text look alike. For example, you can set all text to display as Swiss family, size 200%, and bold. You can then reuse the FontUse property to change one or more specific text types; for example, you might make all legends bold and underlined.

**Note** At design time, the values displayed in the Properties window for the font family, size, and style are shown for the graph title only.

### Data Type

**Integer** (Enumerated)

## Foreground Property, Graph Control

Sets the color of titles, labels, legends, and axes.

**Syntax**

[*form*.]*Graph*.**Foreground**[ = *setting%*]

**Remarks**

The following table lists the Foreground property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | Black |
| 1 | Blue |
| 2 | Green |
| 3 | Cyan |
| 4 | Red |
| 5 | Magenta |
| 6 | Brown |
| 7 | Light gray |
| 8 | Dark gray |
| 9 | Light blue |
| 10 | Light green |
| 11 | Light cyan |
| 12 | Light red |
| 13 | Light magenta |
| 14 | Yellow |
| 15 | White |
| 16 | (Default) Auto black/white |

The graph control automatically uses black or white as its foreground default color. Depending on the background color set, it picks the color that gives the best contrast.

The ColorData property determines the colors of bars, pie slices, and so on.

**Data Type**

**Integer** (Enumerated)

## GraphCaption Property, Graph Control

Example

Accepts a single line of text that is displayed when DrawMode = 1 (Clear).

### Syntax

[*form*.]*Graph*.**GraphCaption**[ = *caption$*]

### Remarks

The colors of the text and the background can be selected using the Foreground and Background properties.

### Data Type

**String**

■

## GraphCaption Example, Graph Control

The following code displays the text, "Graphics Server," as the caption for Graph1.

```
Graph1.GraphCaption = "Graphics Server"
Graph1.DrawMode = 1
```

## GraphData Property, Graph Control

Example

Sets the data to be graphed.

### Syntax

[*form*.]*Graph*.**GraphData**[ = *data!*]

### Remarks

Since this is a two-dimensional array property, the array element you set is determined by the current value of the ThisPoint and ThisSet properties.

When you enter data, you can use the AutoInc property. If you set the AutoInc property to 1 (on), every time you set a new value, the ThisPoint counter is automatically incremented. When it reaches its maximum value (NumPoints), the ThisSet counter is incremented, and ThisPoint is reset to 1. If ThisSet reaches its maximum value (NumSets), it is also reset to 1.

### Data Type

**Single**

■

## GraphData Example, Graph Control

The following code draws the data sets for a bar graph. The data sets are specified by the NumSets property, and the number of points per data set is specified by the NumPoints property. To try this example, paste this code into the Form_Load event procedure of a form that contains a graph (Graph1).

```
Private Sub Form_Load ()
   Graph1.ThisSet = 1
   For I% = 1 to Graph1.NumSets
      Graph1.ThisPoint = 1
      For J% = 1 to Graph1.NumPoints
         Graph1.GraphData = J%*I%
      If Graph1.ThisPoint < Graph1.NumPoints Then
         Graph1.ThisPoint = Graph1.ThisPoint + 1
      End If
      Next J%
      If Graph1.ThisSet < Graph1.NumSets Then
         Graph1.ThisSet = Graph1.ThisSet + 1
      End If
   Next I%
   Graph1.DrawMode = 2
   Graph1.DrawMode = 4
End Sub
```

Using the AutoInc property, the preceding code may be rewritten as:

```
Graph1.AutoInc = 1
For I% = 1 To (Graph1.NumSets * Graph1.NumPoints)
   Graph1.GraphData = Graph1.ThisPoint * Graph1.ThisSet
Next I%
Graph1.DrawMode = 2
Graph1.DrawMode = 4
```

## GraphStyle Property, Graph Control

Sets the characteristics of each type of graph.

**Syntax**

[*form*.]*Graph*.**GraphStyle**[ = *type%*]

**Remarks**

The following table describes the GraphStyle property settings for each type of graph.

| Graph type | GraphStyle setting | | Notes |
|---|---|---|---|
| 2D and 3D pie | 0 | (Default) Lines join labels to pie | If LabelText values are set, then those labels are used; otherwise, the numerical value is used as a label. |
| | 1 | No label lines | |
| | 2 | Colored labels | |
| | 3 | Colored labels without lines | |
| | 4 | % Labels | |
| | 5 | % Labels without lines | |
| | 6 | % Colored labels | |
| | 7 | % Colored labels without lines | |
| 2D bar | 0 | (Default) Vertical bars, clustered if NumSets > 1 | If NumSets = 1, then each bar has a different color. If NumSets > 1, then each data set is a different color. |
| | 1 | Horizontal | |
| | 2 | Stacked | |
| | 3 | Horizontal stacked | |
| | 4 | Stacked % | |
| | 5 | Horizontal stacked% | |
| 3D bar | As preceding, plus: | | |
| | 6 | Z-clustered | Z-clustered means that the data points for successive sets are drawn in front of the previous one. This gives an illusion of depth. |
| | 7 | Horizontal Z-clustered | |
| Gantt | 0 | (Default) Adjacent bars | |
| | 1 | Spaced bars | Spaced bars have a gap of one bar's width between successive bars. |
| Line, Log/Lin, and polar | 0 | (Default) Lines | You can create thick or patterned lines by setting the ThickLine or PatternLine property to 1 (on). |
| | 1 | Symbols | |
| | 2 | Sticks | |
| | 3 | Sticks and symbols | |
| | 4 | Lines | |
| | 5 | Lines and symbols | |
| | 6 | Lines and sticks | |
| | 7 | Lines and sticks | |

| | | | |
|---|---|---|---|
| | | and symbols | |
| Area | 0 | (Default) Stack the data sets | |
| | 1 | Absolute | Absolute uses absolute values from Y = 0 (so values can be hidden). |
| | 2 | Percentage | Percentage shows the sets as a percentage of the total. |
| Scatter | 0 | (Default) Symbols only | Scatter graphs require XPosData to be present. |
| HLC | 0 | (Default) High, low, and close bars | ThickLines may be used. |
| | 1 | No close bar | |
| | 2 | No high-low bars | |
| | 3 | No bars | |

**Data Type**

**Integer** (Enumerated)

## GraphTitle Property, Graph Control

Example

Places a text string above the graph.

### Syntax

[*form*.]*Graph*.**GraphTitle**[ = *title$*]

### Remarks

A graph title cannot contain more than 80 characters.

A graph title may not be displayed if it is too long to fit on a graph. When this occurs, increase the width of the graph to display the graph title.

### Data Type

**String**

■

## GraphTitle Example, Graph Control

The following code places the title, "Title," at the top of a graph (Graph1) when you click a command button and no title currently exists. If the GraphTitle property does have a value, when you click the command button, the title will become blank. To try this example, paste the code into the Declarations section of a form that contains a command button and a graph.

```
Private Sub Command1_Click ()
   Graph1.RandomData = 1
   If Graph1.GraphTitle = "" Then
      Graph1.GraphTitle = "Title"
   Else
      Graph1.GraphTitle = ""
   End If
   Graph1.DrawMode = 2
End Sub
```

## GraphType Property, Graph Control

Specifies the type of graph. For illustrations of the different types of graphs, see the *Custom Control Reference*.

### Syntax

[*form*.]*Graph*.**GraphType**[ = *setting%*]

### Remarks

The following table lists the GraphType property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | None |
| 1 | 2D pie |
| 2 | 3D pie |
| 3 | (Default) 2D bar |
| 4 | 3D bar |
| 5 | Gantt |
| 6 | Line |
| 7 | Log/Lin |
| 8 | Area |
| 9 | Scatter |
| 10 | Polar |
| 11 | HLC |

For each graph type there are many style options. For more information, see the GraphStyle property.

### Data Type

**Integer** (Enumerated)

## GridStyle Property, Graph Control

Places reference grids on the graph axes. For illustrations showing each style of grid, see the *Custom Control Reference*.

### Syntax

[*form*.]*Graph*.**GridStyle**[ = *setting%*]

The following table lists the GridStyle property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) None |
| 1 | Horizontal |
| 2 | Vertical |
| 3 | Both |

For polar graphs, the horizontal axes are concentric circles, and the vertical axes are radial lines (spokes).

### Data Type

**Integer** (Enumerated)

## ImageFile Property, Graph Control

Sets a file name to which the bitmap or metafile is written when DrawMode is set to 6. If a path is not specified, the current directory is used.

### Syntax

[*form*.]*Graph*.**ImageFile**[ = *filename$*]

### Remarks

The appropriate extension (.BMP or .WMF) is appended automatically. If you set DrawMode to 3 (Blit), a bitmap is created; otherwise, a metafile is created.

---

**Note** You cannot use this property to create a 256-color bitmap.

---

### Data Type

**String**

## IndexStyle Property, Graph Control

Example1 Example2

Sets the data array index style.

### Syntax

[*form*.]*Graph*.**IndexStyle**[ = *setting%*]

### Remarks

The following table lists the IndexStyle property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Standard. One-dimensional arrays are accessed through the ThisPoint property. |
| 1 | Enhanced. One-dimensional arrays are accessed through the IndexStyle property. |

When IndexStyle = 1, the graph control's arrays are accessed as described in the following table.

| Array | Properties used |
|---|---|
| GraphData | ThisSet and ThisPoint (two-dimensional array). |
| ColorData | ThisSet or ThisPoint. |
| ExtraData | ThisSet or ThisPoint. |
| LabelText | ThisPoint. |
| LegendText | ThisSet or ThisPoint. |
| PatternData | ThisSet or ThisPoint. |
| SymbolData | ThisSet. |
| XPosData | ThisSet and ThisPoint (two-dimensional array). |

If the current graph type is a pie chart or a single-data-set bar graph, ThisPoint is used. For any other graph types, ThisSet is used. Pie charts and single-data-set bar graphs use ThisPoint because they display legends per point rather than per data set.

**Note** If the AutoInc property is on, the IndexStyle setting does not matter because AutoInc increments ThisSet and ThisPoint correctly irrespective of the IndexStyle setting. Also, once data arrays have been created, graphs are drawn in the normal way, regardless of the IndexStyle property.

### Data Type

**Integer** (Enumerated)

■

## IndexStyle Example 1, Graph Control

```
Graph1.GraphType = 6               ' Line graph.
Graph1.IndexStyle = 1              ' Enhanced index style.

For i% = 1 To Graph1.NumSets
   Graph1.ThisSet = i%
   For j% = 1 To Graph1.NumPoints
      Graph1.ThisPoint = j%
      Graph1.GraphData = your data value
      Graph1.XPosData = your data value
   Next
Next

For i% = 1 to Graph1.NumSets
   Graph1.ThisSet = i%             ' Use ThisSet as index.
   Graph1.LegendText = "Data set" + Str$(i%)
   Graph1.ExtraData = your data value
   Graph1.ColorData = your data value
   Graph1.PatternData = your data value
   Graph1.SymbolData = your data value
Next

For i% = 1 To Graph1.NumPoints
   Graph1.ThisPoint = i%
   Graph1.LabelText = "Data point" = Str$(i%)
Next

Graph1.DrawMode = 2
```

▪

## IndexStyle Example 2, Graph Control

```
Graph1.GraphType = 6               ' Line graph.
Graph1.IndexStyle = 0              ' Standard index style.

For i% = 1 to Graph1.NumSets
   Graph1.ThisSet = i%
   For j% = 1 To Graph1.NumPoints
      Graph1.ThisPoint = j%
      Graph1.GraphData = your data value
      Graph1.XPosData = your data value
   Next
Next

For i% = 1 to Graph1.NumSets
   Graph1.ThisPoint = i%           ' Use ThisPoint as index.
   Graph1.LegendText = "Legend" + Str$(i%)
   Graph1.ExtraData = your data value
   Graph1.ColorData = your data value
   Graph1.PatternData = your data value
   Graph1.SymbolData = your data value
Next

For i% = 1 To Graph1.NumPoints
   Graph1.ThisPoint = i%
   Graph1.LabelText = "Label" = Str$(i%)
Next

Graph1.DrawMode = 2
```

## LabelEvery Property, Graph Control

Determines the frequency of labels displayed on the X axis.

### Syntax

[*form*.]*Graph*.**LabelEvery**[ = *frequency%*]

### Remarks

Enter a value between 1 (the default) and 1000, inclusive.

For example, suppose you have a graph with five points and the LabelText property is set to "Jan," "Feb," "Mar," "Apr," and "May." If the LabelEvery property is set to 1, all five labels are displayed. If it is set to 2, "Jan," "Mar," and "May" (the first, third, and fifth labels) are displayed. Finally, if LabelEvery is set to 3, only "Jan" and "Apr" (the first and fourth labels) are displayed.

**Note** The LabelEvery property only affects the graph control when the XPosData property is not set. Therefore, LabelEvery never affects scatter diagrams, which always use XPosData.

### Data Type

**Integer**

## Labels Property, Graph Control

Determines if labels are displayed along the graph's X and Y axes. For pie charts, this property determines if labels are displayed.

### Syntax

[*form*.]*Graph*.**Labels**[ = *setting%*]

### Remarks

The following table lists the Labels property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Off |
| 1 | On |
| 2 | X labels displayed |
| 3 | Y labels displayed |

You can display the labels for the X and Y axes separately. This property operates independently of the Ticks property.

### Data Type

**Integer** (Enumerated)

## LabelText Property, Graph Control

Allows label text to be entered. For illustrations of this property, see the *Custom Control Reference*.

### Syntax

[*form*.]*Graph*.**LabelText**[ = *label$*]

### Remarks

If no text has been entered, the labels show the value of the ThisPoint property for all graphs except pie charts, which show the magnitude of the slices.

Since this is an array property, the array element you set is determined by the current value of the ThisPoint property.

When entering text, you may use the AutoInc property. If you set the AutoInc property to 1 (on), every time you set a new string, the ThisPoint counter is automatically incremented.

The LabelText property cannot contain more than 80 characters.

Label text may not be displayed if it is too long to fit on a graph.

### Data Type

**String**

## LeftTitle Property, Graph Control

Example

Places the text string that you provide to the left of the vertical axis.

### Syntax

[*form*.]*Graph*.**LeftTitle**[ = *title$*]

### Remarks

This property is ignored for pie charts.

A left title cannot contain more than 80 characters.

A left title may not be displayed if it is too long to fit on a graph. When this occurs, increase the width of the graph to display the left title.

### Data Type

**String**

■

## LeftTitle Example, Graph Control

The following code places the title, "Title," to the left of the vertical axis of a graph (Graph1) when you click a command button and LeftTitle currently has no value. If the LeftTitle property does contain a text string, when you click the command button, the title will become blank. To try this example, paste the code into the Declarations section of a form that contains a command button and a graph.

```
Private Sub Command1_Click ()
   If Graph1.LeftTitle = "" Then
      Graph1.LeftTitle = "Title"
   Else
      Graph1.LeftTitle = ""
   End If
   Graph1.DrawMode = 2
End Sub
```

## LegendStyle Property, Graph Control

Gives the option of coloring the text you enter as legends (LegendText property). This color is in addition to the colored symbols or patterns.

### Syntax

[*form*.]*Graph*.**LegendStyle**[ = *setting%*]

### Remarks

The following table lists the LegendStyle property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | Monochrome |
| 1 | Color |

### Data Type

**Integer** (Enumerated)

## LegendText Property, Graph Control

Allows you to enter text for legends.

### Syntax

[*form*.]*Graph.***LegendText**[ = *text$*]

### Remarks

There should be one text string for each data set. Pie charts and bar graphs with only one data set should have a string for each data point.

Since this is an array property, the array element is determined by the current value of the ThisPoint property.

When entering text, you can use the AutoInc property. If you set the AutoInc property to 1 (on), every time you set a new string, the ThisPoint counter is automatically incremented.

The LegendText property cannot contain more than 80 characters.

Legend text may not be displayed if it is too long to fit on a graph. When this occurs, increase the width of the graph to display the legend text.

### Data Type

**String**

## LineStats Property, Graph Control

Allows statistics lines to be superimposed on the graph. This property is valid for line or log/lin graphs only.

### Syntax

[*form*.]*Graph*.**LineStats**[ = *setting%*]

### Remarks

The following table lists the LineStats property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | None. |
| 1 | Mean. |
| 2 | MinMax. |
| 3 | Mean and MinMax. |
| 4 | StdDev. |
| 5 | StdDev and Mean. |
| 6 | StdDev and MinMax. |
| 7 | StdDev and MinMax and Mean. |
| 8 | BestFit. |
| 9 | BestFit and Mean. |
| 10 | BestFit and MinMax. |
| 11 | BestFit and MinMax and Mean. |
| 12 | BestFit and StdDev. |
| 13 | BestFit and StdDev and Mean. |
| 14 | BestFit and StdDev and MinMax. |
| 15 | All. |

### Data Type

**Integer** (Enumerated)

## NumPoints Property, Graph Control

Specifies the number of data points in each data set.

**Syntax**

[*form*.]*Graph*.**NumPoints**[ = *points%*]

**Remarks**

The minimum value of NumPoints is 2. The default value for this property is 5.

The product of (NumPoints x NumSets) cannot be greater than 3800.

NumPoints can be changed at any time.

If NumPoints is less than the number of data items you have, excess array data is discarded. If NumPoints is greater than the number of data items you have, additional null-value data is created.

**Data Type**

**Integer**

## NumSets Property, Graph Control

Specifies the number of data sets to be graphed.

### Syntax

[*form*.]*Graph*.**NumSets**[ = *sets%*]

### Remarks

The minimum value for NumSets is 1. The default value for this property is 1.

The product of (NumPoints x NumSets) cannot be greater than 3800.

NumSets can be changed at any time.

If NumSets is less than the number of sets of data you have, any excess array data is discarded. If NumSets is greater than the number of data sets, additional null-value data is created.

**Note** Pie charts only use the first data set, even if NumSets > 1.

### Data Type

**Integer**

## Palette Property, Graph Control

Allows you to select a specific set of palette colors.

**Syntax**

[*form*.]*Graph*.**Palette** [ = *setting%*]

**Remarks**

The following table lists the Palette property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Solid |
| 1 | Pastel (dithered) |
| 2 | Grayscale (dithered) |

If the Palette property is set to 1, the color values for the graph change from solid colors to dithered pastel colors. If the Palette property is set to 2, the color values for the graph are changed to the nearest dithered shade of gray equivalent.

**Data Type**

**Integer** (Enumerated)

## PatternData Property, Graph Control

Selects a pattern for solid fills, a line pattern for patterned lines, or a line width (in pixels) for thick lines.

**Syntax**

[*form*.]*Graph*.**PatternData**[ = *pattern%*]

**Remarks**

The PatternData property settings are illustrated in the following figure.

Pattern data values range from 0 to 31. Select one pattern per data set or one pattern per point for pie or bar charts with NumSets = 1.

For illustrations of the PatternData property settings, see the *Custom Control Reference*.

Since this is an array property, the array element you set is determined by the current value of the ThisPoint property.

When you enter data, you can use the AutoInc property. If you set the AutoInc property to 1 (on), every time you set a new value, the ThisPoint counter is automatically incremented.

---

**Note**  Fill patterns 8 through 15 do not exist.

---

**Data Type**

**Integer** (Enumerated)

## PatternedLines Property, Graph Control

Sets the style of the lines connecting the data points.

### Syntax

[*form*.]*Graph*.**PatternedLines**[ = *setting%*]

### Remarks

The following table lists the PatternedLines property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Off |
| 1 | On |

When you set the PatternedLines property to 1 (on), the graph is plotted with dotted lines of pattern 1, unless a different PatternData has been set. For information on different pattern styles, see the PatternData property.

### Data Type

**Integer**

## Picture Property, Graph Control

Example

Passes a graph image directly to a picture control. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*Graph*.**Picture**

### Data Type

**Integer**

■

## Picture Example, Graph Control

The following code puts a copy of the graph currently displayed in Graph1 into Picture1.

```
Picture1.Picture = Graph1.Picture
```

If Picture1 has a different aspect ratio from Graph1, the graph image is stretched or compressed accordingly.

## PrintStyle Property, Graph Control

Selects the print style options when printing the control (DrawMode = 5).

**Syntax**

[*form*.]*Graph*.**PrintStyle**[ = *style%*]

**Remarks**

The following table lists the PrintStyle property settings for the Graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Monochrome |
| 1 | Color |
| 2 | Monochrome with border |
| 3 | Color with border |

The default option temporarily converts the DrawStyle to Monochrome (0) before printing. If you are using a color printer, or have a printer capable of printing gray scales, set PrintStyle = 1.

If you use these options with DrawMode = 5, the graph is printed with the best resolution of your printer. No bitmap is generated.

**Data Type**

**Integer** (Enumerated)

## QuickData Property, Graph Control

Example

Sets or returns all the data in the GraphData array in a single operation. This property is not available at design time.

### Syntax

[*form*.]*Graph*.**QuickData**[ = *data$*]

### Remarks

To assign values to the GraphData array, set this property to a string that contains tab-delimited, numeric values.

To create the string, separate each point in the data set with a tab character (Chr$(9)), and each data set by a CR+LF (Chr$(13) + Chr$(10)).

This property is useful when exchanging data between the graph control and the grid control. The format required by QuickData is the same format used by the grid control's Clip property. In Visual Basic, you assign a grid's data to a GraphData array with a single line of code:

```
Graph1.QuickData = Grid1.Clip
```

**Note** When using QuickData to set the GraphData array, NumPoints and NumSets are automatically set according to the number of points and sets within the QuickData string.

If the format of the QuickData string is incorrect (for example, the data sets do not contain the same number of points), an error will occur. GraphData, NumPoints, and NumSets will not be set.

QuickData must always contain at least one data set with at least two points.

### Data Type

**Integer**

■

## QuickData Example, Graph Control

```
Dim T As String
Dim CL As String
Dim MyDataString As String

T = Chr$(9)
CRLF = Chr$(13) + Chr$(10)
MyDataString = "11" + T + "12" + T + "13" + CRLF + "21" + T + "22" + T + "23"
+ CRLF + "31" + T + "32" + T + "33" + CRLF
Graph1.QuickData = MyDataString
```

## RandomData Property, Graph Control

If you set the RandomData property to 1 (on), it generates random data to be graphed. This is mainly useful at design time, when you want to see how the graph will appear at run time.

**Syntax**

[*form*.]*Graph*.**RandomData**[ = *setting%*]

**Remarks**

The following table lists the RandomData property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | Off |
| 1 | (Default) On |

Random numbers that are generated are never negative. To see the effect of negative values, enter your own data.

**Note** The RandomData property is automatically set to 0 (off) if GraphData values are present. You can override the GraphData values by setting the RandomData property to 1 (on). Setting it to 0 (off) again reinstates the GraphData values. Using DataReset with GraphData (or all data) sets the RandomData property back to 1 (on).

**Data Type**

**Integer**

## SeeThru Property, Graph Control

Example

If you set the SeeThru property to 1 (on), the graph background is not cleared. Instead, whatever was there before you inserted the graph will show through. You can create special effects by drawing a graph over a picture control containing a bitmap. This property is available at run time only.

### Syntax

[*form*.]*Graph*.**SeeThru**[ = *setting%*]

### Remarks

The following table lists the SeeThru property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Off |
| 1 | On |

To function correctly, some programming is necessary. Otherwise, the graph cannot be redrawn if it is covered and then uncovered by another window.

**Note** See-through graphs do not work when DrawMode = 3 (Blit).

### Data Type

**Integer**

▪

## SeeThru Example, Graph Control

Create a picture (Picture1), and then create a graph (Graph1), not as a child of the picture, but directly on your form. Move the graph over the top of the picture, making sure the graph does not entirely cover the picture. Leave a narrow border all the way around to ensure the picture receives paint messages. The BorderStyle should be set to None, or a black line will appear around the area of the graph.

When the picture (Picture1) receives a paint message, it refreshes both itself and the graph (Graph1), ensuring that the graph is still on top of the picture with the picture showing through. The flag is necessary to prevent entering the loop again. The Paint event is triggered by `Picture1.Refresh`.

```
Dim Flag As Integer

Private Sub Form_Load ()
   Flag = 0
   Graph1.SeeThru = 1
End Sub

Private Sub Picture1_Paint ()
   If Flag = 1 Then
      Flag = 0
      Picture1.Refresh
      Graph1.Refresh
   Else
      Flag = 1
   End If
End Sub
```

## SymbolData Property, Graph Control

Selects symbols to be used for line, log/lin, scatter, and polar graphs.

### Syntax

[*form*.]*Graph*.**SymbolData**[ = *symbol%*]

### Remarks

The following table describes the settings for the SymbolData property

| Setting | Description |
|---|---|
| 0 | Cross (+) |
| 1 | Cross (X) |
| 2 | Triangle (up) |
| 3 | Solid Triangle (up) |
| 4 | Triangle (down) |
| 5 | Solid Triangle (down) |
| 6 | Square |
| 7 | Solid Square |
| 8 | Diamond |
| 9 | Solid Diamond |

You should select one symbol per data set. The default setting is 0.

Since this is an array property, the array element you set is determined by the current value of the ThisPoint property.

When you enter data, you can use the AutoInc property. If you set the AutoInc property to 1 (on), every time you set a new value, the ThisPoint counter is automatically incremented.

### Data Type

**Integer** (Enumerated)

## ThickLines Property, Graph Control

Sets the width of the lines. For illustrations, see the *Custom Control Reference*.

### Syntax

[*form*.]*Graph*.**ThickLines**[ = *setting%*]

### Remarks

The following table lists the ThickLines property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Off |
| 1 | On |

When the ThickLines property is set to 1 (on), 3-pixel ■ thick lines are drawn, unless a PatternData property is set. If DrawStyle = 0 (Monochrome), line widths between 2 and 7 pixels (depending on the PatternData property setting) are selected.

### Data Type

**Integer**

## ThisPoint Property, Graph Control

Example

Sets the current point number manually so that a particular data point can be changed.

### Syntax

[*form*.]*Graph*.**ThisPoint**[ = *point%*]

### Remarks

The property settings for ThisPoint are from 1 to NumPoints. Setting ThisPoint overrides the AutoInc setting.

### Data Type

**Integer**

■

## ThisPoint Example, Graph Control

The following code draws a 3D bar graph with 1 data set and 5 points. To try this example, paste this code into the Form_Load event procedure of a form that contains a graph (Graph1).

```
Private Sub Form_Load ()
   Graph1.NumPoints = 5
   Graph1.NumSets = 1
   Graph1.AutoInc = 1
   For I% = 1 to 5
      Graph1.GraphData = i%
   Next I%
   Graph1.ThisPoint = 3
   Graph1.GraphData = 10
   Graph1.GraphType = 4
   Graph1.DrawMode = 2
End Sub
```

## ThisSet Property, Graph Control

Example

Allows you to manually control the current set number so that a particular data set can be changed.

### Syntax

[*form*.]*Graph*.**ThisSet**[ = *set%*]

### Remarks

The property settings for ThisSet are from 1 to NumSets. Setting ThisSet overrides the AutoInc setting. This allows you to address any individual data point when you have multiple data sets.

### Data Type

**Integer**

■

## ThisSet Example, Graph Control

The following code draws a 3D bar graph with 3 data sets with 5 points in each set. To try this example, paste this code into the Form_Load event procedure of a form that contains a graph (Graph1).

```
Private Sub Form_Load ()
   Graph1.NumPoints = 5
   Graph1.NumSets = 3
   Graph1.AutoInc = 1
   For I% = 1 To Graph1.NumPoints * Graph1.NumSets
      Graph1.GraphData = 5
   Next I%
   Graph1.ThisSet = 2
   Graph1.ThisPoint = 3
   Graph1.GraphData = 10
   Graph1.GraphType = 4
   Graph1.DrawMode = 2
End Sub
```

## TickEvery Property, Graph Control

Determines the interval between tick marks on the X axis. The TickEvery value specifies that the tick mark represents *n* data points, where *n* is a value in the range 1 to 1000. The default value for this property is 1.

### Syntax

[*form*.]*Graph*.**TickEvery**[ = *interval%*]

### Remarks

This property is ignored when the XPosData property is set. This means that the TickEvery property never has any effect on scatter graphs, which always have XPosData property values.

If the NumPoints property is less than TickEvery, the X axis of your graph is extended to the value of TickEvery. Also, since there must always be an integral number of ticks, the X axis will be extended to a multiple of TickEvery, if necessary. For example, if NumPoints = 127 and TickEvery = 50, then the X axis is extended to 150.

### Data Type

**Integer**

## Ticks Property, Graph Control

Determines whether axis ticks are displayed.

**Syntax**

[*form*.]*Graph*.**Ticks**[ = *setting%*]

**Remarks**

You can turn ticks on and off separately for the X and Y axes.

This property operates independently of the Labels property. Ticks has no affect on a three-dimensional graph drawn with a cage affect.

The following table lists the Ticks property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Off |
| 1 | On |
| 2 | X ticks |
| 3 | Y ticks |

**Data Type**

**Integer** (Enumerated)

## XPosData Property, Graph Control

Example

Gives an independent X value for a graph.

### Syntax

[*form*.]*Graph*.**XPosData**[ = *xvalue!*]

### Remarks

The property setting for XPosData is any real number.

This property can be set for all graph types except pie and Gantt charts.

Since this is a two-dimensional array property, the array element you set is determined by the current value of the ThisSet and ThisPoint properties.

When you enter data, you can use the AutoInc property. If you set the AutoInc property to 1 (on), every time you set a new value, the ThisSet and ThisPoint counters are automatically incremented.

If you have multiple sets of GraphData, but only one set of XPosData, the graph control automatically applies the single set of XPosData to each set of GraphData.

### Data Type

**Single**

■

## XPosData Example, Graph Control

```
 Sub Form_Load
 Dim I%, J%
 Graph1.AutoInc = 0
 Graph1.NumPoints = 10
 Graph1.NumSets = 2
 For I% = 1 To 2
    Graph1.ThisSet = I%
    For J% = 1 To 10
       Graph1.ThisPoint = J%
       If I% = 1 Then Graph1.GraphData = 5 ■ J%
            If I% = 2 Then Graph1.GraphData = J% ■ 5
            Graph1.XPosData = J%
      Next J%
Next I%
Graph1.DrawMode = 2
```

## YAxisMax, YAxisMin Properties, Graph Control

Specifies the maximum Y-axis value (YAxisMax) and minimum Y-axis value (YAxisMin) on your graph.

### Syntax

[*form*.]*Graph*.**YAxisMax**[ = *max!*]

[*form*.]*Graph*.**YAxisMin**[ = *min!*]

### Remarks

The property settings for YAxisMax and YAxisMin are any real numbers.

These properties are used in combination with YAxisTicks and only take affect when YAxisStyle = 2 (user-defined). For more information, see the YAxisStyle property.

### Data Type

**Single**

## YAxisPos Property, Graph Control

Specifies the position of the Y axis on your graph.

### Syntax

[*form*.]*Graph*.**YAxisPos**[ = *position%*]

### Remarks

The following table lists the YAxisPos property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Y axis is positioned automatically according to your XPosData values. When the values are all positive, the Y axis appears at the leftmost edge of the graph. If the values are all negative, the Y axis appears on the rightmost edge of the graph. |
| 1 | Left. |
| 2 | Right. |

### Data Type

**Integer** (Enumerated)

## YAxisStyle Property, Graph Control

Specifies the method used to scale and range the Y axis on your graph.

**Syntax**

[*form*.]*Graph*.**YAxisStyle**[ = *style%*]

**Remarks**

The following table lists the YAxisStyle property settings for the graph control.

| Setting | Description |
|---|---|
| 0 | (Default) Y-axis range is calculated automatically based on the data to be graphed. The maximum Y-axis value is greater than or equal to the maximum data value. The minimum axis value is 0, or, if the data includes negative values, it is less than or equal to the minimum data value. The Y axis, therefore, always includes the 0 origin. |
| 1 | Variable origin. The maximum Y-axis value is equal to or greater than the maximum data value. The minimum Y-axis value is less than or equal to the minimum data value, whether the data includes negative values or not. The Y axis, therefore, may not include the 0 origin. |
| 2 | User-defined origin. The YAxisMax, YAxisMin, and YAxisTicks properties work together to control the range. |

The variable origin style is useful when you are graphing data with a small variation around a nonzero value. If you use the default style, the variation may not be visible.

Use the user-defined style when you want to present the data in a certain way. For example, to create a series of comparable graphs, you might set the Y-axis range from 1000 to +1000, even though the data values for some graphs are all positive.

**Caution** If your data exceeds the limits of the Y-axis range, the graph is drawn outside of the axes bounds and can result in strange effects.

YAxisTicks specifies the number of ticks from the origin to the greater of the YAxisMax and YAxisMin values, regardless of sign. Because there must always be an integral number of ticks on an axis, the graph will sometimes override the YAxisMin value or YAxisMax value.

In this example, YAxisMax has the greater value: YAxisMax = 300, YAxisMin = 10, and YAxisTicks = 3. The graph places ticks 100 units apart, and the YAxisMin value displayed is 100.

In this example, YAxisMin has the greater value (even though it is negative): YAxisMax = 10, YAxisMin = 300, and YAxisTicks = 3. The YAxisMax value displayed is 100.

**Data Type**

**Integer** (Enumerated)

## YAxisTicks Property, Graph Control

Specifies the number of ticks on the Y axis of your graph.

**Syntax**

[*form*.]*Graph*.**YAxisTicks**[ = *ticks%*]

**Remarks**

Enter a value between 1 (default) and 100, inclusive.

YAxisTicks works in combination with YAxisMax and YAxisMin and is only used when YAxisStyle = 2 (user-defined). For more information, see the YAxisStyle property.

**Data Type**

**Integer**

## Graph Control Extended Version

See Also

The extended version of the Graph control includes the following addition features:

- Rotating graphs
- Hot events for drill-down
- Combo graphs
- Curve fitting
- More graph types
- Extended customization

For more information on the extended version of the Graph control in the Graphics Server Developers Kit, please complete this form and mail or fax it to one of the publishers below.

| USA & International | Germany & Austria | UK & rest of Europe |
|---|---|---|
| Pinnacle Publishing Inc | heilerSoftware | Bits Per Second Ltd |
| PO Box 888, Kent, | Mittlerer Pfad 5 | 14 Regent Hill, Brighton, |
| WA 98035-0888, USA | 70499 Stuttgart | Sussex BN1 3ED, UK |
| Tel: 206/251-1900 | Germany | Tel: 01273 727119 |
| Fax: 206/251-5057 | Tel: 0711 139840 | Fax: 01273 731925 |
| | Fax: 0711 8666301 | |

NAME ______________________________________________

COMPANY ______________________________________________

ADDRESS ______________________________________________

______________________________________________

______________________________________________

FAX ______________________________________________

PHONE ______________________________________________

**See Also**

Graph Control

## Key State Control

Properties Methods Events Constants

You can use the key state control to display or modify the CAPS LOCK, NUM LOCK, INS and SCROLL LOCK keyboard states.

### File Name

KEYSTA16.OCX, KEYSTA32.OCX

### Class Name

mhState

### Remarks

Key state sets or returns the state of certain keys on your keyboard. The Style property determines which key the control affects. At run time, you turn a key on and off by setting the Value property to **True** and **False**, respectively. The user can also change the state of a key at run time by clicking a key state control.

The first 16 controls automatically update their appearance when the user presses the corresponding key. If you create more than 16 controls, the subsequent controls will be visible, however, their appearance will not be updated when the key is pressed.

---

**Distribution Note** When you create and distribute applications that use the key state control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

---

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk(*).

| | | | |
|---|---|---|---|
| AutoSize | HelpContextID | Parent | *Value |
| BackColor | Index | *Style | Visible |
| Container | Left | TabIndex | WhatsThisHelpID |
| DragIcon | MouseIcon | TabStop | *Width |
| DragMode | MousePointer | Tag | |
| Enabled | Name | *TimerInterval | |
| *Height | Object | Top | |

Value is the default value of the control.

**Note** The Name property is equivalent to the CtlName property in Visual Basic 1.0.

**Events**

All of the events for this control are listed in the following table. Events that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk(*).

| | | | |
|---|---|---|---|
| *Change | GotFocus | KeyPress | LostFocus |
| Click | KeyDown | KeyUp | |

**Methods**

All of the methods for this control are listed in the following table.

| | | |
|---|---|---|
| Move | SetFocus | ZOrder |
| Refresh | ShowWhatsThis | |

## Height, Width Properties, Key State Control

Determine the height and width of the key state control.

**Syntax**

[*form*.]*Keystate*.**Height**[ = *setting%*]

[*form*.]*Keystate*.**Width**[ = *setting%*]

**Remarks**

You cannot resize a key state control unless the AutoSize property is set to **False**.

**Data Type**

**Integer**

## Style Property, Key State Control

Determines which keyboard state is associated with the key state control.

**Syntax**

[*form*.]*Keystate*.**Style**[ = *setting%*]

**Remarks**

The following table lists the Style property settings for the key state control.

| Setting | Description |
|---|---|
| 0 | (Default) Capitals lock |
| 1 | Number lock |
| 2 | Insert state |
| 3 | Scroll lock |

**Data Type**

**Integer** (Enumerated)

## TimerInterval Property, Key State Control

Sets or returns the current timer interval setting for all key state controls. The default is 1000 milliseconds.

### Syntax

[*form*.]*Keystate*.**TimerInterval**[ = *milliseconds%*]

### Remarks

This property determines the interval at which the key state is checked. If you are having performance problems, try setting TimerInterval to a higher value.

Only one timer operates all key state controls. If you change the TimerInterval for one control, you are changing it for all of them.

The TimerInterval property cannot be set to a negative value.

### Data Type

**Long**

## Value Property, Key State Control

Sets or returns the current state for the key defined in the Style property. The Value property returns the lock state of the key, not the pressed state. This property is not available at design time.

### Syntax

[*form*.]*Keystate*.**Value**[ = {**True | False**}]

### Remarks

The following table lists the Value property settings for the key state control.

| Setting | Description |
|---|---|
| **False** | Key state is off (for example, Caps Lock is off). |
| **True** | Key state is on (for example, Caps Lock is on). |

### Data Type

**Integer** (Boolean)

## Change Event, Key State Control

Occurs when the Value property changes.

**Syntax**

**Private Sub** *Keystate_***Change** ()

## MAPI Session Control

Properties Methods Constants Error Messages

The messaging application program interface (MAPI) controls allow you to create a mail-enabled Visual Basic MAPI application. There are two MAPI custom controls, MAPI session and MAPI messages. The MAPI session control establishes a MAPI session, and then the MAPI messages control allows the user to perform a variety of messaging system functions.

The MAPI controls are invisible at run time. In addition, there are no events for the controls. To use them, you must specify the appropriate methods.

For these controls to work, MAPI services must be present. MAPI services are provided in MAPI compliant electronic mail systems using Windows version 3.0 or later.

The MAPI session control establishes a messaging session.

### File Name

MSMAPI16.OCX, MSMAPI32.OCX

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control or require special consideration when used with it, are marked with an asterisk (*).

| | | |
|---|---|---|
| *Action | Name | *SessionID |
| *DownloadMail | *NewSession | Tag |
| Index | Object | Top |
| Left | Parent | *UserName |
| *LogonUI | *Password | |

**Methods**

All of the methods for this control are listed in the following table.

| | |
|---|---|
| SignOff | SignOn |

## Action Property (MAPI Session Control)

Determines what action is performed when the MAPI session control is invoked. This property is not available at design time. Setting the Action value at run time invokes the control. The Action property is write-only at run time.

**Note** The Action property is included for compatibility with earlier versions of Visual Basic. For additional functionality, use the new methods listed in the Methods table for the MAPI Session control.

**Syntax**

[*form*.]*MapiSession*.**Action**[ = *setting%*]

**Remarks**

This property is used to select between signing on and signing off from a messaging session. When signing on, a session handle is returned and stored in the SessionID property.

The Action property settings are:

| Setting | Description |
|---|---|
| **SignOn** | Logs user into the account specified by the UserName and Password properties and provides a session handle to the underlying message subsystem. The session handle is stored in the SessionID property.<br>Depending on the value of the NewSession property, the session handle may refer to a newly created session or an existing session. |
| **SignOff** | Ends the messaging session and signs the user off the specified account. |

**Data Type**

Integer (Enumerated)

## DownloadMail Property

Specifies whether new messages are downloaded from the mail server for the designated user.

**Syntax**

[*form*.]*MapiSession*.**DownloadMail**[ = {**True** | **False**}]

**Remarks**

The DownloadMail property settings are:

| Setting | Description |
|---|---|
| **True** | (Default) All new messages from the mail server are forced to the user's Inbox during the sign-on process. A progress indicator is displayed until the message download is complete. |
| **False** | New messages on the server are *not* forced to the user's Inbox immediately, but are downloaded at the time interval set by the user. |

This property can be set to **True** when you want to access the user's complete set of messages when signing on. However, processing time may increase as a result.

**Data Type**

Integer (Boolean)

## LogonUI Property

Specifies whether or not a dialog box is provided for sign-on.

**Syntax**

[*form*.]*MapiSession*.**LogonUI**[ = {**True | False**}]

**Remarks**

The LogonUI property settings are:

| Setting | Description |
|---|---|
| **True** | (Default) A dialog box prompts new users for their user name and password (unless a valid messaging session already exists   see the NewSession property for more information). |
| **False** | No dialog box is displayed. |

The **False** setting is useful when you want to begin a mail session without user intervention, and you already have the account name and password for the user. If insufficient or incorrect values are provided, however, an error is generated.

**Data Type**

Integer (Boolean)

## NewSession Property

Specifies whether a new mail session should be established, even if a valid session currently exists.

**Syntax**

[*form*.]*MapiSession*.**NewSession**[ = {**True | False**}]

**Remarks**

The NewSession property settings are:

| Setting | Description |
|---|---|
| **True** | A new messaging session is established, regardless of whether a valid session already exists. |
| **False** | (Default) Use the existing session established by the user. |

**Data Type**

Integer (Boolean)

## Password Property (MAPI Sessions Control)

Specifies the account password associated with the UserName property.

**Syntax**

[*form*.]*MapiSession*.**Password**[ = *string$*]

**Remarks**

An empty string in this property indicates that a sign-on dialog box with an empty password field should be generated. The default is an empty string.

**Data Type**

String

## SessionID Property (MAPI Sessions Control)

Stores the current messaging session handle. This property is not available at design time, and is read only at run time.

### Syntax

[*form*.]*MapiSession*.**SessionID**

### Remarks

This property is set when you specify the **SignOn** method. The SessionID property contains the unique messaging session handle. The default is 0.

Use this property to set the SessionID property of the MAPI messages control.

### Data Type

Long

## UserName Property

Specifies the account user name.

**Syntax**

[*form*.]*MapiSession*.**UserName**[ = *string$*]

**Remarks**

This property contains the name of the user account desired for sign-on or sign-off. If the LogonUI property is **True**, an empty string in the UserName property indicates that a sign-on dialog box with an empty name field should be generated. The default is an empty string.

**Data Type**

String

## SignOff Method

Ends the messaging session and signs the user off from the account specified by the UserName and Password properties.

### Syntax

[*form*.]*MapiSession*.**SignOff**

## SignOn Method

Logs the user into the account specified by the UserName and Password properties, and provides a session handle to the underlying message subsystem.

**Syntax**

[*form*.]*MapiSession***.SignOn**

**Remarks**

The session handle is stored in the SessionID property. Depending on the value of the NewSession property, the session handle may refer to a newly created session or an existing session.

## MAPI Messages Control

Properties Methods Constants Error Messages

The messaging application program interface (MAPI) controls allow you to create a mail-enabled Visual Basic MAPI application. There are two MAPI custom controls, MAPI session and MAPI messages. The MAPI session control establishes a MAPI session, and then the MAPI messages control allows the user to perform a variety of messaging system functions.

The MAPI controls are invisible at run time. In addition, there are no events for the controls. To use them, you must specify the appropriate methods.

For these controls to work, MAPI services must be present. MAPI services are provided in MAPI compliant electronic mail systems using Windows version 3.0 or later.

The MAPI messages control performs a variety of messaging system functions after a messaging session is established with the MAPI session control.

**Class Name**

MapiMessages

**Remarks**

With the MAPI messages control, you can:

- Access messages currently in the Inbox.
- Compose a new message.
- Add and delete message recipients and attachments.
- Send messages (with or without a supporting user interface).
- Save, copy, and delete messages.
- Display the Address Book dialog box.
- Display the Details dialog box.
- Access attachments, including Object Linking and Embedding (OLE) attachments.
- Resolve a recipient name during addressing.
- Perform reply, reply-all, and forward actions on messages.

Most of the properties of the MAPI messages control can be categorized into four functional areas: address book, file attachment, message, and recipient properties. The file attachment, message, and recipient properties are controlled by the AttachmentIndex, MsgIndex, and RecipIndex properties, respectively.

For example, as the index value changes in the MsgIndex property, all other message, file attachment, and recipient properties change to reflect the characteristics of the specified message. The set of message and recipient properties works the same way. The address book properties specify the appearance of the address book dialog box.

**Message Buffers**

When using the MAPI messages control, you need to keep track of two buffers, the *compose buffer* and the *read buffer*. The read buffer is made up of an indexed set of messages fetched from a user's Inbox. The MsgIndex property is used to access individual messages within this set, starting with a value of 0 for the first message and incrementing by one for each message through the end of the set.

The message set is built using the **Fetch** method. The set includes all messages of type FetchMsgType and is sorted as specified by the FetchSorted property. Previously read messages can be included or left out of the message set with the FetchUnreadOnly property. Messages in the read buffer can't be altered by the user, but can be copied to the compose buffer for alteration.

Messages can be created or edited in the compose buffer. The compose buffer is the active buffer when the MsgIndex property is set to -1. Many of the messaging actions are valid only within the compose buffer, such as sending messages, sending messages with a dialog box, saving messages, or deleting recipients and attachments.

Refer to the object library in the Object Browser for property and error constants for the control.

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control or require special consideration when used with it, are marked with an asterisk (*). (Note that the list order is alphabetic from top to bottom, then left to right.)

| | | |
|---|---|---|
| *Action | *FetchUnreadOnly | *MsgType |
| *AddressCaption | Index | Name |
| *AddressEditFieldCount | *MsgConversationID | Object |
| *AddressLabel | *MsgCount | Parent |
| *AddressModifiable | *MsgDateReceived | *RecipAddress |
| *AddressResolveUI | *MsgID | *RecipCount |
| *AttachmentCount | *MsgIndex | *RecipDisplayName |
| *AttachmentIndex | *MsgNoteText | *RecipIndex |
| *AttachmentName | *MsgOrigAddress | *RecipType |
| *AttachmentPathName | *MsgOrigDisplayName | *SessionID |
| *AttachmentPosition | *MsgRead | Tag |
| *AttachmentType | *MsgReceiptRequested | Top |
| *FetchMsgType | *MsgSent | |
| *FetchSorted | *MsgSubject | |

**Methods**

All of the methods for this control are listed in the following table. (Note that the list order is alphabetic from top to bottom, then left to right.)

| | | |
|---|---|---|
| Compose | Forward | Save |
| Copy | Reply | Send |
| Delete | ReplyAll | Show |
| Fetch | ResolveName | |

## Action Property (MAPI Message Control)

Determines what action is performed when the MAPI messages control is invoked. This property is not available at design time. Setting the Action value at run time invokes the control. This property is write-only at run time.

**Note** The Action property is included for compatibility with earlier versions of Visual Basic. For additional functionality, use the new methods listed in the Methods table for the MAPI Messages control.

**Syntax**

[*form*.]*MapiMessages*.**Action**[ = *setting%*]

**Remarks**

The following table lists the Action property settings from Visual Basic 3.0 and the corresponding new methods in Visual Basic 4.0.

| Action property setting (VB3) | Corresponding method (VB4) |
|---|---|
| MESSAGE_FETCH | Fetch method |
| MESSAGE_SENDDLG | Send method |
| MESSAGE_SEND | Send method |
| MESSAGE_SAVEMSG | Save method |
| MESSAGE_COPY | Copy method |
| MESSAGE_COMPOSE | Compose method |
| MESSAGE_REPLY | Reply method |
| MESSAGE_REPLYALL | ReplyAll method |

| | |
|---|---|
| MESSAGE_FORWARD | Forward method |
| MESSAGE_DELETE | Delete method |
| MESSAGE_SHOWADBOOK | Show method |
| MESSAGE_SHOWDETAILS | Show method |
| MESSAGE_RESOLVENAME | ResolveName method |
| RECIPIENT_DELETE | Delete method |
| ATTACHMENT_DELETE | Delete method |

**Data Type**

Integer

## AddressCaption Property

Specifies the caption appearing at the top of the Address Book dialog box when the **Show** method is specified with the *details* argument missing or set to **False**.

**Syntax**

[*form*.]*MapiMessages*.**AddressCaption**[ = *string$*]

**Remarks**

If this property is a null or empty string, the default value of the Address Book is used.

**Data Type**

String

## AddressEditFieldCount Property

Specifies the number of edit controls available to the user in the Address Book dialog box when the **Show** method is specified with the *details* argument missing or set to **False**.

### Syntax

[*form*.]*MapiMessages*.**AddressEditFieldCount**[ = *setting%*]

### Remarks

The AddressEditFieldCount property settings are**:**

| Setting | Description |
|---|---|
| 0 | No edit controls; only browsing is allowed. |
| 1 | (Default) Only the To edit control should be present in the dialog box. |
| 2 | The To and CC (copy) edit controls should be present in the dialog box. |
| 3 | The To, CC (copy), and BCC (blind copy) edit controls should be present in the dialog box. |
| 4 | Only those edit controls supported by the messaging system should be present in the dialog box. |

For example, if AddressEditFieldCount is 3, the user can select from the To, CC, and BCC edit controls in the Address Book dialog box. The AddressEditFieldCount is adjusted so that it is equal to at least the minimum number of edit controls required by the recipient set.

### Data Type

Integer (Enumerated)

## AddressLabel Property

Specifies the appearance of the To edit control in the Address Book when the **Show** method is specified with the *details* argument missing or set to **False**.

### Syntax

[*form*.]*MapiMessages*.**AddressLabel**[ = *string$*]

### Remarks

This property is normally ignored and should contain an empty string to use the default label "To." However, when the AddressEditFieldCount property is set to 1, the user has the option of explicitly specifying another label (providing the number of editing controls required by the recipient set equals 1).

### Data Type

String

## AddressModifiable Property

Specifies whether the Address Book can be modified.

**Syntax**

[*form*.]*MapiMessages*.**AddressModifiable**[ = {**True** | **False**}]

**Remarks**

The AddressModifiable property settings are:

| Setting | Description |
|---|---|
| **True** | The user is allowed to modify their personal address book. |
| **False** | (Default) The user is not allowed to modify their personal address book. |

**Data Type**

Integer (Boolean)

## AddressResolveUI Property

Specifies whether a dialog box is displayed for recipient name resolution during addressing when the **ResolveName** method is specified.

### Syntax

[*form*.]*MapiMessages*.**AddressResolveUI**[ = {**True** | **False**}]

### Remarks

The AddressResolveUI property settings are:

| Setting | Description |
|---|---|
| **True** | A dialog box is displayed with names that closely match the intended recipient's name. |
| **False** | (Default) No dialog box is displayed for ambiguous names. An error occurs if no potential matches are found (no matches is not an ambiguous situation). |

### Data Type

Integer (Boolean)

## AttachmentCount Property

Specifies the total number of attachments associated with the currently indexed message. This property is not available at design time, and is read-only at run time.

### Syntax

[*form*.]*MapiMessages*.**AttachmentCount**

### Remarks

The default value is 0. The value of AttachmentCount depends on the number of attachments in the current indexed message.

### Data Type

Long

## AttachmentIndex Property

Sets the currently indexed attachment. This property is not available at design time.

**Syntax**

[*form*.]*MapiMessages*.**AttachmentIndex**[ = *index%* ]

**Remarks**

Specifies an index number to identify a particular message attachment. The index number in this property determines the values in the AttachmentFileName, AttachmentPathName, AttachmentPosition, and AttachmentType properties. The attachment identified by the AttachmentIndex property is called the *currently indexed* attachment. The value of AttachmentIndex can range from 0 (the default) to AttachmentCount -1.

To add a new attachment, set the AttachmentIndex to a value greater than or equal to the current attachment count while in the compose buffer (MsgIndex = -1). The AttachmentCount property is updated automatically to reflect the implied new number of attachments.

For example, if the current AttachmentCount property has the value 3, setting the AttachmentIndex property to 4 adds 2 new attachments and increases the AttachmentCount property to 5.

To delete an existing attachment, specify the **Delete** method with the *object* parameter set to 2. Attachments can be added or deleted only when the MsgIndex property is set to -1.

**Data Type**

Long

## AttachmentName Property

Specifies the name of the currently indexed attachment file. This property is not available at design time. It is read-only unless MsgIndex is set to -1.

### Syntax

[*form*.]*MapiMessages*.**AttachmentName**[ = *string$*]

### Remarks

The file name specified is the file name seen by the recipients of the currently indexed message. If AttachmentFileName is an empty string, the file name from the AttachmentPathName property is used.

If the attachment is an OLE object, AttachmentFileName contains the class name of the object, for example, "Microsoft Excel Worksheet."

Attachments in the read buffer are deleted when a subsequent fetch action occurs. The value of AttachmentName depends on the currently indexed message as selected by the AttachmentIndex property.

### Data Type

String

## AttachmentPathName Property

Specifies the full path name of the currently indexed attachment. This property is not available at design time. It is read-only unless MsgIndex is set to -1.

### Syntax

[*form*.]*MapiMessages*.**AttachmentPathName**[ = *string$*]

### Remarks

If you attempt to send a message with an empty string for a path name, an error results. Attachments in the read buffer are deleted when a subsequent fetch action occurs. Attachments in the compose buffer need to be manually deleted. The value of AttachmentPathName depends on the currently indexed message, as selected by the AttachmentIndex property.

### Data Type

String

## AttachmentPosition Property

Specifies the position of the currently indexed attachment within the message body. This property is not available at design time. It is read-only unless MsgIndex is set to -1.

### Syntax

[*form*.]*MapiMessages*.**AttachmentPosition**[ = *position&*]

### Remarks

To determine where an attachment is placed, count the characters in the message body and decide which character position you wish to replace with the attachment. The character count at that position should be used for the AttachmentPosition value.

For example, in a message body that is five-characters long, you could place an attachment at the end of the message by setting AttachmentPosition equal to 4. (The message body occupies character positions 0 to 4.)

You can't place two attachments in the same position within the same message. In addition, you can't place an attachment beyond the end of the message body.

The value of AttachmentPosition depends on the currently indexed message, as selected by the AttachmentIndex property.

### Data Type

Long

## AttachmentType Property

Specifies the type of the currently indexed file attachment. This property is not available at design time. It is read-only unless MsgIndex is set to -1.

### Syntax

[*form*.]*MapiMessages*.**AttachmentType**[ = *type%*]

### Remarks

The AttachmentType property settings are:

| Setting | Description |
|---|---|
| **Data** | The attachment is a data file. |
| **EOLE** | The attachment is an embedded OLE object. |
| **SOLE** | The attachment is a static OLE object. |

The value of AttachmentType depends on the currently indexed message, as selected by the AttachmentIndex property.

### Data Type

Integer (Enumerated)

## FetchMsgType Property

Specifies the message type to populate the message set.

### Syntax

[*form*.]*MapiMessages*.**FetchMsgType**[ = *string$*]

### Remarks

This property determines which message types are added to the message set when the **Fetch** method is specified. A null or empty string in this property specifies an interpersonal message type (IPM), which is the default.

### Data Type

String

## FetchSorted Property

Specifies the message order when populating the message set with messages from the Inbox.

**Syntax**

[*form*.]*MapiMessages*.**FetchSorted**[ = {**True** | **False**}]

**Remarks**

The FetchSorted property settings are:

| Setting | Description |
|---|---|
| **True** | Messages are added to the message set in the order they were received (first in, first out). |
| **False** | (Default) Messages are added in the sort order as specified by the user's Inbox. |

**Data Type**

Integer (Boolean)

## FetchUnreadOnly Property

Determines whether to restrict the messages in the message set to unread messages only.

**Syntax**

[*form*.]*MapiMessages*.**FetchUnreadOnly**[ = {**True** | **False**}]

**Remarks**

The FetchUnreadOnly property settings are:

| Setting | Description |
|---|---|
| **True** | (Default) Only unread messages of the type specified in the FetchMsgType property are added to the message set. |
| **False** | All messages of the proper type in the Inbox are added. |

**Data Type**

Integer (Boolean)

## MsgConversationID Property

Specifies the conversation thread identification value for the currently indexed message. It is read-only unless MsgIndex is set to 1.

**Syntax**

[*form*.]*MapiMessages*.**MsgConversationID**[ = *string$*]

**Remarks**

A conversation thread is used to identify a set of messages beginning with the original message and including all the subsequent replies. Identical conversation IDs indicate that the messages are part of the same thread. New messages are assigned an ID by the message system. The value of MsgConversationID depends on the currently indexed message, as selected by the MsgIndex property.

**Data Type**

String

## MsgCount Property

Indicates the total number of messages present in the message set during the current messaging session. This property is not available at design time, and is read-only at run time.

### Syntax

[*form*.]*MapiMessages*.**MsgCount**

### Remarks

This property is used to get a current count of the messages in the message set. The default value is 0. This property is reset each time a fetch action is performed.

### Data Type

Long

## MsgDateReceived Property

Specifies the date on which the currently indexed message was received. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MapiMessages*.**MsgDateReceived**

### Remarks

The format for this property is YYYY/MM/DD HH:MM. Hours are measured on a standard 24-hour base. The value of MsgDateReceived is set by the message system and depends on the currently indexed message, as selected by the MsgIndex property.

### Data Type

String

## MsgID Property

Specifies the string identifier of the currently indexed message. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*Map*|*iMessages*.**MsgID**

### Remarks

The message-identifier string is a system-specific, nonprintable, 64-character string used to uniquely identify a message. The value of MsgID depends on the currently indexed message, as selected by the MsgIndex property.

### Data Type

String

## MsgIndex Property

Specifies the index number of the currently indexed message. This property is not available at design time.

### Syntax

[*form*.]*MapiMessages*.**MsgIndex**[ = *index&*]

### Remarks

The MsgIndex property determines the values of all the other message-related properties of the MAPI messages control. The index number can range from -1 to MsgCount -1.

---

**Note** Changing the MsgIndex property also changes the entire set of attachments and recipients.

---

The message identified by the MsgIndex property is called the *currently indexed* message. When this index is changed, all of the other message properties change to reflect the characteristics of the indexed message. A value of -1 signifies a message being built in the compose buffer   in other words, an outgoing message.

### Data Type

Long

## MsgNoteText Property

Specifies the text body of the message. This property is not available at design time. It is read-only unless MsgIndex is set to 1.

### Syntax

[*form*.]*MapiMessages*.**MsgNoteText**[ = *string$*]

### Remarks

This property consists of the entire textual portion of the message body (minus any attachments). An empty string indicates no text.

For inbound messages, each paragraph is terminated with a carriage return-line feed pair (0x0d0a). For outbound messages, paragraphs can be delimited with a carriage return 0x0d), line feed 0x0a), or a carriage return-line feed pair (0x0d0a). The value of MsgNoteText depends on the currently indexed message, as selected by the MsgIndex property.

### Data Type

String

## MsgOrigAddress Property

Indicates the mail address of the originator of the currently indexed message. This property is not available at design time and is read-only at run time. The messaging system sets this property for you when sending a message.

### Syntax

[*form*.]*MapiMessages*.**MsgOrigAddress**

### Remarks

The value of MsgOrigAddress depends on the currently indexed message as selected by the MsgIndex property. The value is null in the compose buffer.

### Data Type

String

## MsgOrigDisplayName Property

Specifies the originator's name for the currently indexed message. This property is not available at design time and is read-only at run time. The messaging system sets this property for you.

### Syntax

[*form*.]*MapiMessages*.**MsgOrigDisplayName**

### Remarks

The name in this property is the originator's name, as displayed in the message header. The value of MsgOrigDisplayName depends on the currently indexed message, as selected by the MsgIndex property. The value is null in the compose buffer.

### Data Type

String

## MsgRead Property

Indicates whether the message has already been read. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MapiMessages*.**MsgRead**

### Remarks

The MsgRead property settings are:

| Setting | Description |
|---|---|
| **True** | The currently indexed message has already been read by the user. |
| **False** | (Default) The message remains unread. |

The value of MsgRead depends on the currently indexed message, as selected by the MsgIndex property. The message is marked as read when the note text or any of the attachment information is accessed. However, accessing header information does not mark the message as read.

### Data Type

Integer (Boolean)

## MsgReceiptRequested Property

Specifies whether a return receipt is requested for the currently indexed message. This property is not available at design time.

### Syntax

[*form*.]*MapiMessages*.**MsgReceiptRequested**[ = {**True | False**}]

### Remarks

The MsgReceiptRequested property settings are:

| Setting | Description |
|---|---|
| **True** | A receipt notification is returned to the sender when the recipient opens the message. |
| **False** | (Default) No return receipt is generated. |

The value of MsgReceiptRequested depends on the currently indexed message, as selected by the MsgIndex property.

### Data Type

Integer (Boolean)

## MsgSent Property

Specifies whether the currently indexed message has already been sent to the mail server for distribution. This property is not available at design time and is read-only at run time. The messaging system sets this property for you when sending a message.

### Syntax

[*form*.]*MapiMessages*.**MsgSent**

### Remarks

The MsgSent property settings are:

| Setting | Description |
|---|---|
| **True** | The currently indexed message has already been submitted to the mail server as an outgoing message. |
| **False** | The currently indexed message has not yet been delivered to the server. |

The value of MsgSent depends on the currently indexed message, as selected by the MsgIndex property.

### Data Type

Integer (Boolean)

## MsgSubject Property

Specifies the subject line for the currently indexed message as displayed in the message header. This property is not available at design time. It is read-only unless MsgIndex is set to -1.

### Syntax

[*form*.]*MapiMessages*.**MsgSubject**[ = *string$*]

### Remarks

The value of MsgSubject depends on the currently indexed message, as selected by the MsgIndex property. MsgSubject is limited to 64 characters, including the null character.

### Data Type

String

## MsgType Property

Specifies the type of the currently indexed message. This property is not available at design time. It is read-only unless MsgIndex is set to -1.

### Syntax

[*form*.]*MapiMessages*.**MsgType**[ = *string$*]

### Remarks

The MsgType property is for use by applications other than interpersonal mail (IPM message type). Not all mail systems support message types that are not IPM and may not provide (or may ignore) this parameter.

A null or empty string indicates an IPM message type. The value of MsgType depends on the currently indexed message, as selected by the MsgIndex property. This property is not meant for use as a filter to isolate messages by sender, receipt time, and other categories.

### Data Type

String

## RecipAddress Property

Specifies the electronic mail address of the currently indexed recipient. This property is not available at design time. It is read-only unless MsgIndex is set to -1.

### Syntax

[*form*.]*MapiMessages*.**RecipAddress**[ = *string$*]

### Remarks

The value of RecipAddress depends on the currently indexed recipient, as selected by the RecipIndex property.

### Data Type

String

## RecipCount Property

Specifies the total number of recipients for the currently indexed message. This property is not available at design time, and is read-only at run time.

**Syntax**

[*form*.]*MapiMessages*.**RecipCount**

**Remarks**

The default value is 0. The value of RecipCount depends on the currently indexed message, as selected by the MsgIndex property.

**Data Type**

Long

## RecipDisplayName Property

Specifies the name of the currently indexed recipient. This property is not available at design time. It is read-only unless MsgIndex is set to -1.

### Syntax

[*form*.]*MapiMessages*.**RecipDisplayName**[ = *string$*]

### Remarks

The name in this property is the recipient's name, as displayed in the message header. The value of RecipDisplayName depends on the currently indexed message, as selected by the RecipIndex property. The **ResolveName** method uses the recipient name as it is stored here.

### Data Type

String

## RecipIndex Property

Sets the currently indexed recipient. This property is not available at design time.

### Syntax

[*form*.]*MapiMessages*.**RecipIndex**[ = *index&*]

### Remarks

Specifies an index number to identify a particular message recipient. The index number in this property determines the values in the RecipAddress, RecipCount, RecipDisplayName, and RecipType properties.

The recipient identified by the RecipIndex property is called the *currently indexed* recipient. The value of RecipIndex can range from 0 (the default) to RecipCount -1. When in the read buffer with RecipIndex set to -1, values of the other recipient properties show message originator information. The default setting is 0.

To add a new recipient, set the RecipIndex to a value greater than or equal to the current recipient count while in the compose buffer. The RecipCount property is updated automatically to reflect the implied new number of recipients. For example, if the current RecipCount property has the value 3, setting the RecipIndex property to 4 adds 2 new recipients and increases the RecipCount property to 5.

To delete an existing recipient, specify the **Delete** method with the *object* parameter set to 1. Recipients can be added or deleted only when the MsgIndex property is set to -1.

### Data Type

Long

## RecipType Property

Specifies the type of the currently indexed recipient. This property is not available at design time. It is read-only unless MsgIndex is set to -1.

### Syntax

[*form*.]*MapiMessages*.**RecipType**[ = *setting%*]

### Remarks

The RecipType property settings are:

| Setting | Description |
|---|---|
| **OrigList** | The message originator. |
| **ToList** | The recipient is a primary recipient. |
| **CcList** | The recipient is a copy recipient. |
| **BccList** | The recipient is a blind copy recipient. |

The value of RecipType depends on the currently indexed message, as selected by the RecipIndex property. You cannot set the recipient type to 0 (the message system uses a value of 0 to indicate the message originator.)

### Data Type

Integer

## SessionID Property (MAPI Messages Control)

Stores the current messaging session handle. This property is not available at design time.

**Syntax**

[*form*.]*MapiMessages*.**SessionID**[ = *handle&*]

**Remarks**

This property contains the messaging session handle returned by the SessionID property of the MAPI session control. To associate the MAPI messages control with a valid messaging session, set this property to the SessionID of a MAPI session control that was successfully signed on.

**Data Type**

Long

## Compose Method

Composes a message.

**Syntax**

[*form*.]*MapiMessages***.Compose**

**Remarks**

This method clears all the components of the compose buffer, and sets the MsgIndex property to -1.

## Copy Method

Copies the currently indexed message to the compose buffer.

**Syntax**

[*form*.]*MapiMessages***.Copy**

**Remarks**

This method sets the MsgIndex property to -1.

## Delete Method

Deletes a message, recipient, or attachment.

**Syntax**

[*form*.]*MapiMessages***.Delete** [*object* **As Integer**]

**Remarks**

The values for *object* and their corresponding actions are:

| **Value** | **Description** |
|---|---|
| Missing or 0 | Deletes all components of the currently indexed message, reduces the MsgCount property by 1, and decrements the index number by 1 for each message that follows the deleted message.<br><br>If the deleted message was the last message in the set, this method decrements the MsgIndex property by 1. |
| 1 | Deletes the currently indexed recipient. Automatically reduces the RecipCount property by 1, and decrements the index number by 1 for each recipient that follows the deleted recipient.<br><br>If the deleted recipient was the last recipient in the set, this method decrements the RecipIndex property by 1. |
| 2 | Deletes the currently indexed attachment. Automatically reduces the AttachmentCount property by 1, and decrements the index by 1 for each attachment that follows the deleted attachment.<br><br>If the deleted attachment was the last attachment in the set, this method decrements the AttachmentIndex by 1. |

## Fetch Method

Creates a message set from selected messages in the Inbox.

**Syntax**

[*form*.]*MapiMessages***.Fetch**

**Remarks**

The message set includes all messages in the Inbox which are of the types specifed by the FetchMsgType property. They are sorted as specified by the FetchSorted property. If the FetchUnreadOnly property is set to **True**, only unread messages are included in the message set.

Any attachment files in the read buffer are deleted when a subsequent fetch action occurs.

## Forward Method

Forwards a message.

### Syntax

[*form*.]*MapiMessages***.Forward**

### Remarks

This method copies the currently indexed message to the compose buffer as a forwarded message and adds **FW:** to the beginning of the Subject line. It also sets the MsgIndex property to -1.

## Reply Method

Replies to a message.

### Syntax

[*form*.]*MapiMessages***.Reply**

### Remarks

This method copies the currently indexed message to the compose buffer and adds **RE:** to the beginning of the Subject line. It also sets the MsgIndex property to -1.

The currently indexed message originator becomes the outgoing message recipient.

## ReplyAll Method

Replies to all message recipients.

**Syntax**

[*form*.]*MapiMessages***.ReplyAll**

**Remarks**

This method copies the currently indexed message to the compose buffer and adds **RE:** to the beginning of the Subject line. It also sets the MsgIndex property to -1.

The message is sent to the currently indexed message originator and to all **To:** and **CC:** recipients.

## ResolveName Method

Resolves the name of the currently indexed recipient.

**Syntax**

[*form*.]*MapiMessages***.ResolveName**

**Remarks**

This method searches the address book for a match on the currently indexed recipient name. If no match is found, an error is returned. It does not provide additional resolution of the message originator's name or address.

The AddressResolveUI property determines whether to display a dialog box to resolve ambiguous names.

This method may cause the RecipType property to change.

## Save Method

Saves the message currently in the compose buffer (with MsgIndex = -1).

**Syntax**

[*form*.]*MapiMessages***.Save**

## Send Method

Sends a message.

**Syntax**

[*form*.]*MapiMessages*.**Send** [*dialog* **As Integer**]

**Remarks**

The values for *dialog* and their corresponding actions are:

| Value | Description |
|---|---|
| **True** | Sends a message inside a dialog box. Prompts the user for the various components of the message and submits the message to the mail server for delivery. All message properties associated with the message being built in the compose buffer form the basis for the message dialog box. However, changes made in the dialog box do not alter information in the compose buffer. |
| **False** or missing | Submits the outgoing message to the mail server without displaying a dialog box. An error occurs if you attempt to send a message with no recipients or with missing attachment path names. |

## Show Method

Displays the mail Address Book dialog box or the details of the currently indexed recipient.

**Syntax**

[*form*.]*MapiMessages***.Show** [*details* **As Integer**]

**Remarks**

The values for *details* and their corresponding actions are:

| Value | Description |
|---|---|
| **True** | Displays a dialog box that shows the details of the currently indexed recipient. The amount of information presented in the dialog box is determined by the message system. As a minimum, it contains the display name and address of the recipient. |
| **False** or missing | Displays the mail Address Book dialog box. You can use the address book to create or modify a recipient set. Any changes to the address book outside of the compose buffer are not saved. |

## Masked Edit Control

Properties Methods Events Constants

The masked edit control provides restricted data input as well as formatted data output. This control supplies visual cues about the type of data being entered or displayed. This is what the control looks like as an icon in the Toolbox:

### File Name

MSMASK16.OCX, MSMASK32.OCX

### Class Name

MaskEdBox

### Remarks

The masked edit control generally behaves as a standard text box control with enhancements for optional masked input and formatted output. If you don't use an input mask, the masked edit control behaves much like a standard text box, except for its dynamic data exchange (DDE) capability.

If you define an input mask using the Mask property, each character position in the masked edit control maps to either a placeholder of a specified type or a literal character. Literal characters, or *literals*, give visual cues about the type of data being used. For example, the parentheses surrounding the area code of a telephone number are literals: `(206)`.

If you attempt to enter a character that conflicts with the input mask, the control generates a ValidationError event. The input mask prevents you from entering invalid characters into the control.

The masked edit control has three bound properties: DataChanged, DataField, and DataSource. This means that it can be linked to a data control and display field values for the current record in the recordset. The masked edit control can also write out values to the recordset.

When the value of the field referenced by the DataField property is read, it is converted to a Text property string, if possible. If the recordset is updatable, the string is converted to the data type of the field.

To clear the Text property when you have a mask defined, you first need to set the Mask property to an empty string, and then the Text property to an empty string:

```
MaskedEdit1.Mask = ""
MaskedEdit1.Text = ""
```

When you define an input mask, the masked edit control behaves differently from the standard text box. The insertion point automatically skips over literals as you enter data or move the insertion point.

When you insert or delete a character, all nonliteral characters to the right of the insertion point are shifted, as necessary. If shifting these characters leads to a validation error, the insertion or deletion is prevented, and a ValidationError event is triggered.

Suppose the Mask property is defined as "?###", and the current value of the Text property is "A12." If you attempt to insert the letter "B" before the letter "A," the "A" would shift to the right. Since the second value of the input mask requires a number, the letter "A" would cause the control to generate a ValidationError event.

The masked edit control also validates the values of the Text property at run time. If you set the Text property so that it conflicts with the input mask, the control generates a run-time error.

You may select text in the same way as for a standard text box control. When selected text is deleted, the control attempts to shift the remaining characters to the right of the selection. However, any remaining character that might cause a validation error during this shift is deleted, and no ValidationError event is generated.

Normally, when a selection in the masked edit control is copied onto the Clipboard, the entire selection, including literals, is transferred onto the Clipboard. You can use the ClipMode property to transfer only user-entered data onto the Clipboard   literal characters that are part of the input mask are not copied.

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *AllowPrompt | Enabled | HideSelection | SelLength |
| Appearance | Font | hWnd | SelStart |
| *AutoTab | FontBold | Index | *SelText |
| BackColor | FontItalic | Left | TabIndex |
| BorderStyle | FontName | *Mask | TabStop |
| *ClipMode | FontSize | *MaxLength | Tag |
| *ClipText | FontStrikethru | MouseIcon | *Text |
| Container | *FontUnderline | MousePointer | Top |
| DataChanged | ForeColor | Name | Visible |
| DataField | *Format | Object | WhatsThisHelpID |
| DataSource | *FormattedText | Parent | Width |
| DragIcon | Height | *PromptChar | |
| DragMode | HelpContextID | *PromptInclude | |

Text is the default value of the control

**Events**

All of the events for this control are listed in the following table. Events that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | |
|---|---|---|
| Change | GotFocus | KeyUp |
| DragDrop | KeyDown | LostFocus |
| DragOver | KeyPress | *ValidationError |

**Methods**

All of the methods for this control are listed in the following table.

| | | |
|---|---|---|
| Drag | Refresh | ShowWhatsThis |
| Move | SetFocus | ZOrder |

## AllowPrompt Property

Determines whether or not the prompt character is a valid input character.

### Syntax

[*form*.]*MaskedEdit*.**AllowPrompt** [= {**True | False**}]

### Remarks

The AllowPrompt property settings are as follows:

| Setting | Description |
|---|---|
| **False** | (Default) The prompt character is not a valid input character. A ValidationError event is triggered if you enter the prompt character. |
| **True** | The prompt character is a valid input character. |

For example, suppose you have defined a prompt character of 0, and you want the masked edit control to accept any five digits from 0 to 9. You specify a mask of #####. If the AllowPrompt property is **False** and you enter 0, a ValidationError event occurs. If AllowPrompt is set to **True**, you can enter 0 as a valid input character.

### Data Type

**Integer** (Boolean)

## AutoTab Property

Determines whether or not the next control in the tab order receives the focus as soon as the Text property of the masked edit control is filled with valid data. The Mask property determines whether the values in the Text property are valid.

### Syntax

[*form*.]*MaskedEdit*.**AutoTab**[ = {**True | False**}]

### Remarks

Automatic tabbing occurs only if all the characters defined by the Mask property are entered into the control, the characters are valid, and the AutoTab property is set to **True.**

| Setting | Description |
|---|---|
| **False** | (Default) AutoTab is not on. A ValidationError event occurs when you enter more characters than are defined by the input mask. |
| **True** | AutoTab is on. When you enter all the characters defined by the input mask, focus goes to the next control in the tab sequence, and all subsequent characters entered are handled by the next control. |

The masked edit control is considered filled when you enter the last valid character in the control, regardless of where the character is in the input mask. This property has no effect if the Mask property is set to the empty string ("").

### Data Type

**Integer** (Boolean)

## ClipMode Property

Determines whether to include or exclude the literal characters in the input mask when doing a cut or copy command.

### Syntax

[*form*.]*MaskedEdit*.**ClipMode** [ = *setting%*]

### Remarks

The following table lists the ClipMode property settings for the masked edit control.

| Setting | Description |
|---|---|
| 0 | (Default) Include literals on a cut or copy command. |
| 1 | Exclude literals on a cut or copy command. |

This property has no effect if the Mask property is set to the empty string ("").

### Data Type

**Integer** (Enumerated)

## ClipText Property

Returns the text in the masked edit control, excluding literal characters of the input mask. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MaskedEdit*.**ClipText**

### Remarks

This property acts the same as the SelText property when the Mask property is set to the empty string ("").

### Data Type

**String**

## FontUnderline Property

The masked edit control uses an underline character as a placeholder for user input. Under normal behavior, the underline character disappears when the user enters a valid character. If this property is set to **True**, characters entered in the control remain underlined.

### Syntax

[*form*.]*MaskedEdit.***FontUnderline**[ = {**True | False**}]

### Remarks

The following table lists the FontUnderline property settings for the masked edit control.

| Setting | Description |
|---|---|
| **False** | (Default) Underlined characters in the control disappear when you enter a valid character. |
| **True** | Entered characters are underlined. |

### Data Type

**Integer** (Boolean)

## Format Property

Specifies the format for displaying and printing numbers, dates, times, and text.

**Syntax**

[*form*.]*MaskedEdit*.**Format** [ = *posformat$; negformat$; zeroformat$; nullformat$*]

| Parameter | Description |
|---|---|
| *posformat$* | Expression used to display positive values. |
| *negformat$* | Expression used to display negative values. |
| *zeroformat$* | Expression used to display zero values. |
| *nullformat$* | Expression used to display null or empty values. |

**Remarks**

The Format property defines the format expressions used to display the contents of the control. You can use the same format expressions as defined by the Visual Basic **Format$** function, with the exception that named formats ("On/Off") can't be used.

This property can have from one to four parameters separated by semicolons. If one of the parameters is not specified, the format specified by the first parameter is used. If multiple parameters appear, the appropriate number of separators must be used. For example, to specify *posformat$* and *nullformat$*, use the syntax

[*form*.]*MaskedEdit*.**Format** = *posformat$;;; nullformat$*

The following table shows a number of standard formats available to the user; however, any valid **Format$** expression may be defined.

| Data type | Value | Description |
|---|---|---|
| Number | (Default) Empty string | General Numeric format. Displays as entered. |
| Number | $#,##0.00;($#,##0.00) | Currency format. Uses thousands separator; displays negative numbers enclosed in parentheses. |
| Number | 0 | Fixed number format. Displays at least one digit. |
| Number | #,##0 | Commas format. Uses commas as thousands separator. |
| Number | 0% | Percent format. Multiplies value by 100 and appends a percent sign. |
| Number | 0.00E+00 | Scientific format. Uses standard scientific notation. |
| Date/Time | (Default) c | General Date and Time format. Displays date, time, or both. |
| Date/Time | dddddd | Long Date format. Same as the Long Date setting in the International section of the Microsoft Windows Control Panel. Example: Tuesday, May 26, 1992. |
| Date/Time | dd-mmm-yy | Medium Date format. Example: 26-May-92. |
| Date/Time | ddddd | Short Date format. Same as the Short Date setting in the International section of the Microsoft Windows Control Panel. Example: 5/26/92. |
| Date/Time | ttttt | Long Time format. Same as the Time setting in the International |

| | | |
|---|---|---|
| | | section of the Microsoft Windows Control Panel. Example: 05:36:17 A.M. |
| Date/Time | hh:mm AM/PM | Medium Time format. Example: 05:36 A.M. |
| Date/Time | hh:mm | Short Time format. Example: 05:36. |

**Data Type String**

## FormattedText Property

This is identical to the string displayed in the masked edit control when the control doesn't have the focus. This property is read-only at run time.

**Syntax**

[*form*.]*MaskedEdit.***FormattedText**

**Remarks**

If the Format property is equal to the empty string (""), this property is identical to the Text property, except that it is read-only. If the HideSelection property is set to **False**, the control doesn't display the formatted text when it doesn't have the focus. However, the formatted text is still available through this property.

**Data Type**

**String**

## Mask Property

Determines the input mask for the control.

**Syntax**

[*form*.]*MaskedEdit*.**Mask** [ = *string$*]

**Remarks**

You can define input masks at both design time and run time. However, the following standard, predefined input masks are available at design time.

| Mask | Description |
|---|---|
| Null String | (Default) No mask. Acts like a standard text box. |
| ##-???-## | Medium date (US). Example: 20-May-92 |
| ##-##-## | Short date (US). Example: 05-20-92 |
| ##:## ?? | Medium time. Example: 05:36 AM |
| ##:## | Short time. Example: 17:23 |

The input mask can consist of the following characters.

| Mask character | Description |
|---|---|
| # | Digit placeholder. |
| . | Decimal placeholder. The actual character used is the one specified as the decimal placeholder in your international settings. This character is treated as a literal for masking purposes. |
| , | Thousands separator. The actual character used is the one specified as the thousands separator in your international settings. This character is treated as a literal for masking purposes. |
| : | Time separator. The actual character used is the one specified as the time separator in your international settings. This character is treated as a literal for masking purposes. |
| / | Date separator. The actual character used is the one specified as the date separator in your international settings. This character is treated as a literal for masking purposes. |
| \ | Treat the next character in the mask string as a literal. This allows you to include the '#', '&', 'A', and '?' characters in the mask. This character is treated as a literal for masking purposes. |
| & | Character placeholder. Valid values for this placeholder are ANSI characters in the following ranges: 32-126 and 128-255. |
| > | Convert all the characters that follow to uppercase. |
| < | Convert all the characters that follow to lowercase. |
| A | Alphanumeric character placeholder (entry required). For example: a z, A Z, or 0 9. |
| a | Alphanumeric character placeholder (entry optional). |
| 9 | Digit placeholder (entry optional). For example: 0 9. |
| C | Character or space placeholder (entry optional). |
| ? | Letter placeholder. For example: a z or A Z. |
| Literal | All other symbols are displayed as literals; that is, as themselves. |

When the value of the Mask property is an empty string (""), the control behaves like a standard text box control. When an input mask is defined, underscores appear beneath every placeholder in the mask. You can only replace a placeholder with a character that is of the same type as the one specified in the input mask. If you enter an invalid character, the masked edit control rejects the character and generates a ValidationError event.

**Note** When you define an input mask for the masked edit control and you tab to another control, the

ValidationError event is generated if there are any invalid characters in the masked edit control.

**Data Type**

**String**

## MaxLength Property

Sets or returns the maximum length of the masked edit control.

**Syntax**

[*form*.]*MaskedEdit*.**MaxLength** [ = *setting%*]

**Remarks**

The masked edit field can have a maximum of 64 characters (the valid range for this property is 1 to 64). The default value is set to 64 characters, including literal characters in the input mask.

If the user enters characters beyond the specified maximum length, the control generates a beep.

**Data Type**

**Integer**

## PromptChar Property

Sets or returns the character used to prompt a user for input.

**Syntax**

[*form*.]*MaskedEdit*.**PromptChar** [ = *char$*]

**Remarks**

The underscore character "_" is the default character value for the property. The PromptChar property can only be set to exactly one character.

Use the PromptInclude property to specify whether prompt characters are contained in the Text property.

**Data Type**

**String**

## PromptInclude Property

Specifies whether prompt characters are contained in the Text property value. Use the PromptChar property to change the value of the prompt character.

### Syntax

[*form*.]*MaskedEdit*.**PromptInclude** [ = { **True | False** }]

### Remarks

The following table lists the PromptInclude property settings for the masked edit control.

| Setting | Description |
|---|---|
| **False** | The value of the Text property does not contain any prompt character. |
| **True** | (Default) The value of the Text property contains prompt characters, if any. |

If the masked edit control is bound to a data control, the PromptInclude property affects how the data control reads the bound Text property. If PromptInclude is **False**, the data control ignores any literals or prompt characters in the Text property. In this mode, the value that the data control retrieves from the masked edit control is equivalent to the value of the ClipText property.

If PromptInclude is **True**, the data control uses the value of the Text property as the data value to store.

### Data Type

**Integer** (Boolean)

## SelText Property (Masked Edit Control)

Sets or returns the text contained in the control.

**Syntax**

[*form*.]*MaskedEdit*.**SelText**[ = *string$*]

**Remarks**

If an input mask is not defined for the masked edit control, the SelText property behaves like the standard SelText property for the text box control.

If an input mask is defined and there is selected text in the masked edit control, the SelText property returns a text string. Depending on the value of the ClipMode property, not all the characters in the selected text are returned. If ClipMode is on, literal characters don't appear in the returned string.

When the SelText property is set, the masked edit control behaves as if text was pasted from the Clipboard. This means that each character in *string$* is entered into the control as if the user typed it in.

**Data Type**

**String**

## Text Property (MaskedEdit Control)

Sets or returns the text contained in the control. This property is not available at design time.

**Syntax**

[*form*.]*MaskedEdit*.**Text**[ = *string$*]

**Remarks**

This property sets and retrieves the text in the masked edit control, including literal characters and underscores that are part of the input mask. When setting the text property, the *string$* value must match the characters in the input mask exactly, including literal characters and underscores.

**Note** The ClipMode property setting has no effect on the value of the Text property.

The SelText property provides an easier way of setting the text in the masked edit control.

**Data Type**

**Variant**

## ValidationError Event

Occurs when the masked edit field receives invalid input, as determined by the input mask.

**Syntax**

**Private Sub** *ctlname_***ValidationError**(*InvalidText* **As String**; *StartPosition* **As Integer**)

**Remarks**

*InvalidText* is the value of the Text property, including the invalid character. This means that any placeholders and literal characters used in the input mask are included in *InvalidText*.

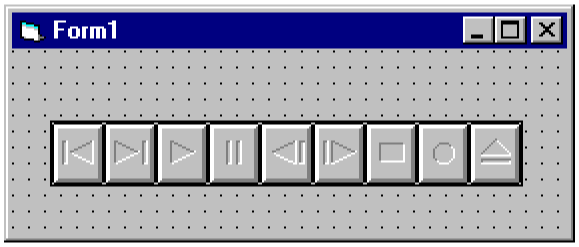
*StartPosition* is the position in *InvalidText* where the error occurred (the first invalid character).

## Multimedia MCI Control

See Also Properties Methods Events Constants Error Messages

The multimedia MCI control manages the recording and playback of multimedia files on Media Control Interface (MCI) devices. Conceptually, this control is a set of push buttons that issues MCI commands to devices such as audio boards, MIDI sequencers, CD-ROM drives, audio CD players, videodisc players, and videotape recorders and players. The MCI control also supports the playback of Video for Windows (*.AVI) files.

When you add the multimedia MCI control to a form at design time, the control appears on the form as follows:



The buttons are defined as Prev, Next, Play, Pause, Back, Step, Stop, Record, and Eject, respectively.

**File Name**

MCI16.OCX, MCI32.OCX

**Class Name**

MMControl

**Remarks**

Your application should already have the MCI device open and the appropriate buttons in the multimedia MCI control enabled by the time the user chooses a button from the multimedia MCI control. In Visual Basic, place the MCI Open command in the Form_Load event.

When you intend to record audio with the multimedia MCI control, open a new file. This action ensures that the data file containing the recorded sound will be in a format compatible with your system's recording capabilities. Also, issue the MCI Save command before closing the MCI device to store the recorded data in the file.

The multimedia MCI control is programmable in several ways:

- The control can be visible or invisible at run time.
- You can augment or completely redefine the functionality of the buttons in the control.
- You can control multiple devices in a form.

If you want to use the buttons in the multimedia MCI control, set the Visible and Enabled properties to **True**. If you do not want to use the buttons in the control, but want to use the multimedia MCI control for its multimedia functionality, set the Visible and Enabled properties to **False**. An application can control MCI devices with or without user interaction.

The events (button definitions) of the multimedia MCI control are programmable. You can augment or completely redefine the functionality of these buttons by developing code for the button events.

The MCI extensions support multiple instances of the multimedia MCI control in a single form to provide concurrent control of several MCI devices. You use one control per device.

---

**Distribution Note** When you create and distribute applications that use the multimedia MCI control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Wizard included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

---

**See Also**

Multimedia MCI

Examples

## Multimedia MCI

Multimedia MCI consists of a set of high-level, device-independent commands that control audio and visual peripherals. The first MCI command you issue is the Open command. This command opens the specified MCI device and identifies the file that will play on the device or be recorded by the device. (Some devices, such as CDAudio, VCR, and videodisc, do not use files and do not require file names.)

Once the device is open, you can issue any of the other MCI commands (Prev, Next, Pause, and so on). The Close command is the last MCI command you issue for the device, returning it to the available pool of system resources. The Close command also closes the data file associated with the device.

For a list of the MCI commands supported by the multimedia MCI control, see the Command property. For additional information on multimedia MCI, refer to either the *Microsoft Multimedia Development Kit Programmer's Workbook* or the *Microsoft Windows Software Development Kit Multimedia Programmer's Reference*.

■

## Examples, Multimedia MCI Control

### Visual Basic Example

The following example illustrates the procedure used to open an MCI device with a compatible data file. By placing this code in the Form_Load procedure, your application can use the multimedia MCI control "as is" to play, record, and rewind the multimedia file GONG.WAV. To try this example, first create a form with a multimedia MCI control.

```
Private Sub Form_Load ()
   ' Set properties needed by MCI to open.
   Form1.MMControl1.Notify = FALSE
   Form1.MMControl1.Wait = TRUE
   Form1.MMControl1.Shareable = FALSE
   Form1.MMControl1.DeviceType = "WaveAudio"
   Form1.MMControl1.FileName = "C:\WINDOWS\MMDATA\GONG.WAV"

   ' Open the MCI WaveAudio device.
   Form1.MMControl1.Command = "Open"
End Sub
```

To properly manage multimedia resources, you should close those MCI devices that are open before exiting your application. You can place the following statement in the Form_Unload procedure to close an open MCI device before exiting from the form containing the multimedia MCI custom control.

```
Private Sub Form_Unload (Cancel As Integer)
   MMControl1.Command = "Close"
End Sub
```

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration, when used with it, are marked with an asterisk (*). Properties beginning with *Button* are defined for each of the nine individual buttons in the multimedia MCI control.

| | | | |
|---|---|---|---|
| *AutoEnable | *Error | Name | *TimeFormat |
| BorderStyle | *ErrorMessage | *Notify | *To |
| *ButtonEnabled | *FileName | *NotifyMessage | Top |
| *ButtonVisible | *Frames | *NotifyValue | *Track |
| *CanEject | *From | Object | *TrackLength |
| *CanPlay | Height | *Orientation | *TrackPosition |
| *CanRecord | HelpContextID | Parent | *Tracks |
| *CanStep | hWnd | *Position | *UpdateInterval |
| *Command | *hWndDisplay | *RecordMode | *UsesWindows |
| Container | Index | *Shareable | *Visible |
| *DeviceID | Left | *Silent | *Wait |
| *DeviceType | *Length | *Start | WhatsThisHelpID |
| DragIcon | *Mode | TabIndex | Width |
| DragMode | MouseIcon | TabStop | |
| *Enabled | MousePointer | Tag | |

**Note** The DragIcon, DragMode, HelpContextID, and Index properties are only available in Visual Basic. The Name property is the equivalent of the CtlName property in Visual Basic 1.0.

## Events

All of the events for this control are listed in the following table. Events that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

Several of the following events are defined for each of the nine individual buttons in the multimedia MCI control. Events defined separately for all nine buttons are described under a heading beginning with *Button*.

| | | | |
|---|---|---|---|
| **Button*Click | **Button*GotFocus | *Done | DragOver |
| **Button*Completed | **Button*LostFocus | DragDrop | *StatusUpdate |

**Note** The DragDrop and DragOver events are only available in Visual Basic.

**Methods**

All of the methods for this control are listed in the following table.

| | | |
|---|---|---|
| Drag | Refresh | ShowWhatsThis |
| Move | SetFocus | ZOrder |

**Note** The **Drag**, **SetFocus**, and **ZOrder** methods are only available in Visual Basic.

## AutoEnable Property, Multimedia MCI Control

Determines if the multimedia MCI control can automatically enable or disable individual buttons in the control. If the AutoEnable property is set to **True**, the multimedia MCI control enables those buttons that are appropriate for the current mode of the specified MCI device type. This property also disables those buttons that the current mode of the MCI device does not support.

### Syntax

[*form*.]*MMControl*.**AutoEnable**[ = {**True | False**}]

### Remarks

The effect of the AutoEnable property is superseded by the Enabled property. The AutoEnable property can automatically enable or disable individual buttons in the control when the multimedia MCI control is enabled (Enabled property set to **True**). When the Enabled property is **False**, keyboard and mouse run-time access to the multimedia MCI control are turned off, regardless of the AutoEnable property setting.

The following table lists the AutoEnable property settings for the multimedia MCI control.

| Setting | Description |
|---|---|
| **False** | Does not enable or disable buttons. The program controls the states of the buttons by setting the Enabled and *Button*Enabled properties. |
| **True** | (Default) Enables buttons whose functions are available and disables buttons whose functions are not. |

The following tables show how the MCI mode settings are reflected in the control's property settings.

Play mode

Record mode

Pause mode

Stop mode

Open mode

Seek or Not Ready modes

The effect of the AutoEnable property supersedes the effects of *Button*Enabled properties. When the Enabled and AutoEnable properties are both **True**, the *Button*Enable properties are not used.

### Data Type

**Integer** (Boolean)

**Play mode**

*Button is enabled if the operation is supported by the open MCI device.

| **Button** | **Status** |
|---|---|
| Back* | Enabled |
| Eject* | Enabled |
| Next | Enabled |
| Pause | Enabled |
| Play* | Disabled |
| Prev | Enabled |
| Record* | Disabled |
| Step* | Enabled |
| Stop | Enabled |

**Record mode**

*Button is enabled if the operation is supported by the open MCI device.

| Button | Status |
|---|---|
| Back* | Enabled |
| Eject* | Enabled |
| Next | Enabled |
| Pause | Enabled |
| Play* | Disabled |
| Prev | Enabled |
| Record* | Disabled |
| Step* | Enabled |
| Stop | Enabled |

**Pause mode**

*Button is enabled if the operation is supported by the open MCI device.

| Button | Status |
|---|---|
| Back* | Enabled |
| Eject* | Enabled |
| Next | Enabled |
| Pause | Enabled |
| Play* | Enabled |
| Prev | Enabled |
| Record* | Enabled |
| Step* | Enabled |
| Stop | Enabled |

**Stop mode**

| Button | Status |
|---|---|
| Back* | Enabled |
| Eject* | Enabled |
| Next | Enabled |
| Pause | Disabled |
| Play* | Enabled |
| Prev | Enabled |
| Record* | Enabled |
| Step* | Enabled |
| Stop | Disabled |

**Open mode**

*Button is enabled if the operation is supported by the open MCI device.

| **Button** | **Status** |
|---|---|
| Back* | Disabled |
| Eject* | Enabled |
| Next | Disabled |
| Pause | Disabled |
| Play* | Disabled |
| Prev | Disabled |
| Record* | Disabled |
| Step* | Disabled |
| Stop | Disabled |

**Seek or Not Ready modes**

*Button is enabled if the operation is supported by the open MCI device.

| Button | Status |
|---|---|
| Back* | Disabled |
| Eject* | Disabled |
| Next | Disabled |
| Pause | Disabled |
| Play* | Disabled |
| Prev | Disabled |
| Record* | Disabled |
| Step* | Disabled |
| Stop | Disabled |

## ButtonEnabled Property, Multimedia MCI Control

Determines if a button in the control is enabled or disabled (dimmed).

**Syntax**

[*form*.]*MMControl*.***Button*****Enabled**[ = {**True | False**}]

**Remarks**

The effects of the *Button*Enabled properties are superseded by the Enabled and AutoEnable properties. Individual *Button*Enabled properties enable or disable the associated buttons in the multimedia MCI control when the multimedia MCI control is enabled (Enabled property set to **True**) and the AutoEnable property is turned off (set to **False**).

For this property, *Button* may be any of the following: Back, Eject, Next, Pause, Play, Prev, Record, Step, or Stop.

The following table lists the *Button*Enabled property settings for the multimedia MCI control.

| Setting | Description |
|---|---|
| **False** | (Default) Disables (dims) the button specified by *Button*. This button's function is not available in the control. |
| **True** | Enables the button specified by *Button*. This button's function is available in the control. |

**Data Type**

**Integer** (Boolean)

## ButtonVisible Property, Multimedia MCI Control

Determines if the specified button is displayed in the control.

**Syntax**

[*form*.]*MMControl*.***Button*****Visible**[ = {**True | False**}]

**Remarks**

The effects of the *Button*Visible properties are superseded by the Visible property. Individual *Button*Visible properties display and hide the associated buttons in the multimedia MCI control when the multimedia MCI control is visible (Visible property set to **True**). If the multimedia MCI control is invisible, these properties are not used.

For this property, *Button* may be any of the following: Back, Eject, Next, Pause, Play, Prev, Record, Step, or Stop.

The following table lists the *Button*Visible property settings for the multimedia MCI control.

| Setting | Description |
|---|---|
| **False** | Does not display the button specified by *Button*. This button's function is not available in the control. |
| **True** | (Default) Displays the button specified by *Button*. |

**Data Type**

**Integer** (Boolean)

## CanEject Property, Multimedia MCI Control

Determines if the open MCI device can eject its media. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl.***CanEject**

### Remarks

The following table lists the CanEject property settings for the multimedia MCI control.

| Setting | Description |
|---|---|
| **False** | (Default) The device cannot eject its media. |
| **True** | The device can eject its media. |

The value of CanEject is retrieved using MCI_GETDEVCAPS during the processing of an Open command.

### Data Type

**Integer** (Boolean)

## CanPlay Property, Multimedia MCI Control

Determines if the open MCI device can play. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**CanPlay**

### Remarks

The following table lists the CanPlay property settings for the multimedia MCI control.

| Setting | Description |
|---|---|
| **False** | (Default) The device cannot play. |
| **True** | The device can play. |

The value of CanPlay is retrieved using MCI_GETDEVCAPS during the processing of an Open command.

### Data Type

**Integer** (Boolean)

## CanRecord Property, Multimedia MCI Control

Determines if the open MCI device can record. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**CanRecord**

### Remarks

The following table lists the CanRecord property settings for the multimedia MCI control.

| Setting | Description |
|---|---|
| **False** | (Default) The device cannot record. |
| **True** | The device can record. |

The value of CanRecord is retrieved using MCI_GETDEVCAPS during the processing of an Open command.

### Data Type

**Integer** (Boolean)

## CanStep Property, Multimedia MCI Control

Determines if the open MCI device can step a frame at a time. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**CanStep**

### Remarks

The following table lists the CanStep property settings for the multimedia MCI control.

| Setting | Description |
|---|---|
| **False** | (Default) The device cannot step a frame at a time. |
| **True** | The device can step a frame at a time. |

Currently only MMMovie, Overlay, and VCR MCI devices can step a frame at a time. Because there is no way to check whether a device can step, programs set the value of this property by checking if the device type is MMMovie, Overlay, or VCR during the processing of an Open command.

### Data Type

**Integer** (Boolean)

## Command Property, Multimedia MCI Control

Specifies an MCI command to execute. This property is not available at design time.

**Syntax**

[*form*.]*MMControl*.**Command**[ = *cmdstring$*]

**Remarks**

The *cmdstring$* argument gives the name of the MCI command to execute: Open, Close, Play, Pause, Stop, Back, Step, Prev, Next, Seek, Record, Eject, Sound, or Save. The command is executed immediately, and the error code is stored in the Error property.

The following table describes each command and lists the properties it uses. If a property is not set, either a default value is used (shown in parentheses following the property name), or the property is not used at all (if no default value is shown).

| Command | Description/Properties used |
|---|---|
| Open | Opens a device using the MCI_OPEN command.<br>Notify (**False**)<br>Wait (**True**)<br>Shareable<br>DeviceType<br>FileName |
| Close | Closes a device using the MCI_CLOSE command.<br>Notify (**False**)<br>Wait (**True**) |
| Play | Plays a device using the MCI_PLAY command.<br>Notify (**True**)<br>Wait (**False**)<br>From<br>To |
| Pause | Pauses playing or recording using the MCI_PLAY command. If executed while the device is paused, tries to resume playing or recording using the MCI_RESUME command.<br>Notify (**False**)<br>Wait (**True**) |
| Stop | Stops playing or recording using the MCI_STOP command.<br>Notify (**False**)<br>Wait (**True**) |
| Back | Steps backwards using the MCI_STEP command.<br>Notify (**False**)<br>Wait (**True**)<br>Frames |
| Step | Steps forwards using the MCI_STEP command.<br>Notify (**False**)<br>Wait (**True**)<br>Frames |
| Prev | Goes to the beginning of the current track using the Seek command. If executed within three seconds of the previous Prev command, goes to the beginning of the previous track or to the beginning of the first track if at the first track.<br>Notify (**False**)<br>Wait (**True**) |
| Next | Goes to the beginning of the next track (if at last track, goes to beginning of last track) using the Seek command.<br>Notify (**False**)<br>Wait (**True**) |

| | |
|---|---|
| Seek | If not playing, seeks a position using the MCI_SEEK command. If playing, continues playing from the given position using the MCI_PLAY command.<br>Notify (**False**)<br>Wait (**True**)<br>To |
| Record | Records using the MCI_RECORD command.<br>Notify (**True**)<br>Wait (**False**)<br>From<br>To<br>RecordMode (0Insert) |
| Eject | Ejects media using the MCI_SET command.<br>Notify (**False**)<br>Wait (**True**) |
| Sound | Plays a sound using the MCI_SOUND command.<br>Notify (**False**)<br>Wait (**False**)<br>FileName |
| Save | Saves an open file using the MCI_SAVE command.<br>Notify (**False**)<br>Wait (**True**)<br>FileName |

**Data Type**

**String**

## DeviceID Property, Multimedia MCI Control

Specifies the device ID for the currently open MCI device. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**DeviceID**[ = *id%*]

### Remarks

The argument *id%* is the device ID of the currently open MCI device. This ID is obtained from MCI_OPEN as a result of an Open command. If no device is open, this argument is 0.

### Data Type

**Integer**

## DeviceType Property, Multimedia MCI Control

Specifies the type of MCI device to open.

### Syntax

[*form*.]*MMControl.***DeviceType**[ = *device$*]

### Remarks

The argument *device$* is the type of MCI device to open: AVIVideo, CDAudio, DAT, DigitalVideo, MMMovie, Other, Overlay, Scanner, Sequencer, VCR, Videodisc, or WaveAudio.

The value of this property must be set when opening simple devices (such as an audio CD that does not use files). It must also be set when opening compound MCI devices when the file-name extension does not specify the device to use.

### Data Type

**String**

## Enabled Property, Multimedia MCI Control

Determines if the control can respond to user-generated events, such as the KeyPress and mouse events.

### Syntax

[*form.*]*MMControl*.**Enabled**[ = {**True | False**}]

### Remarks

This property permits the multimedia MCI control to be enabled or disabled at run time. The effect of the Enabled property supersedes the effects of the AutoEnable and *Button*Enable properties. For example, if the Enabled property is **False**, the multimedia MCI control does not permit access to its buttons, regardless of the settings of the AutoEnable and *Button*Enable properties.

The following table lists the Enabled property settings for the multimedia MCI control

| Setting | Description |
|---|---|
| **False** | All buttons on the control are disabled (dimmed). |
| **True** | (Default) The control is enabled. Use the AutoEnable property to let the multimedia MCI control automatically enable or disable the buttons in the control. Or, use the *Button*Enable properties to enable or disable individual buttons in the control. |

### Data Type

**Integer** (Boolean)

## Error Property, Multimedia MCI Control

Specifies the error code returned from the last MCI command. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**Error**

### Remarks

If the last MCI command did not cause an error, this value is 0.

### Data Type

**Integer**

## ErrorMessage Property, Multimedia MCI Control

Describes the error code stored in the Error property. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**ErrorMessage**

### Data Type

**String**

## FileName Property, Multimedia MCI Control

Specifies the file to be opened by an Open command or saved by a Save command.

**Syntax**

[*form*.]*MMControl*.**FileName**[ = *stringexpression$*]

**Remarks**

The argument *stringexpression$* specifies the file to be opened or saved.

**Data Type**

**String**

## Frames Property, Multimedia MCI Control

Specifies the number of frames the Step command steps forward or the Back command steps backward. This property is not available at design time.

**Syntax**

[*form*.]*MMControl*.**Frames**[ = *frames&*]

**Remarks**

The argument *frames&* specifies the number of frames to step forward or backward.

**Data Type**

**Long**

## From Property, Multimedia MCI Control

Specifies the starting point, using the current time format, for the Play or Record command. This property is not available at design time.

**Syntax**

[*form*.]*MMControl*.**From**[ = *location&*]

**Remarks**

The argument *location&* specifies the starting point for the play or record operation. The current time format is given by the TimeFormat property.

The value you assign to this property is used only with the next MCI command. Subsequent MCI commands ignore the From property until you assign it another (different or identical) value.

**Data Type**

**Long**

## hWndDisplay Property, Multimedia MCI Control

Specifies the output window for MCI MMMovie or Overlay devices that use a window to display output. This property is not available at design time.

### Syntax

[*form*.]*MMControl*.**hWndDisplay**

### Remarks

This property is a handle to the window that the MCI device uses for output. If the handle is 0, a default window (also known as the stage window) is used.

To determine whether a device uses this property, check the UsesWindows property.

In Visual Basic, to get a handle to a control, first use the **SetFocus** method to set the focus to the desired control. Then call the Windows **GetFocus** function. For additional information, see Chapter 25, "Calling Procedures in DLLs," in the *Visual Basic Programmer's Guide.*

To get a handle to a Visual Basic form, use the hWnd property for that form.

### Data Type

**Integer**

## Length Property, Multimedia MCI Control

Specifies, in the current time format, the length of the media in an open MCI device. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**Length**

### Data Type

**Long**

## Mode Property, Multimedia MCI Control

Specifies the current mode of an open MCI device. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**Mode**

### Remarks

The following table lists the Mode property return values for the multimedia MCI control.

| Value | Setting/Device mode |
|---|---|
| 524 | **mciModeNotOpen**<br>Device is not open. |
| 525 | **mciModeStop**<br>Device is stopped. |
| 526 | **mciModePlay**<br>Device is playing. |
| 527 | **mciModeRecord**<br>Device is recording. |
| 528 | **mciModeSeek**<br>Device is seeking. |
| 529 | **mciModePause**<br>Device is paused |
| 530 | **mciModeReady**<br>Device is ready. |

### Data Type

**Long**

## Notify Property, Multimedia MCI Control

Determines if the next MCI command uses MCI notification services. If set to **True**, the Notify property generates a callback event (Done), which occurs when the next MCI command is complete. This property is not available at design time.

### Syntax

[*form*.]*MMControl*.**Notify**[ = {**True | False**}]

### Remarks

The following table lists the Notify property settings for the multimedia MCI control.

| Setting | Description |
|---|---|
| **False** | (Default) The next command does not generate the Done event. |
| **True** | The next command generates the Done event. |

The value assigned to this property is used only with the next MCI command. Subsequent MCI commands ignore the Notify property until it is assigned another (different or identical) value.

**Note** A notification message is aborted when you send a new command that prevents the callback conditions, which were set by a previous command, from being satisfied. For example, to restart a paused device that does not support the MCI Resume command, the multimedia MCI control sends the Play command to the paused device. However, the Play command that restarts the device sets callback conditions, superseding callback conditions and pending notifications from earlier commands.

### Data Type

**Integer** (Boolean)

## NotifyMessage Property, Multimedia MCI Control

Describes the notify code returned in the Done event. This property is not available at design time and is read-only at run time.

**Syntax**

[*form*.]*MMControl*.**NotifyMessage**

**Data Type**

**String**

## NotifyValue Property, Multimedia MCI Control

Specifies the result of the last MCI command that requested a notification. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**NotifyValue**

### Remarks

The following table lists the NotifyValue return values for the multimedia MCI control.

| Value | Setting/Device mode |
|---|---|
| 1 | **mciNotifySuccessful**<br>Command completed successfully. |
| 2 | **mciNotifySuperseded**<br>Command was superseded by another command. |
| 4 | **mciNotifyAborted**<br>Command was aborted by the user. |
| 8 | **mciNotifyFailure**<br>Command failed. |

The program can check the Done event to determine this value for the most recent MCI command.

### Data Type

**Integer** (Enumerated)

## Orientation Property, Multimedia MCI Control

Determines whether buttons on the control are arranged vertically or horizontally.

**Syntax**

[*form*.]*MMControl*.**Orientation**[ = *orientation%*]

**Remarks**

The following table lists the Orientation property settings for the multimedia MCI control.

| Constant | Value | Description |
|---|---|---|
| **mciOrientHorz** | 0 | Buttons are arranged horizontally. |
| **mciOrientVert** | 1 | Buttons are arranged vertically. |

**Data Type**

**Integer** (Enumerated)

## Position Property, Multimedia MCI Control

Specifies, in the current time format, the current position of an open MCI device. This property is not available at design time and is read-only at run time.

### Syntax

[*form.*]*MMControl*.**Position**

### Data Type

**Long**

## RecordMode Property, Multimedia MCI Control

Specifies the current recording mode for those MCI devices that support recording.

### Syntax

[*form*.]*MMControl*.**RecordMode**[ = *mode%*]

### Remarks

The following table lists the RecordMode property settings for the multimedia MCI control.

| Constant | Value | Recording mode |
|---|---|---|
| **mciRecordInsert** | 0 | Insert |
| **mciRecordOverwrite** | 1 | Overwrite |

To determine whether a device supports recording, check the CanRecord property.

A device that supports recording may support either or both of the recording modes. There is no way to check ahead of time which mode a device supports. If recording with a particular mode fails, try the other mode.

WaveAudio devices support Insert mode only.

### Data Type

**Integer** (Enumerated)

## Shareable Property, Multimedia MCI Control

Determines if more than one program can share the same MCI device.

**Syntax**

[*form*.]*MMControl*.**Shareable**[ = {**True | False**}]

**Remarks**

The following table lists the Shareable property settings for the multimedia MCI control.

| Setting | Description |
|---|---|
| **False** | No other controls or applications can access this device. |
| **True** | More than one control or application can open this device. |

**Data Type**

**Integer** (Boolean)

## Silent Property, Multimedia MCI Control

Determines if sound plays.

### Syntax

[*form*.]*MMControl*.**Silent**[ = {**True | False**}]

### Remarks

The following table lists the Silent property settings for the multimedia MCI control.

| Setting | Description |
|---|---|
| **False** | Any sound present is played. |
| **True** | Sound is turned off. |

### Data Type

**Integer** (Boolean)

## Start Property, Multimedia MCI Control

Specifies, in the current time format, the starting position of the current media. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**Start**

### Data Type

**Long**

## TimeFormat Property, Multimedia MCI Control

Specifies the time format used to report all position information.

**Syntax**

[*form*.]*MMControl*.**TimeFormat**[ = *format&*]

**Remarks**

The following table lists the TimeFormat property settings for the multimedia MCI control.

| Value | Setting/Time format |
|---|---|
| 0 | **mciFormatMilliseconds**<br>Milliseconds are stored as a 4-byte integer variable. |
| 1 | **mciFormatHms**<br>Hours, minutes, and seconds are packed into a 4-byte integer. From least significant byte to most significant byte, the individual data values are:<br>Hours (least significant byte)<br>Minutes<br>Seconds<br>Unused (most significant byte) |
| 2 | **mciFormatMsf**<br>Minutes, seconds, and frames are packed into a 4-byte integer. From least significant byte to most significant byte, the individual data values are:<br>Minutes (least significant byte)<br>Seconds<br>Frames<br>Unused (most significant byte) |
| 3 | **mciFormatFrames**<br>Frames are stored as a 4-byte integer variable. |
| 4 | **mciFormatSmpte24**<br>24-frame SMPTE packs the following values in a 4-byte variable from least significant byte to most significant byte:<br>Hours (least significant byte)<br>Minutes<br>Seconds<br>Frames (most significant byte)<br>SMPTE (Society of Motion Picture and Television Engineers) time is an absolute time format expressed in hours, minutes, seconds, and frames. The standard SMPTE division types are 24, 25, and 30 frames per second. |
| 5 | **mciFormatSmpte25**<br>25-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE. |
| 6 | **mciFormatSmpte30**<br>30-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE. |
| 7 | **mciFormatSmpte30Drop**<br>30-drop-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE. |
| 8 | **mciFormatBytes**<br>Bytes are stored as a 4-byte integer variable. |
| 9 | **mciFormatSamples**<br>Samples are stored as a 4-byte integer variable. |
| 10 | **mciFormatTmsf**<br>Tracks, minutes, seconds, and frame are packed in the 4-byte variable from least significant byte to most significant byte: |

Tracks (least significant byte)
Minutes
Seconds
Frames (most significant byte)

Note that MCI uses continuous track numbering.

---

**Note** Not all formats are supported by every device. If you try to set an invalid format, the assignment is ignored.

---

The current timing information is always passed in a 4-byte integer. In some formats, the timing information returned is not really an integer, but single bytes of information packed in the long integer. Properties that access or send information in the current time format are:

| | |
|---|---|
| From | To |
| Length | TrackLength |
| Position | TrackPosition |
| Start | |

## Data Type

**Long** (Enumerated)

## To Property, Multimedia MCI Control

Specifies the ending point, using the current time format, for the Play or Record command. This property is not available at design time.

### Syntax

[*form*.]*MMControl*.**To**[ = *location&*]

### Remarks

The argument *location&* specifies the ending point for the play or record operation. The current time format is given by the TimeFormat property.

The value assigned to this property is used only with the next MCI command. Subsequent MCI commands ignore the To property until it is assigned another (different or identical) value.

### Data Type

**Long**

## Track Property, Multimedia MCI Control

Specifies the track about which the TrackLength and TrackPosition properties return information. This property is not available at design time.

### Syntax

[*form*.]*MMControl*.**Track**[ = *track&*]

### Remarks

The argument *track&* specifies the track number.

This property is used only to get information about a particular track. It has no relationship to the current track.

### Data Type

**Long**

## TrackLength Property, Multimedia MCI Control

Specifies the length, using the current time format, of the track given by the Track property. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**TrackLength**

### Data Type

**Long**

## TrackPosition Property, Multimedia MCI Control

Specifies the starting position, using the current time format, of the track given by the Track property. This property is not available at design time and is read-only at run time.

**Syntax**

[*form*.]*MMControl*.**TrackPosition**

**Data Type**

**Long**

## Tracks Property, Multimedia MCI Control

Specifies the number of tracks available on the current MCI device. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**Tracks**

### Data Type

**Long**

## UpdateInterval Property, Multimedia MCI Control

Specifies the number of milliseconds between successive StatusUpdate events.

### Syntax

[*form*.]*MMControl*.**UpdateInterval**[ = *milliseconds%*]

### Remarks

The argument *milliseconds%* specifies the number of milliseconds between events. If milliseconds is 0, no StatusUpdate events occur.

### Data Type

**Integer**

## UsesWindows Property, Multimedia MCI Control

Determines if the currently open MCI device uses a window for output. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*MMControl*.**UsesWindows**

### Remarks

The following table lists the UsesWindows property return values for the multimedia MCI control.

| Value | Description |
|---|---|
| **False** | The current device does not use a window for output. |
| **True** | The current device uses a window. |

Currently, only MMMovie and Overlay devices use windows for display. Because there is no way to determine whether a device uses windows, the value of UsesWindows is set during processing of an Open command by checking the device type. If the device type is MMMovie, Overlay, or VCR, the device uses windows.

For devices that use windows, you can use the hWndDisplay property to set the window that will display output.

### Data Type

**Integer** (Boolean)

## Visible Property, Multimedia MCI Control

Determines if the multimedia MCI control is visible or invisible at run time.

### Syntax

[*form*.]*MMControl*.**Visible**[ = {**True | False**}]

### Remarks

The effect of the Visible property supersedes the effects of the individual *Button*Visible properties. When the multimedia MCI control is visible, the individual *Button*Visible properties govern the visibility of the associated buttons in the control. When the Visible property is **False**, the entire control is invisible, and the *Button*Visible properties are not used.

The following table lists the Visible property settings for the multimedia MCI control.

| Setting | Description |
|---|---|
| **False** | The control is invisible. |
| **True** | (Default) Each button is visible or hidden individually, depending on its *Button*Visible property. This button's function is still available in the control. |

### Data Type

**Integer** (Boolean)

## Wait Property, Multimedia MCI Control

Determines whether the multimedia MCI control waits for the next MCI command to complete before returning control to the application. This property is not available at design time.

**Syntax**

[*form*.]*MMControl*.**Wait**[ = {**True | False**}]

**Remarks**

The following table lists the Wait property settings for the multimedia MCI control.

| Setting | Description |
|---|---|
| **False** | Multimedia MCI does not wait until the MCI command completes before returning control to the application. |
| **True** | Multimedia MCI waits until the next MCI command completes before returning control to the application. |

The value assigned to this property is used only with the next MCI command. Subsequent MCI commands ignore the Wait property until it is assigned another (different or identical) value.

**Data Type**

**Integer** (Boolean)

## ButtonClick Event, Multimedia MCI Control

Occurs when the user presses and releases the mouse button over one of the buttons in the multimedia MCI control.

### Syntax

**Private Sub** *MMControl*_***Button*****Click (***Cancel* **As Integer**)

### Remarks

*Button* may be any of the following: Back, Eject, Next, Pause, Play, Prev, Record, Step, or Stop.

Each of the *Button*Click events, by default, perform an MCI command when the user chooses a button. The following table lists the MCI commands performed for each button in the control.

| Button | Command |
|---|---|
| Back | MCI_STEP |
| Step | MCI_STEP |
| Play | MCI_PLAY |
| Pause | MCI_PAUSE |
| Prev | MCI_SEEK |
| Next | MCI_SEEK |
| Stop | MCI_STOP |
| Record | MCI_RECORD |
| Eject | MCI_SET with the MCI_SET_DOOR_OPEN parameter |

Setting the *Cancel* parameter for the *Button*Click event to **True** prevents the default MCI command from being performed. The *Cancel* parameter can take either of the following settings.

| Setting | Description |
|---|---|
| **True** | Prevents the default MCI command from being performed. |
| **False** | Performs the MCI command associated with the button after performing the body of the appropriate *Button*Click event. |

The body of an event procedure is performed before performing the default MCI command associated with the event. Adding code to the body of the *Button*Click events augments the functionality of the buttons. If you set the *Cancel* parameter to **True** within the body of an event procedure or pass the value **True** as the argument to a *Button*Click event procedure, the default MCI command associated with the event will not be performed.

**Note** Issuing a Pause command to restart a paused device can end pending notifications from the original Play command if the device does not support the MCI Resume command. The multimedia MCI control uses the MCI Play command to restart devices that do not support the MCI Resume command. Notifications from the Play command that restarts a paused device cancel callback conditions and supersede pending notifications from the original play command.

## ButtonCompleted Event, Multimedia MCI Control

Occurs when the MCI command activated by a multimedia MCI control button finishes.

**Syntax**

**Private Sub** *MMControl_**Button***Completed** (*Errorcode* **As Long**)

**Remarks**

*Button* may be any of the following: Back, Eject, Next, Pause, Play, Prev, Record, Step, or Stop.

The *Errorcode* argument can take the following settings.

| Setting | Description |
|---|---|
| 0 | Command completed successfully. |
| Any other value | Command did not complete successfully. |

If the *Cancel* argument is set to **True** during a *Button*Click event, the *Button*Completed event is not triggered.

## ButtonGotFocus Event, Multimedia MCI Control

Occurs when a button in the multimedia MCI control receives the input focus.

**Syntax**

**Private Sub** *MMControl*_***Button*****GotFocus ()**

**Remarks**

*Button* may be any of the following: Back, Eject, Next, Pause, Play, Prev, Record, Step, or Stop.

## ButtonLostFocus Event, Multimedia MCI Control

Occurs when a button in the multimedia MCI control loses the input focus.

**Syntax**

**Private Sub** *MMControl*_***Button*****LostFocus ()**

**Remarks**

*Button* may be any of the following: Back, Eject, Next, Pause, Play, Prev, Record, Step, or Stop.

## Done Event, Multimedia MCI Control

Occurs when an MCI command for which the Notify property is **True** finishes.

**Syntax**

**Private Sub** *MMControl*_**Done** (*NotifyCode* **As Integer**)

**Remarks**

The *NotifyCode* argument indicates whether the MCI command succeeded. It can take any of the following settings.

| Value | Setting/Result |
|---|---|
| 1 | **mciSuccessful**<br>Command completed successfully. |
| 2 | **mciSuperseded**<br>Command was superseded by another command. |
| 4 | **mciAborted**<br>Command was aborted by the user. |
| 8 | **mciFailure**<br>Command failed. |

## StatusUpdate Event, Multimedia MCI Control

Occurs automatically at intervals given by the UpdateInterval property.

**Syntax**

**Private Sub** *MMControl*_**StatusUpdate ()**

**Remarks**

This event allows an application to update the display to inform the user about the status of the current MCI device. The application can obtain status information from properties such as Position, Length, and Mode.

## Outline Control

See Also Properties Events Methods Constants Error Messages

The outline control is a special type of list box that allows you to display items in a list hierarchically. This is useful for showing directories and files in a file system, which is the technique used by the Windows File Manager.

**File Name**

MSOUTL16.OCX, MSOUTL32.OCX

**Class Name**

Outline

**Remarks**

The outline control displays items in a list box hierarchically. Each item can have subordinate items, which are visually represented by indentation levels. When an item is expanded, its subordinate items are visible; when an item is collapsed, its subordinate items are hidden. Items in the outline control can also display graphical elements to provide visual cues about the state of the item.

**Distribution Note** When you create and distribute applications that use the outline control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Wizard included with Visual Basic provides tools to help you write setup programs that install your applications.

**See Also**

Visual Elements
Hot Spots
Keyboard Interface

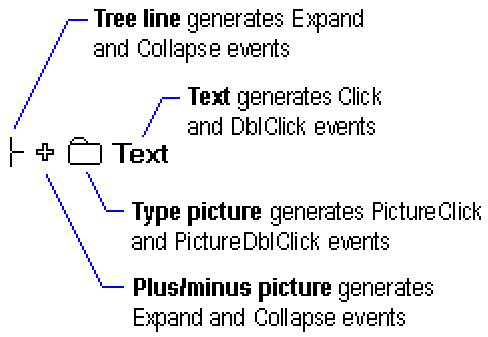
## Visual Elements

The outline control can display graphics and text for each item in a list. An item can have five visual elements:

- **Tree lines**
- vertical and horizontal lines that link items with subordinate items.
- **Indentation**
- an item's level of subordination. Each level of indentation is a level of subordination you specify with the Indent property.
- **Plus/minus pictures**
- indicate whether subordinate items are visible or hidden. When the plus sign is clicked, subordinate items become visible and a minus sign replaces the plus sign. When the minus sign is clicked, the subordinate items are hidden and a plus sign replaces the minus sign.
- **Type pictures**
- indicate the *state* of an item. Type pictures typically show whether an item with subordinate items can be expanded or collapsed. The state of an item is user-defined.
- **Text**
- the string displayed for an item.

## Hot Spots

Each graphical element■tree lines, plus/minus pictures, and type pictures   is a *hot spot* graphic. Clicking a hot spot triggers a special set of events. The following diagram shows an item's possible hot spots.



**Tree line** generates Expand and Collapse events
**Text** generates Click and DblClick events
Text
**Type picture** generates PictureClick and PictureDblClick events
**Plus/minus picture** generates Expand and Collapse events

---

**Note**   To select an item, you must click or double-click the text; you can't select an item by clicking a graphical element.

---

## Keyboard Interface

You can use the keyboard to select items in an outline control's list. The following table lists the keys and their actions.

| This key | Moves focus |
|---|---|
| LEFT ARROW | To the parent item, if the current item is subordinate. |
| RIGHt Arrow | To the first subordinate item, if visible. |
| UP Arrow | To the previous item, if any. |
| DOWN Arrow | To the next item, if any. |
| HOME | To the first item in the list. |
| END | To the last item that is visible. |
| PAGE UP | Backward one page, or to the first item currently displayed. |
| PAGE DOWN | Forward one page, or to the last item currently displayed. |

In addition, you can use two keys to expand and collapse an item that has subordinate items.

| Key | Action |
|---|---|
| + (plus sign) | Expands an item. |
| - (minus sign) | Collapses an item. |

**Properties**

The Properties for this control are listed in the following table. Properties that apply *only* to the outline control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| BackColor | FontUnderline | *ListCount | *PictureType |
| BorderStyle | ForeColor | *ListIndex | *Style |
| Container | *FullPath | MouseIcon | TabIndex |
| DragIcon | *HasSubItems | MousePointer | TabStop |
| DragMode | Height | Name | Tag |
| Enabled | HelpContextID | Object | Text |
| *Expand | hWnd | Parent | Top |
| Font | *Indent | *PathSeparator | *TopIndex |
| FontBold | Index | *PictureClosed | Visible |
| FontItalic | *IsItemVisible | *PictureLeaf | WhatsThisHelpID |
| FontName | ItemData | *PictureMinus | Width |
| FontSize | Left | *PictureOpen | |
| FontStrikethru | *List | *PicturePlus | |

**Note** The DragIcon, DragMode, HelpContextID, Indent, and Parent properties are only available in Visual Basic. The Name property is the same as the CtlName property in Visual Basic 1.0.

**Events**

All the events for this control are listed in the following table. Events that apply *only* to the outline control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| Click | DragOver | KeyPress | MouseMove |
| *Collapse | *Expand | KeyUp | MouseUp |
| DblClick | GotFocus | LostFocus | *PictureClick |
| DragDrop | KeyDown | MouseDown | *PictureDblClick |

**Methods**

All the methods for this control are listed in the following table. Methods that apply *only* to the outline control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | |
|---|---|---|
| *AddItem | Move | SetFocus |
| Clear | Refresh | ShowWhatsThis |
| Drag | *RemoveItem | ZOrder |

**Note** The **Drag**, **SetFocus**, and **ZOrder** methods are only available in Visual Basic.

## Expand Property

Specifies whether an item is expanded (subordinate items visible). If the Expand property is set to **True**, the Expand event will be generated. Not available at design time.

### Syntax

[*form*.]*Outline1***.Expand(***index%***)**[ **=** {**True** | **False**}]

### Remarks

The following table lists the Expand property settings for the outline control.

| Setting | Description |
|---|---|
| **True** | The item has expanded (visible) subordinate items. |
| **False** | The item's subordinate items, if any, are collapsed (hidden). |

The Expand property gives you programmatic control over expanding and collapsing subordinate items. This can be useful when the outline control is context-sensitive in relation to other control values.

If an item is collapsed and you set Expand to **True**, the outline control will generate the run-time error Parent Not Expanded.

### Data Type

**Integer** (Boolean)

## FullPath Property (Outline Control)

Returns the *fully qualified* name of an item. The fully qualified name is the concatenation of the item with its parent item, the parent item's parent item, and so on until the parent item at indentation level 1 is reached. The FullPath property is an array whose index values correspond to the items in the list. Not available at design time and read-only at run time.

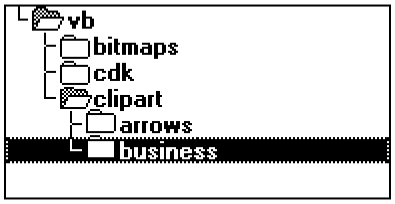
**Syntax**

[*form*.]*Outline1*.**FullPath(***index***)**

**Remarks**

If the first item in the outline control has an indentation level of 0 and is visible, then the FullPath property includes the first item.

Use the PathSeparator property to create a delimiter between the components of the FullPath property. This is useful when the outline control contains file-system components such as directory names and file names.



Using the preceding figure of the outline control, the following code returns the FullPath value of the selected item. First the code sets the PathSeparator property to "\", which means that all items in the value returned by FullPath are delimited by this string. Next, the `FullName$` variable is set to the FullPath property of the currently selected item:

```
Outline1.PathSeparator = "\"
FullName$ = Outline1.FullPath(Outline1.ListIndex)
```

The value of FullName$ is `vb\clipart\business`.

**Data Type**

**String**

## HasSubItems Property

Returns whether an item has subordinate items. The HasSubItems property is an array whose index values correspond to the items in the list. Not available at design time and read-only at run time.

### Syntax

[*form*.]*Outline1*.**HasSubItems(***index***)**

### Remarks

If an item has subordinate items, the HasSubItems property will return **True** regardless of whether the subordinate items are visible. To determine whether a specific item is visible, use the IsItemVisible property.

Use the HasSubItems property to determine which type picture to display for an item. For example, the following code sets a different type picture for each item depending on the return value of HasSubItems:

```
For i = 0 To Outline1.ListCount - 1
   If Outline1.HasSubItems(i) Then
      Outline1.PictureType(i) = outOpen
   Else
      Outline1.PictureType(i) = outLeaf
   End If
Next
```

### Data Type

**Integer** (Boolean)

## Indent Property

Sets and returns the indentation level for the specified index in the list. The Indent property is an array whose index values correspond to the items in the list. Not available at design time.

### Syntax

[*form*.]*Outline1*.**Indent(***index***)**[ **=** *indentation%*]

### Remarks

If the value of *indentation%* is two or more than its parent's indentation level, the run-time error `Bad Outline Indentation` will be generated. For example, if the first item in a list has an indentation value of 1, and you set the second item to an indentation value of 3, the run-time error will occur.

An indentation level of 0 has two meanings. If an item is first, an indentation level of 0 means it is the *root* item in a hierarchy (for example, a drive letter). This is true only for controls whose Style property includes pictures and tree lines. If an item is not first, an indentation level of 0 means it is not visible until the indentation level is greater than 0.

If *index* refers to an item that does not exist, the outline control will automatically add additional items to the list and the ListCount property will be adjusted. For example, notice what happens when you create an outline control and specify the following code in the Form_Load procedure:

```
Private Sub Form_Load ()
   ' Set indentation level.
   Outline1.Indent(3) = 1
End Sub
```

Since *index* refers to 3, the outline control automatically adds 4 items to its list. However, the list will not display any items until you add items to the list with the **AddItem** method, or set items using the List property.

### Data Type

**Integer**

## IsItemVisible Property

Returns whether an item is currently visible. The IsItemVisible property is an array whose index values correspond to the items in the list. Not available at design time and read-only at run time.

### Syntax

[*form*.]*Outline1*.**IsItemVisible(***index***)**

### Data Type

**Integer** (Boolean)

## List Property

Determines the items contained in the control's list portion. The list is a string array in which each element is a list item. Not available at design time.

**Syntax**

[*form*.]*Outline1***.List(***index***)**[ **=** *itemstring$*]

**Remarks**

The outline control's List property is similar to the standard List property for list boxes, except for the following difference: If the *index* of the item doesn't exist, the outline control will automatically add additional items to the list, and the ListCount property will be adjusted. However, the items are not visible until the indentation level is greater than 0.

**Data Type**

**String**

## PathSeparator Property (Outline Control)

Sets and returns the item delimiter string used when accessing the FullPath property. The default value is the backslash character (\).

**Syntax**

[*form*.]*Outline1*.**PathSeparator**[ **=** *delimiter$*]

**Remarks**

For a code example of the PathSeparator property, see the Remarks section for the FullPath property.

**Data Type**

**String**

## PictureClosed, PictureOpen, PictureLeaf Properties

Set and return the type picture associated with the PictureType property. Each item in the outline control has a PictureType equal to 0, 1 or 2. A PictureType of 0 refers to the PictureClosed picture; 1 refers to PictureOpen; 2 refers to PictureLeaf.

### Syntax

[*form*.]*Outline1*.**PictureClosed**[ **=** *picture%*]

[*form*.]*Outline1*.**PictureOpen**[ **=** *picture%*]

[*form*.]*Outline1*.**PictureLeaf**[ **=** *picture%*]

### Remarks

To display a type picture, the Style property must be set to 1, 3, or 5.

The PictureClosed, PictureOpen, and PictureLeaf properties can display either bitmap files (*.BMP) or icon files (*.ICO).

If you don't set a value for PictureClosed, PictureOpen, and PictureLeaf, the outline control will use default pictures. You can also change the picture value at run time (for example, using the return value of the **LoadPicture** statement). In addition, the default bitmaps CLOSED.BMP, OPEN.BMP, and LEAF.BMP are provided in the Visual Basic \BITMAPS\OUTLINE subdirectory.

### Data Type

**Integer**

## PictureMinus, PicturePlus Properties

- PictureMinus
- sets and returns the picture for an item whose subordinate items can be collapsed.
- PicturePlus
- sets and returns the picture for an item whose subordinate items can be expanded.

**Syntax**

[*form*.]*Outline1*.**PictureMinus**[ **=** *picture%*]

[*form*.]*Outline1*.**PicturePlus**[ **=** *picture%*]

**Remarks**

To display plus/minus pictures, the Style property must be set to 2 or 3.

The PictureMinus and PicturePlus properties can display either bitmap files (*.BMP) or icon files (*.ICO).

If you don't set a value for PictureMinus and PicturePlus, the outline control will use default pictures. You can also change the picture value at run time (for example, using the return value of the **LoadPicture** statement). In addition, the default bitmaps MINUS.BMP and PLUS.BMP are provided in the Visual Basic \BITMAPS\OUTLINE subdirectory.

**Data Type**

**Integer**

## PictureType Property

Sets and returns an integer representing the PictureClosed, PictureOpen, or PictureLeaf picture. The PictureType property is an array whose index values correspond to the items in the list. Not available at design time.

**Syntax**

[*form*.]*Outline1*.**PictureType(***index***)**[ **=** *type%*]

**Remarks**

The following table lists the PictureType property settings for the outline control.

| Constant | Value | Description |
|---|---|---|
| **outClosed** | 0 | Use PictureClosed picture. |
| **outOpen** | 1 | Use PictureOpen picture. |
| **outLeaf** | 2 | Use PictureLeaf picture. |

If you don't set a value for PictureClosed, PictureOpen, and PictureLeaf, the outline control will use default pictures.

**Data Type**

**Integer**

## Style Property (Outline Control)

Set and returns the style of graphics and text that appear for each item in the outline control.

**Syntax**

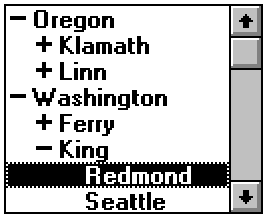
[*form*.]*Outline1*.**Style**[ **=** *style%*]

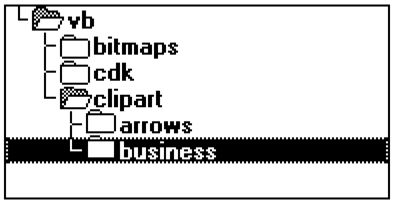
**Remarks**

The following table lists the Style property settings for the outline control.

| Setting | Description |
|---|---|
| 0 | Text only. |
| 1 | Picture and text. |
| 2 | (Default) Plus/minus and text. |
| 3 | Plus/minus, picture, and text. |
| 4 | Tree lines and text. |
| 5 | Tree lines, picture, and text. |

Graphical elements are tree lines, plus/minus pictures, and type pictures. Here are two examples of what you can display:





**Data Type**

**Integer** (Enumerated)

## TopIndex Property

Sets and returns the item that appears in the topmost position in the outline control. If the specified item is not visible because it is collapsed, the next visible item will be set. The default is 0, or the first item. Not available at design time.

### Syntax

[*form*.]*Outline1*.**TopIndex**[ **=** *top%*]

### Data Type

**Integer**

## Collapse Event (Outline Control)

Generated whenever an item is collapsed, which means the item's subordinate items are hidden.

**Syntax**

**Private Sub** *Outline*_**Collapse (**[*Index* **As Integer,** ] *I* **As Integer)**

**Remarks**

This event passes *I*, the index of the item in the list that was closed.

## Expand Event (Outline Control)

Generated whenever an item is expanded, which means the item's subordinate items are visible.

**Syntax**

**Private Sub** *Outline*_**Expand (**[*Index* **As Integer,** ] ***I*** **As Integer)**

**Remarks**

This event passes *I*, the index of the item in the list that was expanded.

You can use the Expand event to change an item's type picture. For example, you can display one picture when an item is expanded, and a different picture when the item is collapsed. The following code displays the PictureOpen picture when the item is expanded:

```
Private Sub Outline1_Expand (I As Integer)
   If Outline1.HasSubItems(I) Then
      Outline1.PictureType(I) = outOpen
   End If
End Sub
```

---

**Note** If you set an item's Expand property to **True**, an Expand event will occur even if the item has no subordinate items.

---

## PictureClick Event

Generated whenever a type picture associated with an item is clicked.

**Syntax**

**Private Sub** *Outline*_**PictureClick (**[*Index* **As Integer,** ] ***I*** **As Integer)**

**Remarks**

This event passes ***I***, the index of the item whose picture was clicked.

## PictureDblClick Event

Generated whenever a type picture associated with an item is double-clicked.

**Syntax**

**Private Sub** *Outline*_**PictureDblClick (**[*Index* **As Integer,** ] *I* **As Integer)**

**Remarks**

This event passes *I*, the index of the item whose picture was double-clicked.

## AddItem Method

Adds an item to the outline control at run time.

**Syntax**

[*form*.]*Outline1***.AddItem** *item* [**,** *index%*]

**Remarks**

If *index%* is specified and refers to an existing item, the new item is inserted into the list, using the existing item's indentation level. However, if *index%* is specified and the item doesn't exist, the item is added with the indentation level set to 0. If an item's indentation level is 0 and it is not the first item in the outline control, the item will not be visible until its indentation level is greater than 0.

If *index%* is not specified, the currently selected item determines where the new item is added. For example, if the ListIndex property is set to 2, the new item is added to the end of the subordinate items for the item whose ListIndex value is 2. In the case where ListIndex is set to -1 (no item selected), the item is added to the end of the list with an indentation level of 1.

## RemoveItem Method

Removes an item and its subordinate items from the outline control at run time.

### Syntax

[*form*.]*Outline1***.RemoveItem** *index%*

### Remarks

When applied to a standard list box or combo box control, the **RemoveItem** method removes only the item specified by the *index%* argument. However, when applied to the outline control, the **RemoveItem** method removes both the specified item and all of its subordinate items.

## Picture Clip Control

Properties    Error Messages

The picture clip control allows you to select an area of a source bitmap and then display the image of that area in a form or picture box. Picture clip controls are invisible at run time. This is a typical bitmap that might be used in the picture clip control:

### File Name

PICCLP16.OCX, PICCLP32.OCX

### Class Name

PictureClip

### Remarks

Picture clip provides an efficient mechanism for storing multiple picture resources. Instead of using multiple bitmaps or icons, create a source bitmap that contains all the icon images required by your application. When you need to access an individual icon, use picture clip to select the region in the source bitmap that contains that icon.

For example, you could use this control to store all the images needed to display a toolbox for your application. It is much more efficient to store all of the toolbox images in a single picture clip control than it is to store each image in a separate picture box. To do this, you first need to create a source bitmap that contains all of the toolbar icons. The preceding picture is an example of such a bitmap.

You can use the following two methods to specify the clipping region in the source bitmap:

- Use the Random Access method to select any portion of the source bitmap as the clipping region. Specify the upper-left corner of the clipping region using the ClipX and ClipY properties. The ClipHeight and ClipWidth properties determine the area of the clipping region. This method is useful when you want to view a portion of a bitmap.
- Use the Enumerated Access method to divide the source bitmap into a specified number of rows and columns. The result is a uniform matrix of picture cells numbered 0, 1, 2, and so on. You can access individual cells with the GraphicCell property. This method is useful when the source bitmap contains a palette of icons that you want to access individually, such as in the preceding bitmap.

Load the source bitmap into the picture clip control using the Picture property. You can only load bitmap (.BMP) files into the picture clip control.

**Distribution Note** When you create and distribute applications that use the picture clip control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| *CellHeight | *ClipY | Name | *StretchY |
| *CellWidth | *Cols | Object | Tag |
| *Clip | *GraphicCell | Parent | *Width |
| *ClipHeight | *Height | *Picture | |
| *ClipWidth | hWnd | *Rows | |
| *ClipX | Index | *StretchX | |

Picture is the default value of the control.

**Note** The Index and Parent properties are only available in Visual Basic. The Name property is the equivalent of the CtlName property in Visual Basic 1.0.

## Clip Property, Picture Clip Control

Example

Returns a bitmap of the area in the picture clip control specified by the ClipX, ClipY, ClipWidth, and ClipHeight properties. This property is read-only at run time.

### Syntax

[*form*.]*PictureClip*.**Clip**

### Remarks

Use this property to specify a clipping region when using the Random Access Method.

When assigning a Clip image to a picture control in Visual Basic, make sure that the ScaleMode property for the picture control is set to 3 (pixels). You must do this since the ClipHeight and ClipWidth properties that define the clipping region are measured in pixels.

### Data Type

**Integer**

▪

## Clip Example, Picture Clip Control

### Visual Basic Example

The following example displays a Clip image in a picture box when the user specifies X and Y coordinates and then clicks a form. First create a form with a picture box, a picture clip control, and two text boxes. At design time, use the Properties window to load a valid bitmap into the picture clip control.

```
Private Sub Form_Click ()
   Dim SaveMode As Integer
   ' Save the current ScaleMode for the picture box.
   SaveMode = Picture1.ScaleMode
   ' Get X and Y coordinates of the clipping region.
   PicClip1.ClipX = Val(Text1.Text)
   PicClip1.ClipY = Val(Text2.Text)
   ' Set the area of the clipping region (in pixels).
   PicClip1.ClipHeight = 100
   PicClip1.ClipWidth = 100
   ' Set the picture box ScaleMode to pixels.
   Picture1.ScaleMode = 3
   ' Set the destination area to fill the picture box.
   PicClip1.StretchX = Picture1.ScaleWidth
   PicClip1.StretchY = Picture1.ScaleHeight
   ' Assign the clipped bitmap to the picture box.
   Picture1.Picture = PicClip1.Clip
   ' Reset the ScaleMode of the picture box.
   Picture1.ScaleMode = SaveMode
End Sub
```

## ClipHeight Property, Picture Clip Control

Specifies the area of the picture clip control to be copied by the Clip property. This property is not available at design time.

### Syntax

[*form*.]*PictureClip*.**ClipHeight**[ = *Height%*]
[*form*.]*PictureClip*.**ClipWidth**[ = *Width%*]
[*form*.]*PictureClip*.**ClipX**[ = *X%*]
[*form*.]*PictureClip*.**ClipY**[ = *Y%*]

### Remarks

This property is measured in pixels.

### Data Type

**Integer**

## ClipWidth Property, Picture Clip Control

Specifies the area of the picture clip control to be copied by the Clip property. This property is not available at design time.

**Syntax**

[*form*.]*PictureClip*.**ClipWidth**[ = *Width%*]

**Remarks**

This property is measured in pixels.

**Data Type**

**Integer**

## ClipX Property, Picture Clip Control

Specifies the area of the picture clip control to be copied by the Clip property. This property is not available at design time.

**Syntax**

[*form*.]*PictureClip*.**ClipX**[ = *X%*]

**Remarks**

This property is measured in pixels.

**Data Type**

**Integer**

## ClipY Property, Picture Clip Control

Specifies the area of the picture clip control to be copied by the Clip property. This property is not available at design time.

**Syntax**

[*form*.]*PictureClip*.**ClipY**[ = *Y%*]

**Remarks**

This property is measured in pixels.

**Data Type**

**Integer**

## Cols, Rows Properties, Picture Clip Control

Set or return the total number of columns or rows in the picture.

### Syntax

[*form*.]*PictureClip*.**Cols**[ = *cols%*]

[*form*.]*PictureClip*.**Rows**[ = *rows%*]

### Remarks

Use these properties to divide the source bitmap into a uniform matrix of picture cells. Use the GraphicCell property to specify individual cells.

A picture clip control must have at least one column and one row.

The height of each graphic cell is determined by dividing the height of the source bitmap by the number of specified rows. Leftover pixels at the bottom of the source bitmap (caused by integer rounding) are clipped.

The width of each graphic cell is determined by dividing the width of the source bitmap by the number of specified columns. Leftover pixels at the right of the source bitmap (caused by integer rounding) are clipped.

### Data Type

**Integer**

## GraphicCell Property, Picture Clip Control

A one-dimensional array of pictures representing all of the picture cells. This property is not available at design time and is read-only at run time.

### Syntax

[*form*.]*PictureClip*.**GraphicCell** (*Index%*)

### Remarks

- Use the Rows and Cols properties to divide a picture into a uniform matrix of graphic cells.
- The cells specified by GraphicCell are indexed, beginning with 0, and increase from left to right and top to bottom.
- Use this property to specify a clipping region when using the Sequential Access method.
- When reading this property, an error is generated when there is no picture or the Rows or Cols property is set to 0.

### Data Type

**Integer**

## Height, Width Properties, Picture Clip Control

Return the height and width (in pixels) of a bitmap displayed in the control. These properties are not available at design time and are read-only at run time.

### Syntax

[*form*.]*PictureClip*.**Height**

[*form*.]*PictureClip*.**Width**

### Remarks

These properties are only valid when the control contains a bitmap.

You can load a bitmap into a picture clip control at design time using the Properties window. In Visual Basic, you can also set this property at run time by using the **LoadPicture** function.

### Data Type

**Integer**

## Picture Property, Picture Clip Control

This property is the same as the standard Visual Basic Picture property except that it only supports bitmap (.BMP) files.

## StretchX, StretchY Properties, Picture Clip Control

Specify the target size for the bitmap created with the Clip property. These properties are not available at design time.

### Syntax

[*form*.]*PictureClip*.**StretchX**[ = *X%*]

[*form*.]*PictureClip*.**StretchY**[ = *Y%*]

### Remarks

Use these properties to define the area to which the Clip bitmap is copied. When the bitmap is copied, it is either stretched or condensed to fit the area defined by StretchX and StretchY.

StretchX and StretchY are measured in pixels.

**Note** In Visual Basic, the default ScaleMode for forms and picture boxes is twips. Set ScaleMode = 3 (pixels) for all controls that display pictures from a picture clip control.

### Data Type

**Integer**

## Spin Button Control

Properties Methods Events Constants Error Messages

Spin button is a spinner control you can use with another control to increment and decrement numbers. You can also use it to scroll back and forth through a range of values or a list of items.

**File Name**

SPIN16.OCX, SPIN32.OCX

**Class Name**

SpinButton

**Remarks**

You can use the spin button control to increment or decrement numbers that are displayed in a text box or other control. At run time, when the user clicks the up (or right) arrow of the spin button, SpinUp events are generated repeatedly until the user releases the mouse. Likewise, when the user clicks the down (or left) arrow, SpinDown events are generated until the user releases the mouse. When using this control, you write code for the SpinUp and SpinDown events that increments or decrements the desired values.

The Delay property determines how often the SpinUp and SpinDown events are generated.

The spin button supports additional color properties that you can set using the Visual Basic Color Palette.

---

**Distribution Note** When you create and distribute applications that use the spin button control, you should install the appropriate file in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

---

**Properties**

All of the properties for this control are listed in the following table. Properties that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| BackColor | ForeColor | MousePointer | *SpinOrientation |
| *BorderColor | Height | Name | TabIndex |
| *BorderThickness | HelpContextID | Object | Tag |
| Container | hWnd | Parent | *TdThickness |
| *Delay | Index | *ShadeColor | Top |
| DragIcon | Left | *ShadowBackColor | Visible |
| DragMode | *LightColor | *ShadowForeColor | WhatsThisHelpID |
| Enabled | MouseIcon | *ShadowThickness | Width |

**Note** The DragIcon, DragMode, HelpContextID, and Index properties are only available in Visual Basic. The Name property is the equivalent of the CtlName property in Visual Basic 1.0.

**Events**

All of the events for this control are listed in the following table. Events that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*).

| | | | |
|---|---|---|---|
| DragDrop | DragOver | *SpinDown | *SpinUp |

**Note** The DragDrop and DragOver events are only available in Visual Basic.

## Methods

All of the methods for this control are listed in the following table. For documentation of the methods not unique to this control, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

| | | |
|---|---|---|
| Drag | Refresh | ZOrder |
| Move | ShowWhatsThis | |

**Note** The **Drag** and **ZOrder** methods are only available in Visual Basic.

## BorderColor Property, Spin Button Control

Determines the color of the border drawn around the control.

### Syntax

[*form*.]*SpinButton*.**BorderColor**[ = *color&*]

### Remarks

The following table lists the BorderColor property settings for the spin button control.

| Setting | Description |
|---|---|
| &H00000000& | (Default) Black. |
| (color) | In Visual Basic, color specified by using the RGB scheme or the **QBColor** function in code. |

### Data Type

**Long**

## BorderThickness Property, Spin Button Control

Sets the width of the border.

**Syntax**

[*form.*]*SpinButton*.**BorderThickness**[ = *setting%*]

**Remarks**

The following table lists the BorderThickness property settings for the spin button control.

| Setting | Description |
|---|---|
| 0 | No border. |
| 1 | (Default) 1-pixel border. |
| (integer) | Width of three-dimensional border, in pixels. |

**Data Type**

**Integer**

## Delay Property, Spin Button Control

Sets the delay between SpinUp or SpinDown events.

The Delay property slows the number of SpinUp or SpinDown events generated when the user clicks one of the arrows in a spin button and then continues to hold down the button.

### Syntax

[*form*.]*SpinButton*.**Delay**[ = *setting%*]

### Remarks

The following table lists the Delay property settings for the spin button control.

| Setting | Description |
|---|---|
| 250 | (Default) 250 milliseconds, 1/4 of a second. |
| (0 ■ 32767) | Milliseconds delay between events. |

### Data Type

**Integer**

## LightColor Property, Spin Button Control

Sets the color of a narrow margin located along the left and upper edges of the control.

### Syntax

[*form*.]*SpinButton*.**LightColor**[ = *color&*]

### Remarks

The following table lists the LightColor property settings for the spin button control.

| Setting | Description |
|---|---|
| &H00FFFFFF& | (Default) White. |
| (color) | In Visual Basic, color specified by using the RGB scheme or the **QBColor** function in code. |

Setting the LightColor property to a lighter shade of the same color as the ShadeColor property generates a raised visual effect. Setting it to a darker shade of the same color as the ShadeColor generates an inset visual effect.

**Note** To see the effect of the LightColor and ShadeColor properties, you should set the TdThickness property to a value greater than 1.

### Data Type

**Long**

## ShadeColor Property, Spin Button Control

Sets the color of a narrow margin that is located along the right and lower edges of the control.

**Syntax**

[*form*.]*SpinButton*.**ShadeColor**[ = *color&*]

**Remarks**

The following table lists the ShadeColor property settings for the spin button control.

| Setting | Description |
|---|---|
| &H007F7F7F& | (Default) Dark gray. |
| (color) | In Visual Basic, color specified by using the RGB scheme or the **QBColor** function in code. |

This property is used with the LightColor property to generate a raised or inset effect.

**Note** To see the effect of the LightColor and ShadeColor properties, you should set the TdThickness property to a value greater than 1.

**Data Type**

**Long**

## ShadowBackColor Property, Spin Button Control

Sets the background color for the shadow effect.

**Syntax**

[*form.*]*SpinButton*.**ShadowBackColor**[ = *color&*]

**Remarks**

The following table lists the ShadowBackColor property settings for the spin button control.

| Setting | Description |
|---|---|
| &H00FFFFFF& | (Default) White. |
| (color) | In Visual Basic, color specified by using the RGB scheme or the **QBColor** function in code. |

ShadowBackColor is usually set to the same color as the surrounding area. It is visible at the lower-left and upper-right areas of the shadow area where the shadow does not cover the background.

**Note** To see the effect of the ShadowBackColor and ShadowForeColor properties, you should set the ShadowThickness property to a value greater than 0.

**Data Type**

**Long**

## ShadowForeColor Property, Spin Button Control

Sets the color of the shadow effect.

**Syntax**

[*form.*]*SpinButton*.**ShadowForeColor**[ = *color&*]

**Remarks**

The following table lists the ShadowForeColor property settings for the spin button control.

| Setting | Description |
|---|---|
| &H007F7F7F& | (Default) Dark Gray. |
| (color) | In Visual Basic, color specified by using the RGB scheme or the **QBColor** function in code. |

ShadowForeColor makes a control appear as if it were floating above the surrounding surface. Usually a control will have a dark shadow, but you can use a variation of the underlying form color. For example, in Visual Basic, if the form's BackColor property is set to green, you may want to set the ShadowForeColor property to a darker shade of green.

**Note**    To see the effect of the ShadowBackColor and ShadowForeColor properties, you should set the ShadowThickness property to a value greater than 0.

**Data Type**

**Long**

## ShadowThickness Property, Spin Button Control

Sets the width of the shadow effect.

### Syntax

[*form.*]*SpinButton*.**ShadowThickness**[ = *setting%*]

### Remarks

The following table lists the ShadowThickness property settings for the spin button control.

| Setting | Description |
|---|---|
| 0 | (Default) No shadow. |
| (integer) | Width of shadow in pixels. |

ShadowThickness varies the width of the shadow to give the control a floating appearance. The floating effect is most realistic when ShadowThickness is just a few pixels.

**Note** To see the effect of the ShadowBackColor and ShadowForeColor properties, you should set the ShadowThickness property to a value greater than 0.

### Data Type

**Integer**

## SpinOrientation Property, Spin Button Control

Sets the direction of the spin control arrows.

**Syntax**

[*form.*]*SpinButton*.**SpinOrientation**[ = *setting%*]

**Remarks**

The following table lists the SpinOrientation property settings for the spin button control.

| Setting | Description |
|---|---|
| 0 | (Default) Vertical: up and down arrows. |
| 1 | Horizontal: left and right arrows. |

**Data Type**

**Integer**

## TdThickness Property, Spin Button Control

Sets the width of the LightColor and the ShadeColor borders.

### Syntax

[*form*.]*SpinButton*.**TdThickness**[ = *setting%*]

### Remarks

The following table lists the TdThickness property settings for the spin button control.

| Setting | Description |
|---|---|
| 0 | (Default) No three-dimensional effect. |
| (integer) | Width of three-dimensional margin in pixels. |

**Note** To see the effect of the LightColor and ShadeColor properties, you should set the TdThickness property to a value greater than 1.

### Data Type

**Integer**

## SpinUp, SpinDown Events, Spin Button Control

Example

Occur when the user clicks one of the arrows of a spin button.

### Syntax

**Private Sub** *SpinButton_***SpinUp ()**

**Private Sub** *SpinButton_***SpinDown ()**

### Remarks

SpinUp is generated when the up or the right arrow is clicked. SpinDown is generated by clicking the down or the left arrow. The arrows can be clicked once to send a single Spin event, or the left mouse button can be held down to generate multiple events.

Holding down the mouse button allows the user to cycle through a range of values. The Delay property slows the rate of cycling.

If you change the value or contents of a control in response to a Spin event, you must also call that control's **Refresh** method to insure that the updated value is displayed.

■
## SpinUp, SpinDown Example, Spin Button Control

### Visual Basic Example

The following examples illustrate how a number is incremented or decremented in a control containing text. To run these examples, create a form with a spin button and a text box.

```
Private Sub Spin1_SpinUp ()
   ' Increment the value in the text box on every SpinUp event.
   Text1.Text = Str$(Val(Text1.Text)+1)
   ' Display the current value in the text box.
   Text1.Refresh
End Sub

Private Sub Spin1_SpinDown ()
   ' Decrement the value in the text box on every SpinDown event.
   Text1.Text = Str$(Val(Text1.Text)-1)
   ' Display the current value in the text box.
   Text1.Refresh
End Sub
```

# 3D Controls Constants

See Also

### AlignTo Constants

| Constant | Value | Description |
|---|---|---|
| **ssTextRight** | 0 | Text to right. |
| **ssTextLeft** | 1 | Text to left. |

### AlignFrameText Constants

| Constant | Value | Description |
|---|---|---|
| **ssLeftJustify** | 0 | Left align text. |
| **ssRightJustify** | 1 | Right align text. |
| **ssCenter** | 2 | Center text. |

### AlignPanelText Constants

| Constant | Value | Description |
|---|---|---|
| **ssLeftTop** | 0 | Text to left and top. |
| **ssLeftMiddle** | 1 | Text to left and middle. |
| **ssLeftBottom** | 2 | Text to left and bottom. |
| **ssRightTop** | 3 | Text to right and top. |
| **ssRightMiddle** | 4 | Text to right and middle. |
| **ssRightBottom** | 5 | Text to right and bottom. |
| **ssCenterTop** | 6 | Text to center and top. |
| **ssCenterMiddle** | 7 | Text to center and middle. |
| **ssCenterBottom** | 8 | Text to center and bottom. |

### AutoSizeButton Constants

| Constant | Value | Description |
|---|---|---|
| **ssNone** | 0 | No autosizing. |
| **ssPictureToButton** | 1 | Autosize picture to button. |
| **ssButtonToPicture** | 2 | Autosize button to picture. |

### AutoSizePanel Constants

| Constant | Value | Description |
|---|---|---|
| **ssNone** | 0 | No autosizing. |
| **ssWidthToCaption** | 1 | Autosize panel width to caption. |
| **ssHeightToCaption** | 2 | Autosize panel height to caption. |
| **ssChildToPanel** | 3 | Autosize child form to panel. |

### Bevel Constants

| Constant | Value | Description |
|---|---|---|
| **ssNone** | 0 | No inner or outer bevel. |
| **ssInset** | 1 | Inset inner or outer bevel. |
| **ssRaised** | 2 | Raised inner or outer bevel. |

### FloodType Constants

| Constant | Value | Description |
|---|---|---|
| **ssNone** | 0 | No flood. |
| **ssLeftToRight** | 1 | Flood from left to right. |
| **ssRightToLeft** | 2 | Flood from right to left. |

| | | |
|---|---|---|
| **ssTopToBottom** | 3 | Flood from top to bottom. |
| **ssBottomToTop** | 4 | Flood from bottom to top. |
| **ssWideningCircle** | 5 | Flood in widening circle. |

**Font3D Constants**

| Constant | Value | Description |
|---|---|---|
| **ssNone** | 0 | No 3-D text font. |
| **ssRaisedLight** | 1 | Font raised with light shading. |
| **ssRaisedHeavy** | 2 | Font raised with heavy shading. |
| **ssInsetLight** | 3 | Font inset with light shading. |
| **ssInsetHeavy** | 4 | Font inset with heavy shading. |

**PictureDnChange Constants**

| Constant | Value | Description |
|---|---|---|
| **ssNoChange** | 0 | Use Up bitmap with no change. |
| **ssDither** | 1 | Dither Up bitmap. |
| **ssInvert** | 2 | Invert Up bitmap. |

**Shadow Color Constants**

| Constant | Value | Description |
|---|---|---|
| **ssDarkGrey** | 0 | Dark gray shadow. |
| **ssBlack** | 1 | Black shadow. |

**ShadowStyle Constants**

| Constant | Value | Description |
|---|---|---|
| **ssInset** | 0 | Shadow inset. |
| **ssRaised** | 1 | Shadow raised. |

## Animated Button Control Constants

See Also

### Cycle Constants

| Constant | Value | Description |
|---|---|---|
| **aniHalfHalf** | 0 | Animated button display. |
| **aniByFrame** | 1 | Automatic multistate display. |
| **aniTwoStateHalfHalf** | 2 | Two-state display. |

### ClickFilter Constants

| Constant | Value | Description |
|---|---|---|
| **aniAnywhere** | 1 | Mouse clicks detected anywhere. |
| **aniTextOrPicture** | 2 | Mouse clicks detected on caption text or image frame. |
| **aniPictureOnly** | 3 | Mouse clicks detected on image frame. |
| **aniTextOnly** | 4 | Mouse clicks detected on caption text. |

### PictDrawMode Constants

| Constant | Value | Description |
|---|---|---|
| **aniAsDefined** | 0 | Positions image at X and Y settings. |
| **aniAutoSize** | 1 | Automatically controls sizing mode. |
| **aniStretch** | 2 | Stretches image to fit button. |

### TextPos Constants

| Constant | Value | Description |
|---|---|---|
| **aniTextOnPicture** | 0 | Positions caption at X and Y settings. |
| **aniTextLeft** | 1 | Positions image at left of control. |
| **aniTextRight** | 2 | Positions image at right of control. |
| **aniTextBelow** | 3 | Positions image at bottom of control. |
| **aniTextAbove** | 4 | Positions image at top of control. |

## Gauge Control Constants

See Also

### Style Constants

| Constant | Value | Description |
|---|---|---|
| **gauHoriz** | 0 | Horizontal linear gauge with fill. |
| **gauVert** | 1 | Vertical gauge with fill. |
| **gauSemi** | 2 | Semicircular needle gauge. |
| **gauFull** | 3 | Full circle needle gauge. |

## Graph Control Constants

See Also

### AutoInc Constants

| Constant | Value | Description |
|---|---|---|
| **gphOff** | 0 | Automatic incrementing off. |
| **gphOn** | 1 | Automatic incrementing on. |

### DrawLineStyle and LegendStyle Constants

| Constant | Value | Description |
|---|---|---|
| **gphMonochrome** | 0 | Sets background white and all colors black. |
| **gphColor** | 1 | Uses specified colors. |

### GraphType Constants

| Constant | Value | Description |
|---|---|---|
| **gphNone** | 0 | No graph. |
| **gphPie2d** | 1 | Two-dimensional pie chart. |
| **gphPie3d** | 2 | Three-dimensional pie chart. |
| **gphBar2d** | 3 | Two-dimensional bar chart. |
| **gphBar3d** | 4 | Three-dimensional bar chart. |
| **gphGantt** | 5 | Gantt chart. |
| **gphLine** | 6 | Line graph. |
| **gphLogLin** | 7 | Log/Lin graph. |
| **gphArea** | 8 | Area graph. |
| **gphScatter** | 9 | Scatter graph. |
| **gphPolar** | 10 | Polar graph. |
| **gphHLC** | 11 | High-low-close graph. |

### BackgroundColor, ForegroundColor, and ColorData Constants

| Constant | Value | Description |
|---|---|---|
| **gphBlack** | 0 | Black. |
| **gphBlue** | 1 | Blue. |
| **gphGreen** | 2 | Green. |
| **gphCyan** | 3 | Cyan. |
| **gphRed** | 4 | Red. |
| **gphMagenta** | 5 | Magenta. |
| **gphBrown** | 6 | Brown. |
| **gphLightGray** | 7 | Light gray. |
| **gphDarkGray** | 8 | Dark gray. |
| **gphLightBlue** | 9 | Light blue. |
| **gphLightGreen** | 10 | Light green. |
| **gphLightCyan** | 11 | Light cyan. |
| **gphLightRed** | 12 | Light red. |
| **gphLightMagenta** | 13 | Light magenta. |
| **gphYellow** | 14 | Yellow. |
| **gphWhite** | 15 | White. |
| **gphAutoBW** | 16 | (Default) Automatic black and white. Only available in Foreground constants. |

### SymbolData Constants

| Constant | Value | Description |
|---|---|---|
| **gphCrossPlus** | 0 | Plus sign (+) symbol. |
| **gphCrossTimes** | 1 | Multiplication sign (x) symbol. |
| **gphTriangleUp** | 2 | Upright triangle symbol. |
| **gphSolidTriangle** | 3 | Solid triangle symbol. |
| **gphTriangleDown** | 4 | Upside-down triangle symbol. |
| **gphSolidTriangle** | 5 | Solid triangle symbol. |
| **gphSquare** | 6 | Square symbol. |
| **gphSolidSquare** | 7 | Solid square symbol. |
| **gphDiamond** | 8 | Diamond symbol. |
| **gphSolidDiamond** | 9 | Solid diamond symbol. |

### GridStyle Constants

| Constant | Value | Description |
|---|---|---|
| **gphNone** | 0 | No grid. |
| **gphHorizontal** | 1 | Horizontal grid. |
| **gphVertical** | 2 | Vertical grid. |
| **gphBoth** | 3 | Both grids. |

### DataReset Constants

| Constant | Value | Description |
|---|---|---|
| **gphNone** | 0 | No reset. |
| **gphGraphData** | 1 | Resets graph data. |
| **gphColorColorData** | 2 | Resets color data. |
| **gphExtraData** | 3 | Resets extra data. |
| **gphLabelText** | 4 | Resets label text. |
| **gphLegendText** | 5 | Resets legend text. |
| **gphPatternData** | 6 | Resets pattern data. |
| **gphSymbolData** | 7 | Resets symbol data. |
| **gphXPosData** | 8 | Resets x-position data. |
| **gphAllData** | 9 | Resets all data. |
| **gphFontInfo** | 10 | Resets font information. |

### DrawMode Constants

| Constant | Value | Description |
|---|---|---|
| **gphNoAction** | 0 | No graph drawn at design time. |
| **gphClear** | 1 | No graph drawn at design time, but properties displayed. |
| **gphDraw** | 2 | Displays graph at design time and run time. |
| **gphBlit** | 3 | Displays graph using blitting technique. |
| **gphCopy** | 4 | Copies graph to Clipboard. |
| **gphPrint** | 5 | Sends graph to printer. |
| **gphWrite** | 6 | Writes graph to disk. |

### FontStyle Constants

| Constant | Value | Description |
|---|---|---|
| **gphDefault** | 0 | Default. |
| **gphItalic** | 1 | Italic. |
| **gphBold** | 2 | Bold. |

| | | |
|---|---|---|
| **gphBoldItalic** | 3 | BoldItalic. |
| **gphUnderlined** | 4 | Underlined. |
| **gphUnderlinedItalic** | 5 | UnderlinedItalic. |
| **gphUnderlinedBold** | 6 | UnderlinedBold. |
| **gphUnderlinedBoldItalic** | 7 | UnderlinedBoldItalic. |

### FontFamily Constants

| Constant | Value | Description |
|---|---|---|
| **gphRoman** | 0 | Roman. |
| **gphSwiss** | 1 | Swiss. |
| **gphModern** | 2 | Modern. |

### FontUse Constants

| Constant | Value | Description |
|---|---|---|
| **gphGraphTitle** | 0 | GraphTitle. |
| **gphOtherTitles** | 1 | OtherTitles. |
| **gphLabels** | 2 | Labels. |
| **gphLegend** | 3 | Legend. |
| **gphAllText** | 4 | AllText. |

### IndexStyle Constants

| Constant | Value | Description |
|---|---|---|
| **gphStandard** | 0 | Standard. |
| **gphEnhanced** | 1 | Enhanced. |

### Labels Constants

| Constant | Value | Description |
|---|---|---|
| **gphOff** | 0 | Off. |
| **gphOn** | 1 | On. |
| **gphXAxisLabelsOnly** | 2 | X-axis labels only. |
| **gphYAxisLabelsOnly** | 3 | Y-axis labels only. |

### LineStats Constants

| Constant | Value | Description |
|---|---|---|
| **gphNone** | 0 | None. |
| **gphMean** | 1 | Mean. |
| **gphMinmax** | 2 | MinMax. |
| **gphMeanMinmax** | 3 | Mean and MinMax. |
| **gphStddev** | 4 | Stddev. |
| **gphStddevMean** | 5 | StdDev and Mean. |
| **gphStddevMinmax** | 6 | StdDev and MinMax. |
| **gphStddevMinmaxMean** | 7 | StdDev and MinMax and Mean. |
| **gphBestfit** | 8 | BestFit. |
| **gphBestfitMean** | 9 | BestFit and Mean. |
| **gphBestfitMinmax** | 10 | BestFit and MinMax. |
| **gphBestfitMinmaxMean** | 11 | BestFit and MinMax and Mean. |
| **gphBestfitStddev** | 12 | BestFit and StdDev. |
| **gphBestfitStddevMean** | 13 | BestFit and StdDev and Mean. |
| **gphBestfitStddevMinmax** | 14 | BestFit and StdDev and MinMax. |
| **gphAll** | 15 | All. |

**Palette Constants**

| Constant | Value | Description |
|---|---|---|
| **gphDefault** | 0 | Default. |
| **gphPastel** | 1 | Pastel. |
| **gphGrayscale** | 2 | Grayscale. |

**PatternedLines Constants**

| Constant | Value | Description |
|---|---|---|
| **gphPatternOff** | 0 | Pattern off. |
| **gphPatternOn** | 1 | Pattern on. |

**PrintStyle Constants**

| Constant | Value | Description |
|---|---|---|
| **gphMonochrome** | 0 | Color. |
| **gphColor** | 1 | Color with border. |
| **gphMonochromeWithBorder** | 2 | Monochrome. |
| **gphColorWithBorder** | 3 | Monochrome with border. |

**RandomData Constants**

| Constant | Value | Description |
|---|---|---|
| **gphOff** | 0 | Off. |
| **gphOn** | 1 | On. |

**ThickLines Constants**

| Constant | Value | Description |
|---|---|---|
| **gphLinesOff** | 0 | Lines off. |
| **gphLinesOn** | 1 | Lines on. |

**YAxisPos Constants**

| Constant | Value | Description |
|---|---|---|
| **gphDefault** | 0 | Default. |
| **gphAlignLeft** | 1 | Align left. |
| **gphAlignRight** | 2 | Align right. |

**YAxisStyle Constants**

| Constant | Value | Description |
|---|---|---|
| **gphDefault** | 0 | Default. |
| **gphVariableOrigin** | 1 | Variable origin. |
| **gphUserDefined** | 2 | User-defined. |

**Ticks Constants**

| Constant | Value | Description |
|---|---|---|
| **gphTicksOff** | 0 | Ticks off. |
| **gphTicksOn** | 1 | Ticks on. |
| **gphXAxisTicksOnly** | 2 | X-axis ticks only. |
| **gphYAxisTicksOnly** | 3 | Y-axis ticks only. |

## Key State Control Constants

See Also

### Style Constants

| Constant | Value | Description |
|---|---|---|
| **keyCapsLock** | 0 | CAPS LOCK key. |
| **keyNumLock** | 1 | NUM LOCK key. |
| **keyInsert** | 2 | INSERT key. |
| **keyScrollLock** | 3 | SCROLL LOCK key. |

## Multimedia MCI Control Constants

See Also

### Mode Constants

| Constant | Value | Description |
|---|---|---|
| **mciModeOpen** | 524 | Device not open. |
| **mciModeStop** | 525 | Device stop. |
| **mciModePlay** | 526 | Device play. |
| **mciModeRecord** | 527 | Device record. |
| **mciModeSeek** | 528 | Device seek. |
| **mciModePause** | 529 | Device pause. |
| **mciModeReady** | 530 | Device ready. |

### Notify Constants

| Constant | Value | Description |
|---|---|---|
| **mciNotifySuccessful** | 1 | Command completed successfully. |
| **mciNotifySuperseded** | 2 | Command superseded by another command. |
| **mciAborted** | 4 | Command aborted by user. |
| **mciFailure** | 8 | Command failed. |

### Orientation Constants

| Constant | Value | Description |
|---|---|---|
| **mciOrientHorz** | 0 | Buttons arranged horizontally. |
| **mciOrientVert** | 1 | Buttons arranged vertically. |

### RecordMode Constants

| Constant | Value | Description |
|---|---|---|
| **mciRecordInsert** | 0 | Insert recording mode. |
| **mciRecordOverwrite** | 1 | Overwrite recording mode. |

### Format Constants

| Constant | Value | Description |
|---|---|---|
| **mciFormatMilliseconds** | 0 | Milliseconds format. |
| **mciFormatHms** | 1 | Hours, seconds, and minutes format. |
| **mciFormatMsf** | 2 | Minutes, seconds, and frames format. |
| **mciFormatFrames** | 3 | Frames format. |
| **mciFormatSmpte24** | 4 | 24-frame SMPTE format. |
| **mciFormatSmpte25** | 5 | 25-frame SMPTE format. |
| **mciFormatSmpte30** | 6 | 30-frame SMPTE format. |
| **mciFormatSmpte30Drop** | 7 | 30-drop-frame SMPTE format. |
| **mciFormatBytes** | 8 | Bytes format. |
| **mciFormatSamples** | 9 | Samples format. |
| **mciFormatTmsf** | 10 | Tracks minutes, seconds, and frames format. |

## Spin Button Control Constants

See Also

### Orientation Constants

| Constant | Value | Description |
|---|---|---|
| **spnVertical** | 0 | Up and down spin arrows. |
| **spnHorizontal** | 1 | Left and right spin arrows. |

## Masked Edit Control Constants

See Also

### ClipMode Constants

| Constant | Value | Description |
|---|---|---|
| **mskIncludeLiterals** | 0 | Include literals on cut or copy. |
| **mskExcludeLiterals** | 1 | Exclude literals on cut or copy. |

## Communications Control Constants

See Also

### Handshake Constants

| Constant | Value | Description |
|---|---|---|
| **comNone** | 0 | No handshaking. |
| **comXonXoff** | 1 | XOn/XOff handshaking. |
| **comRTS** | 2 | Request-to-send/clear-to-send handshaking. |
| **comRTSXOnXOff** | 3 | Both request-to-send and XOn/XOff handshaking. |

### OnComm Constants

| Constant | Value | Description |
|---|---|---|
| **comEvSend** | 1 | Send event. |
| **comEvReceive** | 2 | Receive event. |
| **comEvCTS** | 3 | Change in clear-to-send line. |
| **comEvDSR** | 4 | Change in data-set ready line. |
| **comEvCD** | 5 | Change in carrier detect line. |
| **comEvRing** | 6 | Ring detect. |
| **comEvEOF** | 7 | End of file. |

### Error Constants

| Constant | Value | Description |
|---|---|---|
| **comBreak** | 1001 | Break signal received. |
| **comCTSTO** | 1002 | Clear-to-send timeout. |
| **comDSRTO** | 1003 | Data-set ready timeout. |
| **comFrame** | 1004 | Framing error. |
| **comOverrun** | 1006 | Port overrun. |
| **comCDTO** | 1007 | Carrier detect timeout. |
| **comRxOver** | 1008 | Receive buffer overflow. |
| **comRxParity** | 1009 | Parity error. |
| **comTxFull** | 1010 | Transmit buffer full. |

## MAPI Control Constants

See Also

### SessonAction Constants

| Constant | Value | Description |
|---|---|---|
| **mapSignOn** | 1 | Log user into account. |
| **mapSignOff** | 2 | End messaging session. |

### Delete Constants

| Constant | Value | Description |
|---|---|---|
| **mapMessageDelete** | 10 | Delete current message. |
| **mapRecipientDelete** | 14 | Delete the currently indexed recipient. |
| **mapAttachmentDelete** | 15 | Delete the currently indexed attachment. |

### MAPIErrors

| Constant | Value | Description |
|---|---|---|
| **mapSuccessSuccess** | 32000 | Action returned successfully. |
| **mapUserAbort** | 32001 | User cancelled process. |
| **mapFailure** | 32002 | Unspecified failure. |
| **mapLoginFail** | 32003 | Login failure. |
| **mapDiskFull** | 32004 | Disk full. |
| **mapInsufficientMem** | 32005 | Insufficient memory. |
| **mapAccessDenied** | 32006 | Access denied. |
| **mapGeneralFailure** | 32007 | General failure. |
| **mapTooManySessions** | 32008 | Too many sessions. |
| **mapTooManyFiles** | 32009 | Too many files. |
| **mapTooManyRecipients** | 32010 | Too many recipients. |
| **mapAttachmentNotFound** | 32011 | Attachment not found. |
| **mapAttachmentOpenFailure** | 32012 | Attachment open failure. |
| **mapAttachmentWriteFailure** | 32013 | Attachment write failure. |
| **mapUnknownRecipient** | 32014 | Unknown recipient. |
| **mapBadRecipType** | 32015 | Invalid recipient type. |
| **mapNoMessages** | 32016 | No message. |
| **mapInvalidMessage** | 32017 | Invalid message. |
| **mapTextTooLarge** | 32018 | Text too large. |
| **mapInvalidSession** | 32019 | Invalid session. |
| **mapTypeNotSupported** | 32020 | Type not supported. |
| **mapAmbiguousRecipient** | 32021 | Ambiguous recipient. |
| **mapMessageInUse** | 32022 | Message in use. |
| **mapNetworkFailure** | 32023 | Network failure. |
| **mapInvalidEditFields** | 32024 | Invalid editfields. |
| **mapInvalidRecips** | 32025 | Invalid Recipients. |
| **mapNotSupported** | 32026 | Current action not supported. |
| **mapUserAbout** | 32027 | User aborted previous action. |
| **mapSessionExist** | 32050 | Session ID already exists. |
| **mapInvalidBuffer** | 32051 | Read-only in read buffer. |
| **mapInvalidReadBufferAction** | 32052 | Valid in compose buffer only. |
| **mapNoSession** | 32053 | No valid session ID. |
| **mapInvalidRecipient** | 32054 | Originator information not available. |

| | | |
|---|---|---|
| **mapInvalidComposeBufferAction** | 32055 | Action not valid for Compose Buffer. |
| **mapControlFailure** | 32056 | No messages in list. |
| **mapNoRecipients** | 32057 | No recipients. |
| **mapNoAttachment** | 32058 | No attachments. |

**RecipType Constants**

| Constant | Value | Description |
|---|---|---|
| **mapOrigList** | 0 | Message originator. |
| **mapToList** | 1 | Recipient is a primary recipient. |
| **mapCcList** | 2 | Recipient is a copy recipient. |
| **mapBccList** | 3 | Recipient is a blind copy recipient. |

**AttachType Constants**

| Constant | Value | Description |
|---|---|---|
| **mapData** | 0 | Attachment is a data file. |
| **mapEOLE** | 1 | Attachment is an embedded OLE object. |
| **mapSOLE** | 2 | Attachment is a static OLE object. |

**See Also**

Error Messages, **MAPI** Controls

Visual Basic Custom Control Constants

## Outline Control Constants

See Also

### PictureType Constants

| Constant | Value | Description |
|---|---|---|
| **outClosed** | 0 | PictureClosed picture. |
| **outOpen** | 1 | PictureOpen picture. |
| **outLeaf** | 2 | PictureLeaf picture. |

### Error Constants

| Constant | Value | Description |
|---|---|---|
| **outBadPicFormat** | 32000 | Picture format not supported. |
| **outBadIndentation** | 32001 | Bad outline indentation. |
| **outOutOfMemory** | 32002 | Out of memory. |
| **outParentNotExpanded** | 32003 | Parent not expanded. |

### Style Constants

| Constant | Value | Description |
|---|---|---|
| **outTextOnly** | 0 | Picture and text. |
| **outPictureText** | 1 | Plus/Minus and text. |
| **outPlusMinusText** | 2 | Plus/Minus, picture, and text. |
| **outPlusPictureText** | 3 | Text only. |
| **outTreelinesText** | 4 | Treelines, picture, and text. |
| **outTreelinesPictureText** | 5 | Treelines and text. |

**See Also**

Visual Basic Custom Control Constants

## Visual Basic Custom Control Constants

The following constants are specified by Visual Basic. As a result, they can be used anywhere in your code in place of the actual values.

- 3D Controls Constants
- Animated Button Control Constants
- Communications Control Constants
- Gauge Control Constants
- Graph Control Constants
- ImageList Control Constants
- Key State Control Constants
- ListView Control Constants
- MAPI Control Constants
- Masked Edit Control Constants
- Multimedia MCI Control Constants
- RichTextBox Control Constants
- Slider Control Constants
- Spin Button Control Constants
- SSTab Control Constants
- StatusBar Control Constants
- TabStrip Control Constants
- Toolbar Control Constants
- TreeView Control Constants
- Windows 95 Control Constants

You can search for data access, Visual Basic, and Visual Basic for application constants for more constant lists.

Use the Object Browser to browse the list of built-in constants. From the View menu, choose Object Browser, select the appropriate library, and then select the constants you want to see. Scroll the list in the Methods/Properties box to see the complete list of constants.

## Error Messages, Animated Button Control

The following table lists the trappable errors for the Animated Button control.

| Error Number | Message Explanation |
|---|---|
| 30000 | Error loading picture |
| 30001 | Invalid hot spot |
| 30002 | Invalid CCB file |
| 30003 | Current frame is not a metafile |
| 30004 | Current frame is not an icon |

## Error Messages, Communications Control

The following table lists the trappable errors for the Communications control.

| Error Number | Message Explanation |
|---|---|
| 8000 | Operation not valid while the port is opened |
| 8001 | Timeout value must be greater than zero |
| 8002 | Invalid Port Number |
| 8003 | Property available only at run time |
| 8004 | Property is read only at runtime |
| 8005 | Port already open |
| 8006 | The device identifier is invalid or unsupported |
| 8007 | The device's baud rate is unsupported |
| 8008 | The specified byte size is invalid |
| 8009 | The default parameters are in error |
| 8010 | The hardware is not available (locked by another device) |
| 8011 | The function cannot allocate the queues |
| 8012 | The device is not open |
| 8013 | The device is already open |
| 8014 | Could not enable comm notification |
| 8015 | Could not set comm state |
| 8016 | Could not set comm event mask |
| 8018 | Operation valid only when the port is open |
| 8019 | Device busy |
| 8020 | Error reading comm device |

## Error Messages, Graph Control

The following table lists the trappable errors for the Graph control.

| Error Number | Message Explanation |
|---|---|
| 32000 | String value too long |
| 32001 | Subscript out of range |
| 32002 | Error creating graph image |

## Error Messages, MAPI Controls

The following table lists the trappable errors for the MAPI controls.

| Error Number | Message<br>Explanation |
|---|---|
| 32001 | User cancelled process<br>The current action was not completed because the user cancelled the process. |
| 32002 | Unspecified failure has occurred<br>An unspecified error occurred during the current action. For example, the action was unable to delete or address mail correctly. |
| 32003 | Login has failed<br>There was no default logon, and the user failed to log on correctly. |
| 32004 | Disk is full<br>The disk is full. The current action could not create a disk file. |
| 32005 | Insufficient memory<br>There is insufficient memory to proceed with the current action. |
| 32006 | Access denied |
| 32007 | General Failure<br>This is an unspecified error. |
| 32008 | Too many sessions<br>The user has too many sessions open at once. |
| 32009 | Too many files<br>Too many file attachments are contained in the message. The mail wasn't sent or read. |
| 32010 | Too many recipients<br>There are too many message recipients specified. Mail wasn't sent or read. |
| 32011 | Attachment not found<br>The specified attachment wasn't found, and mail wasn't sent. |
| 32012 | Failure on opening attachment<br>The attachment couldn't be located. Mail wasn't sent. Verify that the **AttachmentPathName** property is valid. |
| 32013 | Failure attempting to write an attachment<br>An attachment could not be written to a temporary file. Check directory permissions. |
| 32014 | Unknown recipient<br>The recipient doesn't appear in the address list. Mail wasn't sent. |
| 32015 | Invalid recipient type<br>The type of recipient was incorrect. Valid type values are 1 (primary recipient), 2 (copy recipient), and 3 (blind copy recipient). |
| 32016 | No messages<br>Unable to find the next message. |
| 32017 | Invalid message<br>An invalid message ID was used. The current action was not completed. |
| 32018 | Text is too large<br>The text in the message was too large to send. The mail wasn't sent. Text is limited to 32K. |
| 32019 | Invalid session<br>An invalid session ID was used. To associate the MAPI messages control with a valid messaging session, set the **SessionID** property to the MAPI session control's SessionID. |
| 32020 | Type not supported |
| 32021 | Ambiguous recipient |

| | |
|---|---|
| | One or more recipient addresses are invalid.   Make sure the addresses for the **RecipAddress** property are valid. |
| 32022 | Message in use |
| 32023 | Network failure |
| 32024 | Invalid editfields |
| | The value of the **AddressEditFieldCount** property is invalid.   Valid values are from 0 to 4. |
| 32025 | Invalid Recipients |
| | One or more recipient addresses are invalid.   Make sure the addresses for the **RecipAddress** property are valid. |
| 32026 | Not supported |
| | The current action is not supported by the underlying mail system. |
| 32027 | The user has aborted the previous action |
| 32050 | Logon failure: valid session ID already exists |
| | The MAPI messages control is already using a valid session ID. |
| 32051 | Property is read only when not using Compose Buffer. Set MsgIndex = -1 |
| 32052 | Action only valid for Compose Buffer. Set MsgIndex = -1 |
| 32053 | MAPI Failure: valid session ID does not exist |
| | The MAPI messages control does not have a valid session handle from the MAPI session control. |
| 32054 | No originator in the Compose Buffer |
| | You cannot see message originator information while in the Compose Buffer (**MsgIndex** set to 1). |
| 32055 | Action not valid for Compose Buffer |
| | The attempted action is not valid in the Compose Buffer (**MsgIndex** set to 1). |
| 32056 | Cannot perform action, no messages in list |
| 32057 | Cannot perform action, no recipients |
| 32058 | Cannot perform action, no attachments |

## Error Messages, Multimedia MCI Control

The following table lists the trappable errors for the Multimedia MCI control.

| Error Number | Message Explanation |
|---|---|
| 30001 | Can't create button |
| 30002 | Can't create a timer resource |
| 30003 | Can't create string. Either string too long or out of memory |

## Error Messages, Outline Control

The following table lists the trappable errors for the Outline control.

| Error Number | Message Explanation |
|---|---|
| 32000 | Outline: Picture format not supported |
| 32001 | Bad outline indentation |
| 32002 | Outline: Out of memory |
| 32003 | Outline: Parent not expanded |
| 32004 | Outline: Unknown error |

## Error Messages, Picture Clip Control

The following table lists the trappable errors for the Picture Clip control.

| Error Number | Message Explanation |
|---|---|
| 32000 | Picture format not supported |
| | You can only load bitmap (.BMP) files into the picture clip control. |
| 32001 | Unable to obtain display context |
| 32002 | Unable to obtain memory device context |
| 32003 | Unable to obtain bitmap |
| 32004 | Unable to select bitmap object |
| 32005 | Unable to allocate internal picture structure |
| 32006 | Bad GraphicCell Index |
| | The *index* argument for the **GraphicCell** property is out of range. This argument must be in the range 0 to (**PicClip**.**Rows** * **PicClip**.**Cols**)■1. |
| 32007 | No GraphicCell picture size specified |
| 32008 | Only bitmap GraphicCell pictures allowed |
| 32010 | Bad GraphicCell picture clip property request |
| 32012 | GetObject() Windows function failure |
| | A call to the Windows function **GetObject ()** failed. |
| 32014 | GlobalAlloc() Windows function failure |
| | A call to the Windows function **GlobalAlloc ()** failed. |
| 32015 | Clip region boundary error |
| | The **ClipHeight** and **ClipWidth** properties specify coordinates which are outside the boundary of the bitmap loaded in the Picture Clip control. |
| 32016 | Cell size too small (must be at least 1 by 1 pixel) |
| 32017 | Rows property must be greater than zero |
| 32018 | Cols property must be greater than zero |
| 32019 | StretchX property cannot be negative |
| 32020 | StretchY property cannot be negative |
| 32021 | No picture assigned |

## Error Messages, Spin Button Control

The following table lists the trappable errors for the Spin Button control.

| Error Number | Message Explanation |
|---|---|
| 30000 | Negative value invalid for this property<br>The **Delay**, **BorderThickness**, **ShadowThickness**, and **TdThickness** properties cannot be set to a negative value. |

## Error Messages, 3D Controls

The following table lists the trappable errors for these 3D controls: Command Button, Group Push Button, and Panel.   The remaining 3D controls▪Check Box**,** Frame, and Option Button ▪have no trappable errors.

### 3D Command Button Control

| Error Number | Message / Explanation |
|---|---|
| 30000 | Only Picture formats '.BMP' & '.ICO' supported |
| | An unsupported graphic type is assigned to the **Picture** property of the command button. Only bitmap and icon formats are supported. |
| 30004 | Bevel width must be from 0 to 10 |
| | The bevel width is set to an invalid value. |

### 3D Group Push Button Control

| Error Number | Message / Explanation |
|---|---|
| 30001 | Only Picture format '.BMP' supported |
| | An unsupported graphic type is assigned to the **Picture** property of the 3D group push button.   Only the bitmap format is supported. |
| 30005 | Group number must be from 0 to 99 |
| | The **GroupNumber** property is set to an invalid value. |
| 30007 | Bevel width must be from 0 to 2 |
| | The bevel width is set to an invalid value. |

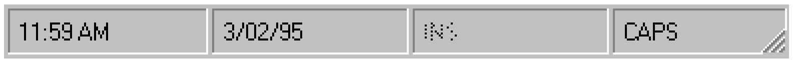
### 3D Panel Control

| Error Number | Message / Explanation |
|---|---|
| 30002 | Bevel width must be from 0 to 30 |
| | The **BevelWidth** property is set to an invalid value. |
| 30003 | Border width must be from 0 to 30 |
| | The **BorderWidth** property is set to an invalid value. |
| 30006 | Flood percent must be from 0 to 100 |
| | The **FloodPercent** property is set to an invalid value. |

## StatusBar Control

See Also Properties Methods Events Constants

A **StatusBar** control provides a window, usually at the bottom of a parent form, through which an application can display various kinds of status data. The **StatusBar** can be divided up into a maximum of sixteen **Panel** objects that are contained in a **Panels** collection.



### Syntax

**StatusBar**

### Remarks

A **StatusBar** control consists of **Panel** objects, each of which can contain text and/or a picture. Properties to control the appearance of individual panels include **Width**, **Alignment** (of text and pictures), and **Bevel**. Additionally, you can use one of seven values of the **Style** property to automatically display common data such as date, time, and keyboard states.

At design time, you can create panels, customize their appearances, and set their functions using the Panel Properties dialog box. At run time, the **Panel** objects can be reconfigured to reflect different functions, depending on the state of the application. For detailed information about the properties, events, and methods of **Panel** objects, see the **Panel** Object, **Panels** Collection topic.

A **StatusBar** control typically displays information about an object being viewed on the form, the object's components, or contextual information that relates to that object's operation. The **StatusBar**, along with other controls such as the **Toolbar** control, gives you the tools to create an interface that is economical and yet rich in information.

---

**Distribution Note** The **StatusBar** control is a 32-bit custom control that can only run on 32-bit systems such as Windows 95 and Windows NT version 3.51 or higher. Additionally, the **StatusBar** control is part of a group of custom controls that are found in the COMCTL32.OCX file. To use the **StatusBar** control in your application, you must add the COMCTL32.OCX file to the project. When distributing your application, install the COMCTL32.OCX file in the user's Microsoft Windows SYSTEM directory. For more information on how to add a custom control to a project, see the *Programmer's Guide*.

---

**See Also**

**Panel** Object, **Panels** Collection

## StatusBar Control Properties

- **Align** Property
- **Container** Property
- **DragIcon** Property
- **DragMode** Property
- **Enabled** Property
- **Font** Property
- **Height** Property
- **hWnd Property**
- **Index** Property
- **Left** Property
- **MouseIcon** Property
- **MousePointer** Property
- **Name** Property
- **Negotiate** Property
- **Panels** Property
- **Parent** Property
- **SimpleText** Property
- **Style** Property (**StatusBar** Control)
- **Tag** Property
- **Top** Property
- **Visible** Property
- **WhatsThisHelpID** Property
- **Width** Property

Close

## StatusBar Control Methods

**Move** Method
**Refresh** Method
**ShowWhatsThis** Method
**ZOrder** Method

Close

**StatusBar Control Events**

## StatusBar Control Constants

See Also

### Sbar Style Constants

| Constant | Value | Description |
|---|---|---|
| **sbrNormal** | 0 | Normal. **StatusBar** is divided into panels. |
| **sbrSimple** | 1 | Simple. **StatusBar** has only one large panel and SimpleText. |

### PanelAlignment Constants

| Constant | Value | Description |
|---|---|---|
| **sbrLeft** | 0 | Text to left. |
| **sbrCenter** | 1 | Text centered. |
| **sbrRight** | 2 | Text to right. |

### PanelAutoSize Constants

| Constant | Value | Description |
|---|---|---|
| **sbrNoAutoSize** | 0 | No Autosizing. |
| **sbrSpring** | 1 | Extra space divided among panels. |
| **sbrContents** | 2 | Fit to contents. |

### PanelBevel Constants

| Constant | Value | Description |
|---|---|---|
| **sbrNoBevel** | 0 | No bevel. |
| **sbrInset** | 1 | Bevel inset. |
| **sbrRaised** | 2 | Bevel raised. |

### PanelStyle Constants

| Constant | Value | Description |
|---|---|---|
| **sbrText** | 0 | Text and/or bitmap displayed. |
| **sbrCaps** | 1 | Caps Lock status displayed. |
| **sbrNum** | 2 | Number Lock status displayed. |
| **sbrIns** | 3 | Insert key status displayed. |
| **sbrScrl** | 4 | Scroll Lock status displayed. |
| **sbrTime** | 5 | Time displayed in System format. |
| **sbrDate** | 6 | Date displayed in System format. |

**See Also**

**Alignment** Property (**Panel** Object)
**Autosize** Property (**Panel** Object)
**Bevel** Property (**Panel** Object)
**SimpleText** Property
**Style** Property (**StatusBar** Control)
**Style** Property (**Panel** Object)
Visual Basic Custom Control Constants
Windows 95 Controls Constants

## Panel Object, Panels Collection

See Also Properties Methods Events

- A **Panel** object can contain text and a bitmap that can be used to reflect the status of an application.
- A **Panels** collection contains a collection of **Panel** objects.

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*statusbar.***Panels**

*statusbar.***Panels(***index***)**

The syntax lines above refer to the collection and to individual elements in the collection, respectively, according to standard collection syntax.

The **Panel** object, **Panels** collection syntax has these parts:

| Part | Description |
|---|---|
| *statusbar* | An object expression that evaluates to a **StatusBar** control. |
| *index* | An integer or string that uniquely identifies the object in the collection. The integer is the value of the **Index** property; the string is the value of the **Key** property. |

### Remarks

The **Panels** collection is a 1-based array of **Panel** objects. By default, there is one **Panel** object on a **StatusBar** control. Therefore, if you want three panels to be created, you only need toad two objects to the default collection.

The **Panels** property returns a reference to a **Panels** collection.

To add a **Panel** object to a collection, use the **Add** method for **Panel** objects at run time, or the Panel Properties tab on the Status Bar Control Properties dialog box at design time.

Each item in the collection can be accessed by its **Index** property or its **Key** property. For example, to get a reference to the third **Panel** object in a collection, use the following syntax:

```
Dim pnlX As Panel
Set pnlX = StatusBar1.Panels(3)          ' Reference by index number.
   ' or
Set pnlX = StatusBar1.Panels.("Third")   ' Reference by unique key.
   ' or
Set pnlX = StatusBar1.Panels.Item(3)     ' Use Item method.
```

**See Also**

**Add** Method (**Panels** Collection)
**Index** Property
**Key** Property
**Panels** Property
**StatusBar** Control

Close

**Panel Object, Panels Collection Properties**

Legend

**Alignment** Property (**Panel** Object)

**Autosize** Property (**Panel** Object)■
**Bevel** Property(**Panel** Object)■
**Count** Property □
**Enabled** Property■
**Index** Property■
**Left** Property■
**MinWidth** Property■
**Picture** Property■
**Style** Property (**Panel** Object)■
**Tag** Property■
**Text** Property■

**Visible** Property■
**Width** Property (**Panel** Object)■

Close

**Panel Object, Panels Collection Methods**

Legend

**Add** Method (**Panels** Collection)■

**Clear** Method■
**Item** Method■
**Remove** Method■

Close

**Panel Object, Panels Collection Events**

Legend

PanelClick Event■

PanelDblClick Event■

## Add Method (Panels Collection)

See Also Example

Adds a **Panel** object to a **Panels** collection and returns a reference to the newly created **Panel** object. Doesn't support named arguments.

---

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

---

### Syntax

*object*.**Add***(index, key, text, style, picture)*

The **Add** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Panels** collection. |
| *index* | Optional. An integer specifying the position where the **Panel** object is to be inserted. If no *index* is specified, the **Panel** is added to the end of the **Panels** collection. |
| *key* | Optional. A unique string that identifies the **Panel**. Use the *key* to retrieve a specific **Panel**. |
| *text* | Optional. A string that appears in the **Panel**. |
| *style* | Optional. The style of the panel. The available styles are detailed in the **Style** Property (**Panel** Object). |
| *picture* | Optional. Specifies the bitmap displayed in the active **Panel**. For more information, see the **LoadPicture** function. |

### Remarks

At run time, the **Add** method returns a reference to the newly inserted **Panel** object. With this reference, you can set properties for every new **Panel** in the following manner:

```
Dim pnlX as Panel
Dim I as Integer
For I = 1 to 6                     ' Add six Panel objects.
' Create a panel and get a reference to it simultaneously.
Set pnlX = StatusBar1.Panels.Add(,"Panel " & I) ' Set Key property.
pnlX.Style = I                     ' Set Style property.
pnlX.AutoSize = sbrContents        ' Set AutoSize property.
Next I
```

If you set the **Style** property for a **Panel** object to any value other than 0 (text and picture), any text you set for the **Text** property will not appear unless you reset the **Style** property to 0.

The **Panels** collection is a 1-based collection. In order to get a reference to the first (default) **Panel** in a collection, use the **Item** method:

```
Dim pnlX As Panel
' Get a reference to first Panel.
Set pnlX = StatusBar1.Panels.Item(1)
pnlX.Text = "Changed text"         ' Alter the Panel object's text.
```

By default, one **Panel** already exists on the control. Therefore, after adding panels to a collection, the **Count** will be one more than the number of panels added. For example:

```
Dim I as Integer
For I = 1 to 4  ' Add four panels.
   StatusBar1.Panels.Add   ' Add panels without any properties.
Next I
MsgBox StatusBar1.Panels.Count   ' Returns 5 panels.
```

**See Also**

**Alignment** Property (**Panel** Object)
**AutoSize** Property (**Panel** Object)
**Bevel** Property (**Panel** Object)
**Count** Property
**Index** Property
**Item** Method
**Key** Property
**Panel** Object, **Panels** Collection
**StatusBar** Control
**Style** Property (**Panel** Object)
**Visible** Property

■

## Add Method (Panels Collection) Example

This example uses the **Add** method to add two new **Panel** objects to a **StatusBar** control.   To try the example, place a **StatusBar** control on a form and paste the code into the form's Declarations section. Run the example.

```
Private Sub Form_Load()
Dim pnlX as Panel
   ' Add a panel with a clock icon and time style.
   Set pnlX = StatusBar1.Panels.Add _
   (,,,sbrTime,LoadPicture("icons\misc\clock03.ico"))
   ' Add second panel, with bitmap and Date style.
   Set pnlX = StatusBar1.Panels.Add _
   (,,,sbrDate,LoadPicture("bitmaps\assorted\calendar.bmp"))
   ' Set Bevel property for last Panel object.
   pnlX.Bevel = sbrInset           ' Inset bevel.
   pnlX.Alignment = sbrRight       ' Set Alignment property for last object.
   ' Set Text and AutoSize properties for first (default )Panel object.
   StatusBar1.Panels(1).Text = "Add Panel Example"
   StatusBar1.Panels(1).AutoSize = sbrContents
End Sub
```

## Alignment Property (Panel Object)

See Also Example

Returns or sets the alignment of text in the caption of a **Panel** object in a **StatusBar** control.

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object.***Alignment** [= *number*]

The **Alignment** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Panel** object. |
| *number* | A constant or value specifying the type of action, as described in Settings. |

### Settings

The settings for *number* are:

| Constant | Value | Description |
|---|---|---|
| **sbrLeft** | 0 | (Default). Text appears left-justified and to right of bitmap. |
| **sbrCenter** | 1 | Text appears centered and to right of bitmap. |
| **sbrRight** | 2 | Text appears right-justified and to left of bitmap. |

**See Also**

**Add** Method (**Panels** Collection)
**Clear** Method
**Item** Method
**Panel** Object, **Panels** Collection
**Remove** Method
**StatusBar** Control
**StatusBar** Control Constants

■

## Alignment Property (Panel Object) Example

This example adds two **Panel** objects to a **StatusBar** control and aligns the text in each panel using one of the three available styles. To try the example, place a **StatusBar** control on a form and paste the code into the Declarations section of the form. Run the example.

```
Sub Form_Load()
   Me.ScaleMode = vbTwips              ' Set ScaleMode to twips.
   Me.Width = 8145                     ' Make sure form is wide enough to see all
panels.
   ' Declare variables.
   Dim pnlX As Panel
   Dim I As Integer

   For I = 1 to 2                      ' Add two panels.
      StatusBar1.Panels.Add
   Next I

   For I = 1 to 3                      ' Add pictures to each Panel.
      Set pnlX = StatusBar1.Panels(I)
      Set pnlX.Picture = LoadPicture("icons\comm\net12.ico")
   Next I

   ' Set styles and alignment.
   With StatusBar1.Panels
   .Item(1).Text = "Left"
   .Item(1).Alignment = sbrLeft        ' Left alignment.
   .Item(1).MinWidth = 2500            ' Allow space to see effect.
   .Item(2). Text = "Center"
   .Item(2).Alignment = sbrCenter      ' Centered alignment.
   .Item(2).MinWidth = 2500            ' Allow space to see effect.
   .Item(3).Text = "Right"
   .Item(3).Alignment = sbrRight       ' Right alignment.
   .Item(3).MinWidth = 2500            ' Allow space to see effect.
   End With
End Sub
```

## AutoSize Property (Panel Object)

See Also Example

Returns or sets a value that allows the width of a **StatusBar** control's **Panel** object to be automatically sized when the panel's contents change or the parent form resizes.

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object.***AutoSize** [= *number*]

The **AutoSize** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Panel** object. |
| *number* | A constant or value specifying the type of action, as described in Settings. |

### Settings

The settings for *number* are:

| Constant | Value | Description |
|---|---|---|
| **sbrNoAutoSize** | 0 | (Default). None. No autosizing occurs. The width of the **Panel** is always and exactly that specified by the **Width** property. |
| **sbrSpring** | 1 | Spring. When the parent form resizes and there is extra space available, all panels with this setting divide the space and grow accordingly. However, the panels' width never falls below that specified by the **MinWidth** property. |
| **sbrContents** | 2 | Content. The **Panel** is resized to fit its contents. |

### Remarks

**Panel** objects with the Content style have precedence over those with the Spring style. This means that a Spring-style **Panel** is shortened if a **Panel** with the Contents style requires that space.

**See Also**

**MinWidth** Property
**Panel** Object, **Panels** Collection
**StatusBar** Control
**StatusBar** Control Constants
**Width** Property (**Panel** Object)

■

## AutoSize Property (Panel Object) Example

This example adds two **Panel** objects to a **StatusBar** control and sets the **AutoSize** property to Content for all panels. As the cursor is moved over the objects on the form, the x and y coordinates are displayed as well as the **Tag** property value for each control. To try the example, place a **StatusBar**, a **PictureBox**, and a **CommandButton** on a form, then paste the code into the Declarations section. Run the example and move the cursor over the various controls.

```
Private Sub Form_Load()
   Dim pnlX As Panel
   ' Set long tags for each object.
   Form1.Tag = "Project 1 Form"
   Command1.Tag = "A command button"
   Picture1.Tag = "Picture Box Caption"
   StatusBar1.Tag = "Application StatusBar1"
   ' Set the AutoSize style of the first panel to Contents.
   StatusBar1.Panels(1).AutoSize = sbrContents
   ' Add 2 more panels, and set them to Contents.
   Set pnlX = StatusBar1.Panels.Add
   pnlX.AutoSize = sbrContents
   Set pnlX = StatusBar1.Panels.Add
   pnlX.AutoSize = sbrContents
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, x As Single,
y As Single)
   ' Display the control's tag in panel 1, and x and y
   ' coordinates in panels 2 and 3. Because AutoSize = Contents,
   ' the first panel stretches to accommodate the varying text.
   StatusBar1.Panels(1).Text = Form1.Tag
   StatusBar1.Panels(2).Text = "X = " & x
   StatusBar1.Panels(3).Text = "Y = " & y
End Sub

Private Sub Command1_MouseMove(Button As Integer, Shift As Integer, x As
Single, y As Single)
   StatusBar1.Panels(1).Text = Command1.Tag
   StatusBar1.Panels(2).Text = "X = " & x
   StatusBar1.Panels(3).Text = "Y = " & y
End Sub

Private Sub Picture1_MouseMove(Button As Integer, Shift As Integer, x As
Single, y As Single)
   StatusBar1.Panels(1).Text = Picture1.Tag
   StatusBar1.Panels(2).Text = "X = " & x
   StatusBar1.Panels(3).Text = "Y = " & y
End Sub

Private Sub StatusBar1_MouseMove(Button As Integer, Shift As Integer, x As
Single, y As Single)
   StatusBar1.Panels(1).Text = StatusBar1.Tag
   StatusBar1.Panels(2).Text = "X = " & x
   StatusBar1.Panels(3).Text = "Y = " & y
End Sub
```

## Bevel Property (Panel Object)

See Also        Example

Returns or sets the bevel style of a **StatusBar** control's **Panel** object.

---

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

---

### Syntax

*object.***Bevel** [= *value*]

The **Bevel** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Panel** object. |
| *value* | A constant or value which determines the bevel style, as specified in Settings. |

### Settings

The settings for *value* are:

| Constant | Value | Description |
|---|---|---|
| **sbrNoBevel** | 0 | None. The **Panel** displays no bevel, and text looks like it is displayed right on the status bar. |
| **sbrInset** | 1 | (Default). Inset. The **Panel** appears to be sunk into the status bar. |
| **sbrRaised** | 2 | Raised. The **Panel** appears to be raised above the status bar. |

**See Also**

**Add** Method (**Panels** Collection)
**Panel** Object, **Panels** Collection
**StatusBar** Control
**StatusBar** Control Constants

■

## Bevel Property (Panel Object) Example

This example adds two **Panel** objects to a **StatusBar** control, and gives each **Panel** a different bevel style.   To use the example, place a **StatusBar** control on a form and paste the code into the Declarations section.   Run the example.

```
Private Sub Form_Load()
   Dim pnlX As Panel
   Dim I as Integer

   For I = 1 to 2
      Set pnlX = StatusBar1.Panels.Add() ' Add 2 panels.
   Next I

   With StatusBar1.Panels
      .Item(1).Style = sbrCaps      ' Caps Lock
      .Item(1).Bevel = sbrInset     ' Inset
      .Item(2).Style = sbrNum' NumLock
      .Item(2).Bevel = sbrNoBevel   ' No bevel
      .Item(3).Style = sbrDate      ' Date
      .Item(3).Bevel = sbrRaised    ' Raised bevel
   End With
End Sub
```

## MinWidth Property

See Also Example

Returns or sets the minimum width of a **StatusBar** control's **Panel** object.

---

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

---

### Syntax

*object.***MinWidth** [= *value*]

The **MinWidth** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Panel** object. |
| *value* | An integer that determines the minimum width of a **Panel** object. The scale mode for this value is determined by the container of the control. |

### Remarks

The default value is the same as the default of the **Width** property. The *value* argument uses the same scale units as the scale mode of the parent form or container.

The **Width** property always reflects the actual width of a **Panel**. The **Width** and **MinWidth** properties can only be different if the **AutoSize** property is set to Spring style and there is extra space in the status bar. In this case, the **Panel** is widened.

**See Also**

**AutoSize** Property (**Panel** Object)
**Panel** Object, **Panels** Collection
**Width** Property (**Panel** Object)
**StatusBar** Control

■

## MinWidth Property Example

This example creates a **StatusBar** control with three **Panel** objects, and sets each of their **MinWidth** properties to different values. To use the example, place a **StatusBar** control on a form, and paste the code into the Declarations section. Run the example and click on any **Panel** to make it grow.

```
Private Sub Form_Load()
   Dim I as Integer
   Form1.ScaleMode = vbTwips 'Twips
   For I = 1 to 2
      StatusBar1.Panels.Add  ' Add 2 panels.
   Next I

With StatusBar1.Panels
   .Item(1).Text = "Short"
   .Item(1).AutoSize = sbrSpring    ' AutoSize = Spring
   .Item(1).MinWidth = 200          ' A short panel
   .Item(2).Text = "Long"
   .Item(2).AutoSize = sbrSpring    ' AutoSize = Spring
   .Item(2).MinWidth = 1000         ' A long panel
   .Item(3).Style = sbrTime         ' Time
   .Item(3).AutoSize = sbrSpring    ' Spring
End With
End Sub

Private Sub StatusBar1_PanelClick(ByVal Panel As Panel)
   Panel.MinWidth = 2000
End Sub
```

## Panels Property

See Also

Returns a reference to a collection of **Panel** objects.

---

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

---

### Syntax

*object*.**Panels**

The *object* placeholder is an object expression that evaluates to a **StatusBar** control.

**See Also**

**Add** Method (**Panels** Collection)
**Count** Property
**Item** Method
**Panel** Object, **Panels** Collection

## PanelClick Event

See Also Example

Similar to the standard Click event, but the PanelClick event occurs when a user presses and then releases a mouse button over any of the **StatusBar** control's **Panel** objects.

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

**Private Sub** *object*_**PanelClick(ByVal** *panel* **As Panel)**

The PanelClick event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **StatusBar** control. |
| *panel* | A reference to a **Panel** object. |

### Remarks

The standard Click event also occurs when a **Panel** object is clicked.

The PanelClick event is only generated when the click occurs over a **Panel** object. When the **StatusBar** control's **Style** property is set to Simple style, panels are hidden, and therefore the PanelClick event is not generated.

You can use the reference to the **Panel** object to set properties for that panel. For example, the following code resets the **Bevel** property of a clicked **Panel**:

```
Private Sub StatusBar1_PanelClick(ByVal Panel As Panel)
If Panel.Index = 1 Then
   Panel.Bevel = sbrRaised   ' Reset Bevel property.
End If
End Sub
```

**See Also**

Click Event
**Panel** Object, **Panels** Collection
PanelDblClick Event
**StatusBar** Control
**Style** Property (**Panel** Object)

■

## PanelClick Event Example

This example adds two **Panel** objects to a **StatusBar** control; when each **Panel** is clicked, the value of the **Key** and **Width** properties of the clicked **Panel** are displayed in the second **Panel**. To try the example, place a **StatusBar** control on a form and paste the code into the Declarations section. Run the example.

```
Private Sub Form_Load()
   Dim I as Integer
   For I = 1 to 2
      StatusBar1.Panels.Add
   Next I

   With StatusBar1.Panels
      .Item(1).Style = sbrDate
      .Item(1).Key = "Date panel"
      .Item(1).AutoSize = sbrContents
      .Item(1).MinWidth = 2000
      .Item(2).Style = sbrTime
      .Item(2).Key = "Time panel"
      .Item(3).AutoSize = sbrContents    ' Content
      .Item(3).Text = "Miscellaneous Data"
      .Item(3).Key = "Panel 3"
   End With
End Sub

Private Sub StatusBar1_PanelClick(ByVal Panel As Panel)
   ' Show clicked panel's key and width in Panel 3.
   StatusBar1.Panels(3).Text = Panel.Key & " Width = " & Panel.Width
End Sub
```

## PanelDblClick Event

See Also  Example

Similar to the standard DblClick Event, the PanelDblClick occurs when a user presses and then releases a mouse button twice over a **StatusBar** control's **Panel** object.

---

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

---

### Syntax

**Sub** *object*_**PanelDblClick(ByVal** *panel* **As Panel)**

The PanelDblClick event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **StatusBar** control. |
| *panel* | A reference to the double-clicked **Panel**. |

### Remarks

The standard DblClick event also occurs when a **Panel** is double-clicked.

The PanelDblClick event is only generated when the double-click occurs over a **Panel** object. When the **StatusBar** control's **Style** property is set to Simple style, panels are hidden, and therefore the PanelDblClick event is not generated.

**See Also**

[DblClick Event]
PanelClick Event
**Panel** Object, **Panels** Collection
**StatusBar** Control

▪

## PanelDblClick Event Example

This example adds two **Panel** objects to a **StatusBar** control. When the user double-clicks on the control, the text of the clicked **Panel** object is displayed. To try the example, place a **StatusBar** control on a form and paste the code into the form's Declarations section. Run the example and double-click on the control.

```
Private Sub Form_Load()
Dim I as Integer
   For I = 1 to 2
      StatusBar1.Panels.Add
   Next I

   With StatusBar1.Panels
      .Item(1).Text = "A long piece of information."
      .Item(1).AutoSize = sbrContents     ' Content
      .Item(2).Style = sbrDate            ' Date style
      .Item(2).AutoSize = sbrContents     ' Content
      .Item(3).Style = sbrTime            ' Time style
   End With
End Sub

Private Sub StatusBar1_PanelDblClick(ByVal Panel As Panel)
   MsgBox Panel.Style
End Sub
```

## SimpleText Property

See Also    Example

Returns or sets the text displayed when a **StatusBar** control's **Style** property is set to Simple.

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object.***SimpleText** [= *string*]

The **SimpleText** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **StatusBar** control. |
| *string* | A string that is displayed when the **Style** property is set to Simple. |

### Remarks

The **StatusBar** control has a **Style** property which can be toggled between Simple and Normal styles. When in Simple style, the status bar displays only one **Panel**. The text displayed in Simple style is also different from that displayed in Normal style. This text is set with the **SimpleText** property.

The **SimpleText** property can be used in situations where an application's mode of operation temporarily switches. For example, when a menu is pulled down, the **SimpleText** could describe the menu's purpose.

**See Also**

**StatusBar** Control

**Style** Property (**StatusBar** Control)

■

## SimpleText Property Example

This example adds two **Panel** objects to a **StatusBar** control that appear in Normal style, and then adds a string (using the **SimpleText** property) that appears when the **Style** property is set to Simple. The control toggles between the Simple style and the Normal style. To try the example, place a **StatusBar** control on a form and paste the code into the Declarations section of the form. Run the example and click on the **StatusBar** control.

```
Private Sub Form_Load()
   Dim I As Integer
   For I = 1 to 2
      StatusBar1.Panels.Add        ' Add 2 Panel objects.
   Next I

   With StatusBar1.Panels
      .Item(1).Style = sbrNum      ' Number lock
      .Item(2).Style = sbrCaps     ' Caps lock
      .Item(3).Style = sbrScrl     ' Scroll lock
   End With

   ' This text will be displayed when the StatusBar is in Simple style.
   StatusBar1.SimpleText = "Date and Time: " & Now
End Sub

Private Sub StatusBar1_Click()
   ' Toggle between simple and normal style.
   With StatusBar1
      If .Style = 0 Then
         .Style = sbrSimple        ' Simple style.
      Else
         .Style = sbrNormal        ' Normal style.
      End If
   End With
End Sub
```

## Style Property (StatusBar Control)

See Also Example

Returns or sets the style of a **StatusBar** control.

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object.***Style** [= *number*]

The **Style** property syntax has these parts:

| Part | Description |
|---|---|
| object | An object expression that evaluates to a **StatusBar** control. |
| number | An integer or constant that determines the appearance of the **StatusBar** control, as specified in Settings. |

### Settings

The settings for *number* are:

| Constant | Value | Description |
|---|---|---|
| **sbrNormal** | 0 | (Default). Normal. The **StatusBar** control shows all **Panel** objects. |
| **sbrSimple** | 1 | Simple. The control displays only one large **Panel**. |

### Remarks

The **StatusBar** can toggle between two modes: Normal and Simple. When in Simple style, the **Statusbar** displays only one panel. The appearance also changes: the bevel style is raised with no borders. This allows the control to have two appearances, both of which are maintained separately from each other.

You can display different strings depending on the control's style. Use the **SimpleText** property to set the text of the string to be displayed when the **Style** property is set to Simple.

**See Also**

SimpleText Property
StatusBar Control
**Style** Property (**Panel** Object)
**StatusBar** Control Constants

▪

## Style Property (StatusBar Control) Example

This example adds two **Panel** objects to a **StatusBar** control that appear in Normal style, and then adds a string (using the **SimpleText** property) that will appear when the **Style** property is set to Simple.   The control toggles between the Simple style and the Normal style to show the **SimpleText** property string. To try the example, place a **StatusBar** control on a form and paste the code into the Declarations section of the form.   Run the example and click on the **StatusBar** control.

```
Private Sub Form_Load()
   Dim I As Integer
   For I = 1 to 2
      StatusBar1.Panels.Add
   Next I
   With StatusBar1.Panels
      .Item(1).Style = sbrDate       ' Date
      .Item(2).Style = sbrCaps       ' Caps lock
      .Item(3).Style = sbrScrl       ' Scroll lock
   End With
   StatusBar1.SimpleText = Time     ' Show the time.
End Sub

Private Sub StatusBar1_Click()
   With StatusBar1
      If .Style = sbrNormal Then
         .Style = sbrSimple          ' Simple style
      Else
         .Style = sbrNormal          ' Normal style
      End If
   End With
End Sub
```

## Style Property (Panel Object)

See Also  Example

Returns or sets the style of a **StatusBar** control's **Panel** object.

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object.***Style** [= *number*]

The **Style** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Panel** object. |
| *number* | An integer or constant specifying the style of the **Panel**, as described in Settings. |

### Settings

The settings for *number* are:

| Constant | Value | Description |
|---|---|---|
| **sbrText** | 0 | (Default). Text and/or a bitmap. Set text with the **Text** property. |
| **sbrCaps** | 1 | Caps Lock key. Displays the letters CAPS in bold when Caps Lock is enabled, and dimmed when disabled. |
| **sbrNum** | 2 | Number Lock. Displays the letters NUM in bold when the number lock key is enabled, and dimmed when disabled. |
| **sbrIns** | 3 | Insert key. Displays the letters INS in bold when the insert key is enabled, and dimmed when disabled. |
| **sbrScrl** | 4 | Scroll Lock key. Displays the letters SCRL in bold when scroll lock is enabled, and dimmed when disabled. |
| **sbrTime** | 5 | Time. Displays the current time in the system format. |
| **sbrDate** | 6 | Date. Displays the current date in the system format. |

### Remarks

If you set the **Style** property to any style except 0 (text and bitmap), any text set with the **Text** property will not display unless the **Style** property is set to 0.

The **Style** property can be set as **Panel** objects are added to a collection. See the **Add** method for more information.

**Note** The **StatusBar** control also has a **Style** property. When the **StatusBar** control's **Style** is set to Simple, the control displays only one large panel and its string (set with the **SimpleText** property).

**See Also**

**Add** Method (**Panels** Collection)
**SimpleText** Property
**StatusBar** Control Constants
**Style** Property (**StatusBar** Control)

■

## Style Property (Panel Object) Example

This example displays data in the various styles on a **StatusBar** control.   To try this example, place a **StatusBar** control on a form and paste the code into the form's Declarations section, and run the example.

```
Private Sub Form_Load()
   ' Dim variables.
   Dim I as Integer
   Dim pnlX as Panel

   For I = 1 to 5                    ' Add 5 panels.
      Set pnlX = StatusBar1.Panels.Add( )
   Next I

   ' Set the style of each panel.
   With StatusBar1.Panels
      .Item(1).Style = sbrDate     ' Date
      .Item(2).Style = sbrTime     ' Time
      .Item(3).Style = sbrCaps     ' Caps lock
      .Item(4).Style = sbrNum      ' Number lock
      .Item(5).Style = sbrIns      ' Insert key
      .Item(6).Style = sbrScrl     ' Scroll lock
   End With
   Form1.Width = 9140 ' Widen form to show all panels.
End Sub
```

## Width Property (Panel Object)

See Also Example

Returns or sets the current width of a **StatusBar** control's **Panel** object.

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object*.**Width**[= *number*]

The **Width** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Panel** object. |
| *number* | An integer that determines the width of the **Panel**. |

### Remarks

The **Width** property always reflects the actual width of a **Panel**. The **Width** and **MinWidth** properties can only be different if the **AutoSize** property is set to Spring, and if there is extra space in the status bar. In this case the **Panel** is widened.

The **Width** property can't be smaller than the **MinWidth** property.

**See Also**

**AutoSize** Property (**Panel** Object)
**MinWidth** Property
**Panel** Object, **Panels** Collection
**StatusBar** Control

■

## Width Property (Panel Object) Example

This example creates three **Panel** objects and sets their **Width** property to different values. When you click on the form, the **Width** property of the first **Panel** is reset. To try the example, place a **StatusBar** control on a form, and paste the code into the Declarations section. Run the example and click on each panel to see its width.

```
Private Sub Form_Load()
   Dim X As Panel
   Dim I as Integer
   For I = 1 to 2                          ' Add 2 panels.
      Set X = StatusBar1.Panels.Add()
   Next I
   With StatusBar1.Panels
      .Item(1).Text = "Path = " & App.Path
      .Item(1).AutoSize = sbrContents      ' Contents
      .Item(1).Width = 5000                ' A long panel
      .Item(2).Text = "Record Field"
      .Item(2).AutoSize = sbrSpring        ' Spring
      .Item(2).Width = 1000                ' A medium panel
      .Item(3).Style = sbrTime             ' Time
      .Item(3).AutoSize = sbrSpring        ' Spring
      .Item(3).Width = 1000                ' A medium panel
   End With
End Sub

Private Sub StatusBar1_PanelClick(ByVal Panel As Panel)
   MsgBox Panel.Width            ' Click each Panel to see its width.
End Sub

Private Sub Form_Click()
   ' Change Width.
   StatusBar1.Panels(1). Width = 800
End Sub
```

## Panels

The **Panels** keyword is used in these contexts:

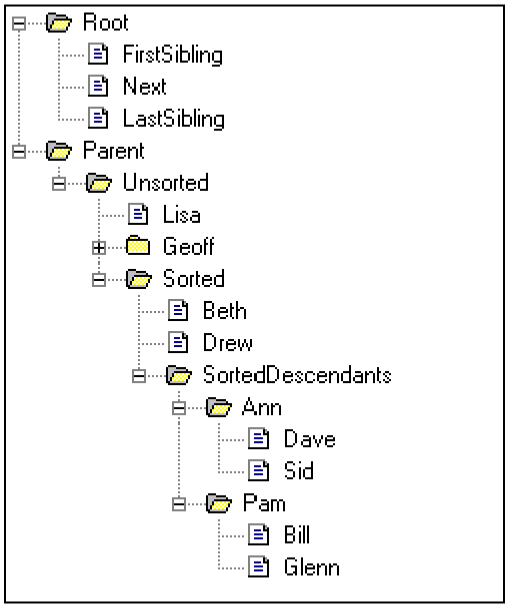
**Panels** Collection

**Panels** Property

## TreeView Control

See Also Properties Methods Events Constants

A **TreeView** control displays a hierarchical list of **Node** objects, each of which consists of a label and an optional bitmap. A **Treeview** is typically used to display the headings in a document, the entries in an index, the files and directories on a disk, or any other kind of information that is conducive to a hierarchical view.



### Syntax

**Treeview**

### Remarks

After creating a **TreeView** control, you can add, remove, arrange, and otherwise manipulate **Node** objects by setting properties and invoking methods. You can programmatically expand and collapse **Node** objects to display or hide all child nodes. Three events, the Collapse, Expand, and NodeClick event, also provide programming functionality.

You can navigate through a tree in code by retrieving a reference to **Node** objects using **Root**, **Parent**, **Child**, **FirstSibling**, **Next**, **Previous**, and **LastSibling** properties. Users can navigate through a tree using the keyboard as well. UP ARROW and DOWN ARROW keys cycle downward through all expanded **Node** objects. **Node** objects are selected from left to right, and top to bottom. At the bottom of a tree, the selection jumps back to the top of the tree, scrolling the window if necessary. RIGHT ARROW and LEFT ARROW keys also tab through expanded **Node** objects, but if the RIGHT ARROW key is pressed while an unexpanded **Node** is selected, the **Node** expands; a second press will move the selection to the next **Node.** Conversely, pressing the LEFT ARROW key while an expanded **Node** has the focus collapses the **Node**. If a user presses an ANSI key, the focus will jump to the nearest **Node** that begins with that letter. Subsequent pressings of the key will cause the selection to cycle downward through all expanded nodes that begin with that letter.

Several styles are available which alter the appearance of the control. **Node** objects can appear with text, bitmaps, lines, and plus/minus signs, or one of seven combinations of the above.

The **TreeView** control uses the **ImageList** control, specified by the **ImageList** property, to store the bitmaps and icons that are displayed in **Node** objects. A **TreeView** control can use only one **ImageList** at a time. This means that every item in the **TreeView** control will have an equal-sized image next to it when the **TreeView** control's **Style** property is set to a style which displays images.

---

**Distribution Note** The **TreeView** control is a 32-bit custom control that can only run on Windows 95 or Windows NT 3.51 or higher. The **TreeView** control is part of a group of custom controls that are found in the COMCTL32.OCX file.   To use the **TreeView** control in your application, you must add the COMCTL32.OCX file to the project.   When distributing your application, install the COMCTL32.OCX file in the user's Microsoft Windows SYSTEM directory.

**See Also**

**ImageList** Control

**Node** Object, **Nodes** Collection

Close

**Treeview Control Properties**

- **BorderStyle** Property
- **Container** Property
- **DragIcon** Property
- **DragMode** Property
- **DropHighlight** Property
- **Enabled** Property
- **Font** Property
- **Height, Width** Properties
- **HelpContextID** Property
- **HideSelection** Property
- **hWnd** Property
- **ImageList** Property
- **Indentation** Property
- **Index** Property
- **LabelEdit** Property
- **Left, Top** Properties
- **LineStyle** Property
- **MousePointer** Property
- **MouseIcon** Property
- **Name** Property
- **Nodes** Property
- **PathSeparator** Property
- **Parent** Property
- **Scrollbars** Property
- **SelectedItem** Property
- **Sorted** Property
- **Style** Property (**Treeview** Control)
- **TabIndex** Property
- **TabStop** Property
- **Tag** Property
- **Visible** Property
- **WhatsThisHelpID** Property

Close

**Treeview Control Methods**

**Clear** Method
**GetVisibleCount** Method
**HitTest** Method
**Move** Method
**Refresh** Method
**Remove** Method
**SetFocus** Method
**StartLabelEdit** Method
**ShowWhatsThis** Method
**ZOrder** Method

Close

## Treeview Control Events

AfterLabelEdit Event
BeforeLabelEdit Event
Click Event
Collapse Event
DblClick Event
DragDrop Event
DragOver Event
Expand Event
GotFocus Event
KeyDown Event
KeyPress Event
KeyUp Event
LostFocus Event
MouseDown Event
MouseMove Event
MouseUp Event
NodeClick Event

## TreeView Control Constants

See Also

### TreeLine Constants

| Constant | Value | Description |
|---|---|---|
| **tvwTreeLines** | 0 | Treelines shown. |
| **tvwRootLines** | 1 | Rootlines shown with Treelines. |

### TreeRelationship Constants

| Constant | Value | Description |
|---|---|---|
| **tvwFirst** | 0 | First Sibling. |
| **tvwLast** | 1 | Last Sibling. |
| **tvwNext** | 2 | Next sibling. |
| **tvwPrevious** | 3 | Previous sibling. |
| **tvwChild** | 4 | Child. |

### TreeStyle Constants

| Constant | Value | Description |
|---|---|---|
| **tvwTextOnly** | 0 | Text only. |
| **tvwPictureText** | 1 | Picture and text. |
| **tvwPlusMinusText** | 2 | Plus/minus and text. |
| **tvwPlusPictureText** | 3 | Plus/minus, picture, and text. |
| **tvwTreelinesText** | 4 | Treelines and text. |
| **tvwTreelinesPictureText** | 5 | Teelines, Picture, and Text. |
| **tvwTreelinesPlusMinusText** | 6 | Treelines, Plus/Minus, and Text. |
| **tvwTreelinesPlusMinusPictureText** | 7 | Treelines, Plus/Minus, Picture, and Text. |

### LabelEdit Constants

| Constant | Value | Description |
|---|---|---|
| **tvwAutomatic** | 0 | Label Editing is automatic. |
| **tvwManual** | 1 | LabelEditing must be invoked. |

**See Also**

**Add** Method (**Nodes** Collection)
**Child** Property
**FirstSibling** Property
**LastSibling** Property
**LineStyle** Property
**Next** Property
**Style** Property (**TreeView** Control)
**TreeView** Control
Visual Basic Custom Control Constants
Windows 95 Controls Constants

## Node Object, Nodes Collection

See Also Properties Methods

- A **Node** object is an item in a **TreeView** control that can contain images and text.
- A **Nodes** collection contains one or more **Node** objects.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*treeview*.**Nodes**

*treeview.***Nodes.Item(***index***)**

The syntax lines above refer to the collection and to individual elements in the collection, respectively, according to standard collection syntax.

The **Node** object, **Nodes** collection syntax has these parts:

| Part | Description |
|---|---|
| *treeview* | An object expression that evaluates to a **TreeView** control. |
| *index* | Either an integer or string that uniquely identifies a member of a **Nodes** collection. The integer is the value of the **Index** property; the string is the value of the **Key** property. |

### Remarks

Nodes can contain both text and pictures. However, to use pictures, you must associate an **ImageList** control using the **ImageList** property.

Pictures can change depending on the state of the node; for example, a selected node can have a different picture from an unselected node if you set the **SelectedImage** property to an image from the associated **ImageList**.

**See Also**

**Add** Method (**Nodes** Collection)
**ImageList** Control
**ImageList** Property
**Nodes** Property
**TreeView** Control

Close

## Node Object, Nodes Collection Properties

Legend

**Child** Property■
**Children** Property■
**Count** Property■
**Enabled** Property■
**Expanded** Property■
**ExpandedImage** Property■
**FirstSibling** Property■
**FullPath** Property■
**Image** Property■
**Index** Property■
**Key** Property■
**LastSibling** Property■
**Next** Property■
**Parent** Property (**Node** Object)■
**Previous** Property (**Node** Object)■
**Root** Property (**Node** Object)■
**Selected** Property■
**SelectedImage** Property■
**Sorted** Property (**TreeView**)■
**Tag** Property■
**Text** Property■
**Visible** Property■

Close

**Node Object, Nodes Collection Methods**

Legend

**Add** Method (Nodes Object)■

**Clear** Method■
**CreateDragImage** Method■
**EnsureVisible** Method■
**Item** Method■
**Remove** Method■

## Add Method (Nodes Collection)

See Also Example

Adds a **Node** object to a **Treeview** control's **Nodes** collection. Doesn't support named arguments.

---

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**Add(***relative, relationship, key, text, image, selectedimage***)**

The **Add** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **Nodes** collection. |
| *relative* | Optional. The index number or key of a pre-existing **Node** object. The relationship between the new node and this pre-existing node is found in the next argument, *relationship*. |
| *relationship* | Optional. Specifies the relative placement of the **Node** object, as described in Settings. |
| *key* | Optional. A unique string that can be used to retrieve the **Node** with the **Item** method. |
| *text* | Required. The string that appears in the **Node**. |
| *image* | Optional. The index of an image in an associated **ImageList** control. |
| *selectedimage* | Optional. The index of an image in an associated **ImageList** control that is shown when the **Node** is selected. |

### Settings

The settings for *relationship* are:

| Constant | Value | Description |
|---|---|---|
| **tvwLast** | 1 | Last. The **Node** is placed after all other nodes at the same level of the node named in *relative*. |
| **tvwNext** | 2 | Next. The **Node** is placed after the node named in *relative*. |
| **tvwPrevious** | 3 | Previous. The **Node** is placed before the node named in *relative*. |
| **tvwChild** | 4 | (Default) Child. The **Node** becomes a child node of the node named in *relative*. |

---

**Note** If no **Node** object is named in *relative*, the new node is placed in the last position of the top node hierarchy.

---

### Remarks

The **Nodes** collection is a 1-based collection.

As a **Node** object is added it is assigned an index number, which is stored in the **Node** object's **Index** property. This value of the newest member is the value of the **Node** collection's **Count** property plus 1.

Because the **Add** method returns a reference to the newly created **Node** object, it is most convenient to set properties of the new **Node** using this reference. The following example adds several **Node** objects with identical properties:

```
Dim nodX As Node     ' Create the object variable.
Dim I as Integer     ' Create a counter variable.
For I = 1 to 4
   Set nodX = TreeView1.Nodes.Add(,,,"Node " & Cstr(i))
   ' Use the reference to set other properties, such as Enabled.
   nodX.Enabled = True
   ' Set image property to image 3 in an associated ImageList.
   nodX.ExpandedImage = 3
```

```
Next I
```

**See Also**

**Clear** Method
**Index** Property
**Item** Method (Custom Controls)
**Key** Property
**Node** Object, **Nodes** Collection
**Remove** Method (Custom Controls)
**TreeView** Control

## Add Method Example (Nodes Collection)

The following example adds several **Node** objects with images to a **TreeView** control. To try the example, place a **TreeView** control and an **ImageList** control on a form, paste the code into the form's Declarations section. Run the example, and click other **Node** objects to see their keys.

```
Private Sub Form_Load()
   ' Load pictures into ImageList control.
   Dim imgI As ListImage            ' Create Image variable.
   ' Image 1: Open folder, key = "open."
   Set imgI = ImageList1.ListImages.Add _
   (, "open", LoadPicture("bitmaps\outline\open.bmp"))
   ' Image 2: Closed folder, key = "closed."
   Set imgI = ImageList1.ListImages.Add _
   (, "closed", LoadPicture("bitmaps\outline\closed.bmp"))
   ' Image 3: document, key = "leaf."
   Set imgI = ImageList1.ListImages.Add _
   (, "leaf", LoadPicture("bitmaps\outline\leaf.bmp"))

   ' Set Treeview control properties.
   TreeView1.ImageList = ImageList1' Initialize ImageList.
   TreeView1.Style = tvwTreelinesPlusMinusPictureText ' Style 7
   TreeView1.LineStyle = tvwRootLines    ' Linestyle 1

   ' Add Node objects.
   Dim nodX As Node                ' Create variable.

   ' First node with 'Root' as text, image 2 ("closed") for Image.
   Set nodX = TreeView1.Nodes.Add(, ,"r", "Root", "closed")
   nodX.ExpandedImage = "open"  ' Open folder for expanded node.

   ' Second node has 'Parent' as text, image 2 for Image.
   Set nodX = TreeView1.Nodes.Add(, , "p", "Parent", "closed")
   nodX.ExpandedImage = "open"  ' Open folder for expanded node.

   ' This next node is a child of Node 1 ("Root"), and uses
   ' image 3 ("leaf") for Image.
   Set nodX = TreeView1.Nodes.Add(1,tvwChild,"c", "Child", "leaf")

   ' This next node is a child of "p." Instead of using an index,
   ' to specify the relative, we use its key "p."
   Set nodX = TreeView1.Nodes.Add _
   ("p", tvwChild, "uns", "Unsorted", "closed")
   nodX.ExpandedImage = "open"

   ' Add three Nodes, children of "Unsorted."
   Set nodX = TreeView1.Nodes. _
   Add("uns",tvwChild,"xx","Xu Xiang","leaf")
   Set nodX = TreeView1.Nodes. _
   Add("uns",tvwChild,"date","1967","leaf")
   Set nodX = TreeView1.Nodes. _
   Add("uns",tvwChild,"srt","Sorted",2)
   nodX.ExpandedImage = "open"
   ' Children of last created node will be sorted.
   nodX.Sorted = True
   ' Add three Nodes, children of "Sorted," with image "leaf."
   Set nodX = TreeView1.Nodes.Add("srt",tvwChild,"x","X", "leaf")
   Set nodX = TreeView1.Nodes.Add("srt",tvwChild,"j","J", "leaf")
   Set nodX = TreeView1.Nodes.Add("srt",tvwChild,"a","A", "leaf")
   nodX.EnsureVisible ' Expand tree to see all nodes
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
   ' Form caption shows index, parent, and key.
   Dim strI As String
   strI = "Index = " & Node.Index
   On Error Resume Next ' Level 1 nodes have no parents--an error.
   strI = strI & ": Parent =" & Node.Parent.Text
   strI = strI & ": Key =" & Node.Key
   Me.Caption = strI
End Sub
```

## Child Property

See Also Example

Returns a reference to the first child of a **Node** object in a **TreeView** control.

---

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**Child**

The *object* placeholder represents an object expression that evaluates to a **Node** object.

### Remarks

The **Child**, **FirstSibling**, **LastSibling**, **Previous**, **Parent**, **Next**, and **Root** properties all return a reference to another **Node** object. Therefore, you can simultaneously reference and perform operations on a **Node**, as follows:

```
With TreeView1.Nodes(x).Child
   .Text = "New text"
   .Key = "New key"
   .SelectedImage = 3
End With
```

You can also set an object variable to the referenced **Node**, as follows:

```
Dim NodChild As Node
' Get a reference to the child of Node x.
Set NodChild = TreeView1.Nodes(x).Child
' Use this reference to perform operations on the child Node.
With nodChild
   .Text = "New text"              '  Change the text.
   .Key = "New key"                ' Change key.
   .SelectedImage = 3              ' Change SelectedImage.
End With
```

**See Also**

**Children** Property
**Node** Object, **Nodes** Collection
**Parent** Property (**Node** Object)
**TreeView** Control

▪
## Child Property Example

This example creates several **Node** objects. When you click on a **Node** object, the code uses the **Child** property to navigate down the tree and return the names of all **Child** nodes. As long as a **Node** object has a **Child** node, the text of that **Child** node will be stored in a variable. The process stops when a **Node** has no children. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the program and click on any **Node** to see its children and descendants.

```
Private Sub Form_Load()
   TreeView1.Style = tvwTreelinesPlusMinusText ' Style 6.
   TreeView1.LineStyle = tvwRootLines   'Linestyle 1.

   ' Add several Node objects.
   Dim nodX As Node                     ' Create variable.

   Set nodX = TreeView1.Nodes.Add(, , "r", "Root")
   Set nodX = TreeView1.Nodes.Add("r", tvwChild, "c1", "Child 1")
   Set nodX = TreeView1.Nodes.Add("c1", tvwChild, "c2", "Child 2")
   Set nodX = TreeView1.Nodes.Add("c2", tvwChild, "c3", "Child 3")
   Set nodX = TreeView1.Nodes.Add("c3", tvwChild, "c4", "Child 4")
   Set nodX = TreeView1.Nodes.Add("c2", tvwChild, "c5", "Child 5")
   Set nodX = TreeView1.Nodes.Add("c3", tvwChild, "c6", "Child 6")
   Set nodX = TreeView1.Nodes.Add("c4", tvwChild, "c7", "Child 7")
   nodX.EnsureVisible ' Show all nodes.
   Set nodX = TreeView1.Nodes.Add("c5", tvwChild, "c8", "Child 8")
   Set nodX = TreeView1.Nodes.Add("c8", tvwChild, "c9", "Child 9")
   nodX.EnsureVisible ' Show all nodes.
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
   Dim strC As String
   Dim n As Integer
   ' Set n to current node's index.
   n = Node.Index
   Dim blnFlag As Boolean
   blnFlag = True
   ' Put current node's text into the string variable.
   strC = Node.Text & Chr(10)
   ' Create two Node variables.
   Dim nod1, nod2 As Node
   While blnFlag
      ' Set first variable to child of Node n.
      Set nod1 = TreeView1.Nodes(n).Child
      If nod1 is Nothing then
         blnFlag = False
      Else
      ' Put text of child node into string variable.
      strC = strC & nod1.Text & Chr(10)
      ' Reset n to child node's index.
      n = TreeView1.Nodes(n).Child.Index
      ' Set second variable to next child.
      Set nod2 = TreeView1.Nodes(n).Child
      ' If next child's index = n, then stop.
```

```
         If nod2 Is Nothing Then
            blnFlag = False
         End If
      End If
   Wend
   MsgBox strC ' Show Child nodes.
End Sub
```

## Children Property

See Also Example

Returns the number of children a **Node** object has.

---

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object.***Children**

The *object* placeholder represents an object expression that evaluates to a **Node** object.

### Remarks

The **Children** property can be used to check if a **Node** object has any children before performing an operation that affects the children. For example, the following code checks for the presence of children nodes before retrieving the **Text** property of the first **Node**, using the **Child** property.

```
Private Sub TreeView1_NodeClick(ByVal Node As Node)
   If Node.Children > 0 Then
      MsgBox Node.Child.Text
   End If
End Sub
```

**See Also**

**Child** Property

**Node** Object, **Nodes** Collection

**TreeView** Control

▪
## Children Property Example

This example puts several **Node** objects in a **TreeView** control. The code checks to see if a **Node** has children nodes. If so, then it displays the text of the children nodes. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, click a **Node** object to select it, then click the form to see the text of the **Node** object's children.

```
Private Sub Form_Click()
   Dim strC As String
   Dim N As Integer
   If TreeView1.SelectedItem.Children > 0 Then ' There are children.

      ' Get first child's text, and set N to its index value.
      strC = TreeView1.SelectedItem.Child.Text & Chr(10)
      N = TreeView1.SelectedItem.Child.Index

      ' While N is not the index of the child node's
      ' last sibling, get next sibling's text.
      While N <> TreeView1.SelectedItem.Child.LastSibling.Index
         strC = strC & TreeView1.Nodes(N).Next.Text & Chr(10)
         ' Reset N to next sibling's index.
         N = TreeView1.Nodes(N).Next.Index
      Wend
      ' Show results.
      MsgBox "Children of " & TreeView1.SelectedItem.Text & _
      " are: " & Chr(10) & strC
   Else ' There are no children.
      MsgBox TreeView1.SelectedItem.Text & " has no children"
   End If
End Sub

Private Sub Form_Load()
   TreeView1.BorderStyle = 1  ' Ensure border is visible
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,"d","Dates")
   Set nodX = TreeView1.Nodes.Add("d",tvwChild,"d89","1989")
   Set nodX = TreeView1.Nodes.Add("d",tvwChild,"d90","1990")

   ' Create children of 1989 node.
   Set nodX = TreeView1.Nodes.Add("d89",tvwChild, ,"John")
   Set nodX = TreeView1.Nodes.Add("d89",tvwChild, ,"Brent")
   Set nodX = TreeView1.Nodes.Add("d89",tvwChild, ,"Eric")
   Set nodX = TreeView1.Nodes.Add("d89",tvwChild, ,"Ian")
   nodX.EnsureVisible ' Show all nodes.

   ' Create children of 1990 node.
   Set nodX = TreeView1.Nodes.Add("d90",tvwChild, ,"Randy")
   Set nodX = TreeView1.Nodes.Add("d90",tvwChild, ,"Ron")
   nodX.EnsureVisible ' Show all nodes.
End Sub
```

## Collapse Event (TreeView Control)

See Also    Example

Generated when any **Node** object in a **TreeView** control is collapsed.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

**Private Sub** *object*_**Collapse(ByVal** *node* **As Node)**

The Collapse event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TreeView** control. |
| *node* | A reference to the clicked **Node** object. |

### Remarks

The Collapse event occurs before the standard Click event.

There are three methods of collapsing a **Node**: by setting the **Node** object's **Expanded** property to **False**, by double-clicking a **Node** object, and by clicking a plus/minus image when the **TreeView** control's **Style** property is set to a style that includes plus/minus images. All of these methods generate the Collapse event.

The event passes a reference to the collapsed **Node** object which can be used to validate an action, as in the following example:

```
Private Sub TreeView1_Collapse(ByVal Node As Node)
   If Node.Index = 1 Then
      Node.Expanded = True         ' Expand the node again.
   End If
End Sub
```

**See Also**

Expand Event
**Expanded** Property
**Node** Object, **Nodes** Collection
NodeClick Event
**SelectedItem** Property
**Style** Property (**TreeView** Control)
**TreeView** Control

■

## Collapse Event (TreeView Control) Example

This example adds one **Node** object, with several child nodes, to a **TreeView** control. When the user collapses a **Node,** the code checks to see how many children the **Node** has. If it has more than one child, the **Node** is re-expanded. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, and double-click a **Node** to collapse it and generate the event.

```
Private Sub Form_Load()
   TreeView1.Style = tvwTreelinesPlusMinusText ' Style 6.
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,"DV","Da Vinci")
   Set nodX = TreeView1.Nodes.Add("DV",tvwChild,"T","Titian")
   Set nodX = TreeView1.Nodes.Add("T",tvwChild,"R","Rembrandt")
   Set nodX = TreeView1.Nodes.Add("R",tvwChild,,"Goya")
   Set nodX = TreeView1.Nodes.Add("R",tvwChild,,"David")
   nodX.EnsureVisible              ' Show all nodes.
End Sub

Private Sub TreeView1_Collapse(ByVal Node As Node)
   ' If the Node has more than one child node,
   ' keep the node expanded.
   Select Case Node.Children
      Case Is > 1
         Node.Expanded = True
   End Select
End Sub
```

## AfterLabelEdit Event (ListView, TreeView Controls)

See Also Examples

Occurs after a user edits the label of the currently selected **Node** or **ListItem** object.

---

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

**Sub** *object*_**AfterLabelEdit(***cancel* **As Integer**, *newstring* **As String)**

The AfterLabelEdit event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListView** or **TreeView** control. |
| *cancel* | An integer that determines if the label editing operation is canceled. Any nonzero integer cancels the operation. Boolean values are also accepted. |
| *newstring* | The string the user entered, or **Null** if the user canceled the operation. |

### Remarks

Both the AfterLabelEdit and the BeforeLabelEdit events are generated only if the **LabelEdit** property is set to 1 (Automatic), or if the **StartLabelEdit** method is invoked.

The AfterLabelEdit event is generated after the user finishes the editing operation, which occurs when the user clicks on another **Node** or **ListItem** or presses the ENTER key.

To cancel a label editing operation, set *cancel* to any nonzero number or to **True**. If a label editing operation is canceled, the previously existing label is restored.

The *newstring* argument can be used to test for a condition before canceling an operation. For example, the following code verifies that *newstring* is a numeral before allowing the operation to conclude:

```
Private Sub TreeView1_AfterLabelEdit(Cancel As Integer, NewString As String)
   If IsNumeric(NewString) Then
      MsgBox "No numbers allowed"
      Cancel = True
   End If
End Sub
```

**See Also**

BeforeLabelEdit Event
**LabelEdit** Property
**ListItem** Object**, ListItems** Collection
**ListView** Control
NodeClick Event
**Node** Object, **Nodes** Collection
**StartLabelEdit** Method
**TreeView** Control

■

## AfterLabelEdit Event (ListView, TreeView Controls) Example

This example adds three **Node** objects to a **TreeView** control. When you attempt to edit a **Node** object's label, the object's index is checked. If it is 1, the operation is canceled. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, click twice on the top **Node** object's label to edit it, type in some text, and press ENTER.

```
Private Sub Form_Load()
   TreeView1.Style = tvwTreelinesText ' Lines and text.
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,,"Parent")
   Set nodX = TreeView1.Nodes.Add(1,tvwChild,,"Child1")
   Set nodX = TreeView1.Nodes.Add(1,tvwChild,,"Child2")
   nodX.EnsureVisible              ' Make sure all nodes are visible.
End Sub

Private Sub TreeView1_AfterLabelEdit _
(Cancel As Integer, NewString As String)
   ' If current node's index is 1, edit is canceled.
   If TreeView1.SelectedItem.Index = 1 Then
      Cancel = True
      MsgBox "Can't replace " & TreeView1.SelectedItem.Text & _
      " with " & NewString
   End If
End Sub
```

This example adds three **ListItem** objects to a **ListView** control. When you attempt to edit a **ListItem** object's label, the object's index is checked. If it is 1, the operation is canceled. To try the example, place a **ListView** control on a form and paste the code into the form's Declarations section. Run the example, click twice on any **ListItem** object's label to edit it, type in some text, and press ENTER.

```
Private Sub Form_Load()
   Dim itmX As ListItem
   Set itmX = ListView1.ListItems.Add(,,"Item1")
   Set itmX = ListView1.ListItems.Add(,,"Item 2")
   Set itmX = ListView1.ListItems.Add(,,"Item 3")
End Sub

Private Sub ListView1_AfterLabelEdit _
(Cancel As Integer, NewString As String)
   ' If current ListItem's index is 1, edit is canceled.
   If ListView1.SelectedItem.Index = 1 Then
      Cancel = True
      MsgBox "Can't replace " & ListView1.SelectedItem.Text & _
      " with " & NewString
   End If
End Sub
```

## BeforeLabelEdit Event (ListView, TreeView Controls)

See Also Example

Occurs when a user attempts to edit the label of the currently selected **ListItem** or **Node** object.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

**Sub** *object*_**BeforeLabelEdit(***cancel* **As Integer)**

The BeforeLabelEdit event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TreeView** or a **ListView** control. |
| *cancel* | An integer that determines if the operation is canceled. Any nonzero integer cancels the operation. The default is 0. |

### Remarks

Both the AfterLabelEdit and the BeforeLabelEdit events are generated only if the **LabelEdit** property is set to 1 (Automatic), or if the **StartLabelEdit** method is invoked.

The BeforeLabelEdit event occurs after the standard **Click** event.

To begin editing a label, the user must first click the object to select it, and click it a second time to begin the operation. The BeforeLabelEdit event occurs after the second click.

To determine which object's label is being edited, use the **SelectedItem** property. The following example checks the index of a selected **Node** before allowing an edit:

```
Private Sub TreeView1_BeforeLabelEdit(Cancel As Integer)
   If TreeView1.SelectedItem.Index = 1 Then
      Cancel = True          ' Cancel the operation
   End If
End Sub
```

**See Also**

AfterLabelEdit Event
**LabelEdit** Property
**ListView** Control
**Node** Object, **Nodes** Collection
**SelectedItem** Property
**StartLabelEdit** Method
**TreeView** Control

▪

## BeforeLabelEdit Event (ListView, TreeView Controls) Example

This example adds several **Node** objects to a **TreeView** control. If you try to edit a label, the **Node** object's index is checked. If it is 1, the edit is prevented. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, and try to edit the labels.

```
Private Sub Form_Load()
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,"P1","Parent 1")
   Set nodX = TreeView1.Nodes.Add("P1",tvwChild,,"Child 1")
   Set nodX = TreeView1.Nodes.Add("P1",tvwChild,,"Child 2")
   nodX.EnsureVisible              ' Make sure all nodes are visible.
End Sub

Private Sub TreeView1_BeforeLabelEdit(Cancel As Integer)
   ' Check selected node's index. If it is 1,
   ' then cancel the editing operation.
   If TreeView1.SelectedItem.Index = 1 Then
      MsgBox "Can't edit " + TreeView1.SelectedItem.Text
      Cancel = True
   End If
End Sub
```

This example adds several **ListItem** objects to a **ListView** control. If you try to edit a label, the **ListItem** object's index is checked. If it is 1, the edit is prevented. To try the example, place a **ListView** control on a form and paste the code into the form's Declarations section. Run the example, and try to edit the labels.

```
Private Sub Form_Load()
   Dim nodX As ListViewItem
   Set nodX = ListView1.ListItems.Add(, , "Item 1")
   Set nodX = ListView1.ListItems.Add(, , "Item 2")
   Set nodX = ListView1.ListItems.Add(, , "Item 3")
End Sub

Private Sub ListView1_BeforeLabelEdit(Cancel As Integer)
   ' Check selected item's index. If it is 1,
   ' then cancel the editing operation.
   If ListView1.SelectedItem.Index = 1 Then
      MsgBox "Can't edit " + ListView1.SelectedItem.Text
         Cancel = True
   End If
End Sub
```

## CreateDragImage Method

See Also Example

Creates a drag image using a dithered version of an object's associated image and label. This image is typically used in drag-and-drop operations.

---

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**CreateDragImage**

The *object* placeholder represents an object expression that evaluates to a **ListItem** or **Node** object.

### Remarks

The **CreateDragImage** method is typically used to assign an image to a **DragIcon** property at the start of a drag-and-drop operation.

**See Also**

**DropHighLight** Property
**HitTest** Method
**Node** Object, **Nodes** Collection
**TreeView** Control

▪

## CreateDragImage Method Example

This example adds several **Node** objects to a **TreeView** control. After you select a **Node** object, you can drag it to any other **Node**.   To try the example, place **TreeView** and **ImageList** controls on a form and paste the code into the form's Declaration section. Run the example and drag **Node** objects around to see the result.

```
' Declare global variables.
Dim indrag As Boolean ' Flag that signals a Drag Drop operation.
Dim nodX As Object ' Item that is being dragged.

Private Sub Form_Load()
   ' Load a bitmap into an Imagelist control.
   Dim imgX As ListImage
   Dim BitmapPath As String
   BitmapPath = "icons\mail\mail01a.ico"
   Set imgX = imagelist1.ListImages.Add(, , LoadPicture(BitmapPath))

   ' Initialize TreeView control and create several nodes.
   TreeView1.ImageList = imagelist1
   Dim nodX As Node  ' Create a tree.
   Set nodX = TreeView1.Nodes.Add(, , , "Parent1", 1)
   Set nodX = TreeView1.Nodes.Add(, , , "Parent2", 1)
   Set nodX = TreeView1.Nodes.Add(1, tvwChild, , "Child 1", 1)
   Set nodX = TreeView1.Nodes.Add(1, tvwChild, , "Child 2", 1)
   Set nodX = TreeView1.Nodes.Add(2, tvwChild, , "Child 3", 1)
   Set nodX = TreeView1.Nodes.Add(2, tvwChild, , "Child 4", 1)
   Set nodX = TreeView1.Nodes.Add(3, tvwChild, , "Child 5", 1)
   nodX.EnsureVisible ' Expand tree to show all nodes.
End Sub

Private Sub TreeView1_MouseDown_
(Button As Integer, Shift As Integer, x As Single, y As Single)
   Set nodX = TreeView1.SelectedItem ' Set the item being dragged.
End Sub

Private Sub TreeView1_MouseMove _
(Button As Integer, Shift As Integer, x As Single, y As Single)
   If Button = vbLeftButton Then ' Signal a Drag operation.
      indrag = True ' Set the flag to true.
      ' Set the drag icon with the CreateDragImage method.
      TreeView1.DragIcon = TreeView1.SelectedItem.CreateDragImage
      TreeView1.Drag vbBeginDrag ' Drag operation.
   End If
End Sub

Private Sub TreeView1_DragDrop_
(Source As Control, x As Single, y As Single)
   If TreeView1.DropHighlight Is Nothing Then
      Set TreeView1.DropHighlight = Nothing
      indrag = False
      Exit Sub
   Else
      If nodX = TreeView1.DropHighlight Then Exit Sub
      Cls
      Print nodX.Text & " dropped on " & TreeView1.DropHighlight.Text
      Set TreeView1.DropHighlight = Nothing
      indrag = False
   End If
End Sub

Private Sub TreeView1_DragOver(Source As Control, x As Single, y As Single,
State As Integer)
   If indrag = True Then
      ' Set DropHighlight to the mouse's coordinates.
      Set TreeView1.DropHighlight = TreeView1.HitTest(x, y)
   End If
End Sub
```

## DropHighlight Property (ListView, TreeView Controls)

See Also    Example

Returns or sets a reference to a **Node** or **ListItem** object that is highlighted with the system highlight color.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**DropHighlight** [ = *value*]

The **DropHighlight** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListView** or **TreeView** control. |
| *value* | A **Node** or **ListItem** object. |

### Remarks

The **DropHighlight** property is typically used in combination with the **HitTest** method in drag-and-drop operations. As the cursor is dragged over a control, the **HitTest** method returns a reference to any object it is dragged over. In turn, the **DropHighlight** property is set to the hit object, and simultaneously returns a reference to that object. The **DropHighlight** property then highlights the hit object with the system highlight color. The following code sets the **DropHighlight** property to the object hit with the **HitTest** method.

```
Private Sub TreeView1_DragOver _
(Source As Control, X As Single, Y As Single, State As Integer)
   Set TreeView1.DropHighlight = TreeView1.HitTest(X,Y)
End Sub
```

Subsequently, you can use the **DropHighlight** property in the DragDrop event to return a reference to the last object the source control was dropped over, as shown in the following code:

```
Private Sub TreeView1_DragDrop _
(Source As Control, x As Single, y As Single)
   ' DropHighlight returns a reference to object drop occurred over.
   Me.Caption = TreeView1.DropHighlight.Text
   ' To release the DropHighlight reference, set it to Nothing.
   Set TreeView1.DropHighlight = Nothing
End Sub
```

Note in the preceding example that the **DropHighlight** property is set to Nothing after the procedure is completed. This must be done to release the highlight effect.

**See Also**

**HitTest** Method
**ListView** Control
**TreeView** Control

■

## DropHighlight Property Example

This example adds several **Node** objects to a **TreeView** control. After you select a **Node** object, you can drag it to any other **Node**.   To try the example, place **TreeView** and **ImageList** controls on a form and paste the code into the form's Declaration section. Run the example and drag **Node** objects around to see the result.

```
' Declare global variables.
Dim indrag As Boolean ' Flag that signals a Drag Drop operation.
Dim nodX As Object ' Item that is being dragged.

Private Sub Form_Load()
   ' Load a bitmap into an Imagelist control.
   Dim imgX As ListImage
   Dim BitmapPath As String
   BitmapPath = "icons\mail\mail01a.ico"
   Set imgX = imagelist1.ListImages.Add(, , LoadPicture(BitmapPath))

   ' Initialize TreeView control and create several nodes.
   TreeView1.ImageList = imagelist1
   Dim nodX As Node  ' Create a tree.
   Set nodX = TreeView1.Nodes.Add(, , , "Parent1", 1)
   Set nodX = TreeView1.Nodes.Add(, , , "Parent2", 1)
   Set nodX = TreeView1.Nodes.Add(1, tvwChild, , "Child 1", 1)
   Set nodX = TreeView1.Nodes.Add(1, tvwChild, , "Child 2", 1)
   Set nodX = TreeView1.Nodes.Add(2, tvwChild, , "Child 3", 1)
   Set nodX = TreeView1.Nodes.Add(2, tvwChild, , "Child 4", 1)
   Set nodX = TreeView1.Nodes.Add(3, tvwChild, , "Child 5", 1)
   nodX.EnsureVisible ' Expand tree to show all nodes.
End Sub

Private Sub TreeView1_MouseDown_
(Button As Integer, Shift As Integer, x As Single, y As Single)
   Set nodX = TreeView1.SelectedItem ' Set the item being dragged.
End Sub

Private Sub TreeView1_MouseMove _
(Button As Integer, Shift As Integer, x As Single, y As Single)
   If Button = vbLeftButton Then ' Signal a Drag operation.
      indrag = True ' Set the flag to true.
      ' Set the drag icon with the CreateDragImage method.
      TreeView1.DragIcon = TreeView1.SelectedItem.CreateDragImage
      TreeView1.Drag vbBeginDrag ' Drag operation.
   End If
End Sub

Private Sub TreeView1_DragDrop_
(Source As Control, x As Single, y As Single)
   If TreeView1.DropHighlight Is Nothing Then
      Set TreeView1.DropHighlight = Nothing
      indrag = False
      Exit Sub
   Else
      If nodX = TreeView1.DropHighlight Then Exit Sub
      Cls
      Print nodX.Text & " dropped on " & TreeView1.DropHighlight.Text
      Set TreeView1.DropHighlight = Nothing
      indrag = False
   End If
End Sub

Private Sub TreeView1_DragOver(Source As Control, x As Single, y As Single,
State As Integer)
   If indrag = True Then
      ' Set DropHighlight to the mouse's coordinates.
      Set TreeView1.DropHighlight = TreeView1.HitTest(x, y)
   End If
End Sub
```

## EnsureVisible Method

See Also Example

Ensures that a specified **ListItem** or **Node** object is visible. If necessary, this method scrolls and expands the **TreeView** or **ListView** control.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**EnsureVisible**

The *object* placeholder represents an object expression that evaluates to a **ListItem** or **Node** object.

### Return Values

| Value | Description |
|---|---|
| **True** | The method returns **True** if the **ListView** or **TreeView** control must scroll and/or expand to expose the object. |
| **False** | The method returns **False** if no scrolling and/or expansion is required. |

### Remarks

Use the **EnsureVisible** method when you want a particular **Node** or **ListItem** object, which might be hidden deep in a **TreeView** or **ListView** control, to be visible.

**See Also**

Collapse Event
Expand Event
**Expanded** Property
**ListItem** Object**, ListItems** Collection
**ListView** Control
**Node** Object, **Nodes** Collection
**TreeView** Control

■

## EnsureVisible Method Example

This example adds many nodes to a **TreeView** control, and uses the **EnsureVisible** method to scroll and expand the tree.   To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section.  Run the example, and click the form to see the **TreeView** expand.

```
Private Sub Form_Load()
   Dim nodX As Node
   Dim i as Integer
   TreeView1.BorderStyle = FixedSingle ' Show borders.

   Set nodX = TreeView1.Nodes.Add(,,,"Root")   Add first node.
   For i = 1 to 15                   ' Add 15 nodes
      Set nodX = TreeView1.Nodes.Add(i,,,"Node " & CStr(i))
   Next i

   Set nodX = TreeView1.Nodes.Add(,,,"Bottom") ' Add one with text.
   Set nodX = TreeView1.Nodes.Add(i,,,"Expanded") ' Add child to node.
   Set nodX = TreeView1.Nodes.Add(i+1,,,"Show me") ' Add a final child.
End Sub

Private Sub Form_Click()
   ' Tree will scroll and expand when you click the form.
   TreeView1.Nodes(TreeView1.Nodes.Count).EnsureVisible
End Sub
```

## Expand Event (TreeView Control)

See Also Example

Occurs when a **Node** object in a **TreeView** control is expanded; that is, when its child nodes become visible.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

**Sub** *object*_**Expand(ByVal** *node* **As Node)**

The Expand event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TreeView** control. |
| *node* | A reference to the expanded **Node** object. |

### Remarks

The Expand event occurs after the Click and DblClick events.

The Expand event is generated in three ways: when the user double-clicks a **Node** object that has child nodes; when the **Expanded** property for a **Node** object is set to **True**; and when the plus/minus image is clicked.   Use the Expand event to validate an object, as in the following example:

```
Private Sub TreeView1_Expand(ByVal Node As Node)
   If Node.Index <> 1 Then
      Node.Expanded = False        ' Prevent expand.
   End If
End Sub
```

**See Also**

Collapse Event
**Expanded** Property
**Node** Object, **Nodes** Collection
**TreeView** Control

■

## Expand Event Example

This example adds several **Node** objects to a **TreeView** control.   When a **Node** is expanded, the Expand event is generated, and information about the **Node** is displayed.   To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section.  Run the example, and expand the nodes.

```
Private Sub Form_Load()
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(, , "RP", "Root Parent")
   Set nodX = TreeView1.Nodes.Add("RP", tvwChild, "C1", "Child1")
   Set nodX = TreeView1.Nodes.Add("C1", tvwChild, "C2", "Child2")
   Set nodX = TreeView1.Nodes.Add("C2", tvwChild, "C3", " Child3")
   Set nodX = TreeView1.Nodes.Add("C2", tvwChild, "C4", " Child4")
   TreeView1.Style = tvwTreelinesPlusMinusText   ' Style 6.
   TreeView1.LineStyle = tvwRootLines            ' Style 1
End Sub

Private Sub TreeView1_Expand(ByVal Node As Node)
   Select Case Node.Key Like "C*"
   Case Is = True
      MsgBox Node.Text & " is a child node."
   End Select
End Sub
```

## Expanded Property

See Also Example

Returns or sets a value that determines whether a **Node** object in a **TreeView** control is currently expanded or collapsed.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Expanded**[= *boolean*]

The **Expanded** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Node** object. |
| *boolean* | A Boolean expression specifying whether the node is expanded or collapsed. |

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | The **Node** is currently expanded. |
| **False** | The **Node** is currently collapsed. |

### Remarks

You can use the **Expanded** property to programmatically expand a **Node** object. The following code has the same effect as double-clicking the first **Node**:

```
TreeView1.Nodes(1).Expanded = True
```

When a **Node** object is expanded, the Expand event is generated.

If a **Node** object has no child nodes, the property value is ignored.

**See Also**

**EnsureVisible** Method
Expand Event
**Node** Object, **Nodes** Collection
**TreeView** Control

■

## Expanded Property Example

This example adds several **Node** objects to a **TreeView** control. When you click the form, the **Expanded** property for each **Node** is set to **True**. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, and click the form to expand all the **Node** objects.

```
Private Sub Form_Load()
   Dim nodX As Node
   Dim i as Integer
   TreeView1.BorderStyle = vbFixedSingle ' Show border.

   ' Create a root node.
   Set nodX = TreeView1.Nodes.Add(,,"root","Root")

   For i = 1 to 5                    ' Add 5 child nodes.
      Set nodX = TreeView1.Nodes.Add(i,tvwChild,,"Node " & CStr(i))
   Next i
End Sub

Private Sub Form_Click()
   Dim I as Integer
   For I = 1 to TreeView1.Nodes.Count
      ' Expand all nodes.
      TreeView1.Nodes(i).Expanded = True
   Next I
End Sub
```

## ExpandedImage Property

See Also

Returns or sets the index or key value of a **ListImage** object in an associated **ImageList** control; the **ListImage** is displayed when a **Node** object is expanded.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**ExpandedImage**[ = *number*]

The **ExpandedImage** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Node** object. |
| *number* | A numeric expression that specifies the index of the image to be displayed. |

### Remarks

This property allows you to change the image associated with a **Node** object when the user double-clicks the node or when the **Node** object's **Expanded** property is set to **True**.

**See Also**

**Expanded** Property
Expand Event
**Image** Property
**ImageList** Control
**ImageList** Property
**SelectedImage** Property
**TreeView** Control

## FirstSibling Property

See Also Example

Returns a reference to the first sibling of a **Node** object in a **TreeView** control.

---

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**FirstSibling**

The *object* placeholder represents an object expression that evaluates to a **Node** object.

### Remarks

The first sibling is the **Node** that appears in the first position in one level of a hierarchy of nodes. Which **Node** actually appears in the first position depends on whether or not the **Node** objects at that level are sorted, which is determined by the **Sorted** property. To sort **Node** objects, set the **Sorted** property of the **Parent** node to **True**, as follows:

```
Private Sub TreeView1_NodeClick(ByVal Node As Node)
   Node.Parent.Sorted = True
End Sub
```

The **Child**, **FirstSibling**, **LastSibling**, **Previous**, **Parent**, **Next**, and **Root** properties all return a reference to another **Node** object. Therefore you can simultaneously reference and perform operations on a **Node**, as follows:

```
With TreeView1.Nodes(x).Child
   .Text = "New text"
   .Key = "New key"
   .SelectedImage = 3
End With
```

You can also set an object variable to the referenced **Node**, as follows:

```
Dim NodChild As Node
' Get a reference to the child of Node x.
Set NodChild = TreeView1.Nodes(x).Child
' Use this reference to perform operations on the child Node.
With nodChild
   .Text = "New text"      '  Change the text.
   .Key = "New key"        ' Change key.
   .SelectedImage = 3      ' Change SelectedImage.
End With
```

**See Also**

**Child** Property
**LastSibling** Property
**Node** Object, **Nodes** Collection
**Parent** Property
**Sorted** Property (**TreeView**)
**TreeView** Control

▪

## FirstSibling Property Example

This example adds several nodes to a **TreeView** control. The **FirstSibling** property, in conjunction with the **Next** property and the **LastSibling** property, is used to navigate through a clicked **Node** object's hierarchy level. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example and click the various nodes to see what is returned.

```
Private Sub Form_Load()
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,"dad","Mike") ' A first sibling.
   Set nodX = TreeView1.Nodes.Add(,,"mom","Carol")
   Set nodX = TreeView1.Nodes.Add(,,,"Alice")

   ' Marsha is the FirstSibling.
   Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Marsha")
   Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Jan")
   Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Cindy")
   nodX.EnsureVisible ' Show all nodes.

   ' Greg is the FirstSibling.
   Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Greg")
   Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Peter")
   Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Bobby")
   nodX.EnsureVisible ' Show all nodes.
End Sub


Private Sub TreeView1_NodeClick(ByVal Node As Node)
   Dim strText As String
   Dim n As Integer
   ' Set n to FirstSibling's index.
   n = Node.FirstSibling.Index
   ' Place FirstSibling's text & linefeed in string variable.
   strText = Node.FirstSibling.Text & Chr(10)
   While n <> Node.LastSibling.Index
   ' While n is not the index of the last sibling, go to the
   ' next sibling and place its text into the string variable.
      strText = strText & TreeView1.Nodes(n).Next.Text & Chr(10)
   ' Set n to the next node's index.
      n = TreeView1.Nodes(n).Next.Index
   Wend
   MsgBox strText      ' Display results.
End Sub
```

## FullPath Property (TreeView Control)

See Also Example

Returns the fully qualified path of the currently selected **Node** object in a **TreeView** control.

---

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**FullPath**

The *object* placeholder represents an object expression that evaluates to a **Node** object.

### Remarks

The fully qualified name is the concatenation of the text in the selected **Node** object's **Text** property with the **Text** property values of all its ancestors. The value of the **PathSeparator** property determines the delimiter.

**See Also**

**Node** Object, **Nodes** Collection
**PathSeparator** Property
**TreeView** Control

▪
## FullPath Property Example

This example adds several **Node** objects to a **TreeView** control and displays the fully qualified path of each when selected.   To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section.   Run the example, then select a node and click the form to display the **Node** object's full path.

```
Private Sub Form_Load()
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,,"Root")
   Set nodX = TreeView1.Nodes.Add(1,tvwChild,,"Dir1")
   Set nodX = TreeView1.Nodes.Add(2,tvwChild,,"Dir2")
   Set nodX = TreeView1.Nodes.Add(3,tvwChild,,"Dir3")
   Set nodX = TreeView1.Nodes.Add(4,tvwChild,,"Dir4")
   nodX.EnsureVisible              ' Show all nodes.
   TreeView1.Style = tvwTreelinesText ' Style 4.
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
   MsgBox Node.FullPath
End Sub
```

## GetVisibleCount Method

See Also Example

Returns the number of **Node** objects that fit in the internal area of a **TreeView** control.

---

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**GetVisibleCount**

The *object* placeholder represents an object expression that evaluates to a **TreeView** control.

### Remarks

The number of **Node** objects is determined by how many lines can fit in a window. The total number of lines possible is determined by the height of the control and the **Size** property of the **Font** object. The count includes the partially visible item at the bottom of the list.

You can use the **GetVisibleCount** property to make sure that a minimum number of lines are visible so the user can accurately assess a hierarchy. If the minimum number of lines is not visible, you can reset the size of the **TreeView** using the **Height** property.

If a particular **Node** object must be visible, use the **EnsureVisible** method to scroll and expand the **TreeView** control.

**See Also**

**EnsureVisible** Method
**Height** Property
**Node** Object, **Nodes** Collection
**TreeView** Control

■

## GetVisibleCount Method Example

This example adds several **Node** objects to a **TreeView** control.   When you click the form, the code uses the **GetVisibleCount** method to check how many lines are visible, and then enlarges the control to show all the objects.   To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section.   Run the example, and click the form to enlarge the control.

```
Private Sub Form_Load()
   Dim nodX As Node
   Dim i as Integer
   TreeView1.BorderStyle = 1 ' Show border.
   For i = 1 to 20
      Set nodX = TreeView1.Nodes.Add(,,,"Node " & CStr(i))
   Next I
   TreeView1.Height = 1500 ' TreeView is short, for comparison's sake.
End Sub

Private Sub Form_Click()
   While Treeview1.GetVisibleCount < 20
      ' Make the treeview larger.
      TreeView1.Height = TreeView1.Height + TreeView1.Font.Size
   Wend
End Sub
```

## HitTest Method (ListView, TreeView Controls)

See Also    Example

Returns a reference to the **ListItem** object or **Node** object located at the coordinates of x and y.   Most often used with drag-and-drop operations to determine if a drop target item is available at the present location.

**Important**    This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**HitTest (***x* **As Single,** *y* **As Single)**

The **HitTest** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TreeView** or **ListView** control. |
| *x,y* | Coordinates of a target object, which is either a **Node** object or a **ListItem** object. |

### Remarks

If no object exists at the specified coordinates, the **HitTest** method returns **Nothing**.

The **HitTest** method is most frequently used with the **DropHighlight** property to highlight an object as the mouse is dragged over it.   The **DropHighlight** property requires a reference to a specific object that is to be highlighted.   In order to determine that object, the **HitTest** method is used in combination with an event that returns x and y coordinates, such as the DragOver event, as follows:

```
Private Sub TreeView1_DragOver _
(Source As Control, X As Single, Y As Single, State As Integer)
   Set TreeView1.DropHighlight = TreeView1.HitTest(X,Y)
End Sub
```

Subsequently, you can use the **DropHighlight** property in the DragDrop event to return a reference to the last object the source control was dropped over, as shown in the following code:

```
Private Sub TreeView1_DragDrop _
(Source As Control, x As Single, y As Single)
   ' DropHighlight returns a reference to object drop occurred over.
   Me.Caption = TreeView1.DropHighlight.Text
   ' To release the DropHighlight reference, set it to Nothing.
   Set TreeView1.DropHighlight = Nothing
End Sub
```

Note in the preceding example that the **DropHighlight** property is set to **Nothing** after the procedure is completed.   This must be done to release the highlight effect.

**See Also**

DragDrop Event
DragOver Event
**DropHighLight** Property
**ListItem** Object**, ListItems** Collection
**ListView** Control
**Node** Object, **Nodes** Collection
**TreeView** Control

▪
## HitTest Method (ListView, TreeView Controls) Example

This example adds several **Node** objects to a **TreeView** control. After you select a **Node** object, you can drag it to any other **Node**.   To try the example, place **TreeView** and **ImageList** controls on a form and paste the code into the form's Declaration section. Run the example and drag **Node** objects around to see the result.

```
' Declare global variables.
Dim indrag As Boolean ' Flag that signals a Drag Drop operation.
Dim nodX As Object ' Item that is being dragged.

Private Sub Form_Load()
   ' Load a bitmap into an Imagelist control.
   Dim imgX As ListImage
   Dim BitmapPath As String
   BitmapPath = "icons\mail\mail01a.ico"
   Set imgX = imagelist1.ListImages.Add(, , LoadPicture(BitmapPath))

   ' Initialize TreeView control and create several nodes.
   TreeView1.ImageList = imagelist1
   Dim nodX As Node  ' Create a tree.
   Set nodX = TreeView1.Nodes.Add(, , , "Parent1", 1)
   Set nodX = TreeView1.Nodes.Add(, , , "Parent2", 1)
   Set nodX = TreeView1.Nodes.Add(1, tvwChild, , "Child 1", 1)
   Set nodX = TreeView1.Nodes.Add(1, tvwChild, , "Child 2", 1)
   Set nodX = TreeView1.Nodes.Add(2, tvwChild, , "Child 3", 1)
   Set nodX = TreeView1.Nodes.Add(2, tvwChild, , "Child 4", 1)
   Set nodX = TreeView1.Nodes.Add(3, tvwChild, , "Child 5", 1)
   nodX.EnsureVisible ' Expand tree to show all nodes.
End Sub

Private Sub TreeView1_MouseDown_
(Button As Integer, Shift As Integer, x As Single, y As Single)
   Set nodX = TreeView1.SelectedItem ' Set the item being dragged.
End Sub

Private Sub TreeView1_MouseMove _
(Button As Integer, Shift As Integer, x As Single, y As Single)
   If Button = vbLeftButton Then ' Signal a Drag operation.
      indrag = True ' Set the flag to true.
      ' Set the drag icon with the CreateDragImage method.
      TreeView1.DragIcon = TreeView1.SelectedItem.CreateDragImage
      TreeView1.Drag vbBeginDrag ' Drag operation.
   End If
End Sub

Private Sub TreeView1_DragDrop_
(Source As Control, x As Single, y As Single)
   If TreeView1.DropHighlight Is Nothing Then
      Set TreeView1.DropHighlight = Nothing
      indrag = False
      Exit Sub
   Else
      If nodX = TreeView1.DropHighlight Then Exit Sub
      Cls
      Print nodX.Text & " dropped on " & TreeView1.DropHighlight.Text
      Set TreeView1.DropHighlight = Nothing
      indrag = False
   End If
End Sub

Private Sub TreeView1_DragOver(Source As Control, x As Single, y As Single,
State As Integer)
   If indrag = True Then
      ' Set DropHighlight to the mouse's coordinates.
      Set TreeView1.DropHighlight = TreeView1.HitTest(x, y)
   End If
End Sub
```

## Indentation Property

See Also    Example

Returns or sets the width of the indentation for a **TreeView** control.   Each new child **Node** object is indented by this amount.

---

**Important**    This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**Indentation**[ = *number*]

The **Indentation** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TreeView** control. |
| *number* | An integer specifying the width that each child **Node** is indented. |

### Remarks

If you change the **Indentation** property at run time, the **TreeView** is redrawn to reflect the new width. This property uses the scale mode of its container.   The property value cannot be negative.

**See Also**

**Node** Object, **Nodes** Collection
**TreeView** Control

▪

## Indentation Property Example

This example adds several **Node** objects to a **TreeView** control, while the **Indentation** property is shown in the form's caption.   A **ComboBox** control provides alternate values for the **Indentation** width.   To try the example, place a **TreeView** control and a **ComboBox** control on a form, and paste the code into the form's Declarations section.   Run the example, and use the **ComboBox** to change the **Indentation** property.

```
Private Sub Form_Load()
   With combo1 ' Populate ComboBox with alternate values.
   .AddItem "250"
   .AddItem "550"
   .AddItem "1000"
   .ListIndex = 1
   End With

   Dim nodX As Node
   Dim i As Integer

   Set nodX = TreeView1.Nodes.Add(,,,CStr(1)) ' Add first node.

   For i = 1 To 6 ' Add 6 nodes.
      Set nodX = TreeView1.Nodes.Add(i,tvwChild,,CStr(i + 1))
   Next i

   nodX.EnsureVisible   ' Makes sure all nodes are visible.
   Form1.Caption = "Indentation = " & TreeView1.Indentation
End Sub

Sub combo1_Click()   ' Change Indentation with ComboBox value.
   TreeView1.Indentation = combo1.Text
   Form1.Caption = "Indentation = " & TreeView1.Indentation
End Sub
```

## LabelEdit Property

See Also Example

Returns or sets a value that determines if a user can edit labels of **ListItem** or **Node** objects in a **ListView** or **TreeView** control.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**LabelEdit** [ = *integer*]

The **LabelEdit** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TreeView** or **ListView** control. |
| *integer* | An integer that determines whether the label of a **Node** or **ListItem** object can be edited, as specified in Settings. |

### Settings

The settings for *integer* are:

| Constant | Value | Description |
|---|---|---|
| **lvwAutomatic** | 0 | (Default) Automatic. The BeforeLabelEdit event is generated when the user clicks the label of a selected node. |
| **lvwManual** | 1 | Manual. The BeforeLabelEdit event is generated only when the **StartLabelEdit** method is invoked. |

**Note** The constants above are for the **ListView** control. The constants for the **TreeView** control are: **tvwAutomatic** and **tvwManual**.

### Remarks

Label editing of an object is initiated when a selected object is clicked. That is, the first click on an object will select it; a second (single) click on the object will initiate the label editing operation.

The **LabelEdit** property, in combination with the **StartLabelEdit** method, allows you to programmatically determine when and which labels can be edited. When the **LabelEdit** property is set to 1, no label can be edited unless the **StartLabelEdit** method is invoked. For example, the following code allows the user to edit a **Node** object's label by clicking a Command button:

```
Private Sub Command1_Click()
   ' Determine if the right Node is selected.
   If TreeView1.SelectedItem.Index = 1 Then
      TreeView1.StartLabelEdit ' Let user begin editing.
   End If
End Sub
```

**See Also**

AfterLabelEdit Event
BeforeLabelEdit Event
**ListItem** Object, **ListItems** Collection
**ListView** Control
**ListView** Control Constants
**Node** Object, **Nodes** Collection
**StartLabelEdit** Method
**TreeView** Control Constants
**TreeView** Control

■

## LabelEdit Property Example

This example initiates label editing when you click the Command button. It allows a **Node** object to be edited unless it is a root **Node**. The **LabelEdit** property must be set to **Manual**. To try the example, place a **TreeView** control and a CommandButton on a form. Paste the code into the form's Declarations section. Run the example, select a node to edit, and press the Command button.

```
Private Sub Form_Load()
   Dim nodX As Node
   Dim i As Integer
   TreeView1.LabelEdit = tvwManual   ' Set property to manual.
   Set nodX = TreeView1.Nodes.Add(,,," Node 1")' Add first node.

   For i = 1 to 5                     ' Add 5 nodes.
      Set nodX = TreeView1.Nodes.Add(i,tvwChild,,"Node " & CStr(i + 1))
   Next I

   nodX.EnsureVisible                 ' Show all nodes.
End Sub

Private Sub Command1_Click()
   ' Invoke the StartLabelEdit method on the selected node,
   ' which triggers the BeforeLabelEdit event.
   TreeView1.StartLabelEdit
End Sub

Private Sub TreeView_BeforeLabelEdit (Cancel as Integer)
   ' If the selected item is the root, then cancel the edit.
   If TreeView1.SelectedItem Is TreeView1.SelectedItem.Root Then
      Cancel = True
   End If
End Sub

Private Sub TreeView_AfterLabelEdit _
(Cancel As Integer, NewString As String)
   ' Assume user has entered some text and pressed the ENTER key.
   ' Any nonempty string will be valid.
   If Len(NewString) = 0 Then
      Cancel = True
   End If
End Sub
```

## LastSibling Property

See Also Example

Returns a reference to the last sibling of a **Node** object in a **TreeView** control.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**LastSibling**

The *object* placeholder represents an object expression that evaluates to a **Node** object.

### Remarks

The last sibling is the **Node** that appears in the last position in one level of a hierarchy of nodes. Which **Node** actually appears in the last position depends on whether or not the **Node** objects at that level are sorted, which is determined by the **Sorted** property. To sort the **Node** objects at one level, set the **Sorted** property of the **Parent** node to **True**. The following code demonstrates this:

```
Private Sub TreeView1_NodeClick(ByVal Node As Node)
   Node.Parent.Sorted = True
End Sub
```

The **Child**, **FirstSibling**, **LastSibling**, **Previous**, **Parent**, **Next**, and **Root** properties all return a reference to another **Node** object. Therefore, you can simultaneously reference and perform operations on a **Node**, as follows:

```
With TreeView1.Nodes(x).Child
   .Text = "New text"
   .Key = "New key"
   .SelectedImage = 3
End With
```

You can also set an object variable to the referenced **Node**, as follows:

```
Dim NodChild As Node
' Get a reference to the child of Node x.
Set NodChild = TreeView1.Nodes(x).Child
' Use this reference to perform operations on the child Node.
With nodChild
   .Text = "New text"      '  Change the text.
   .Key = "New key"        ' Change key.
   .SelectedImage = 3      ' Change SelectedImage.
End With
```

**See Also**

**FirstSibling** Property

**Next** Property

**Node** Object, **Nodes** Collection

**TreeView** Control

■

## LastSibling Property Example

This example adds several **Node** objects to a **TreeView** control.   The **LastSibling** property, in conjunction with the **Next** property and the **FirstSibling** property, is used to navigate through a clicked **Node** object's hierarchy level.   To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section.   Run the example, and click the various nodes to see what is returned.

```
Private Sub Form_Load()
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,"dad","Mike")
   Set nodX = TreeView1.Nodes.Add(,,"mom","Carol")
   ' Alice is the LastSibling.
   Set nodX = TreeView1.Nodes.Add(,,,"Alice")

   Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Marsha")
   Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Jan")
   ' Cindy is the LastSibling.
   Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Cindy")
   nodX.EnsureVisible ' Show all nodes.

   Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Greg")
   Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Peter")
   ' Bobby is the LastSibling.
   Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Bobby")
   nodX.EnsureVisible ' Show all nodes.
End Sub


Private Sub TreeView1_NodeClick(ByVal Node As Node)
   Dim strText As String
   Dim n As Integer
   ' Set n to FirstSibling's index.
   n = Node.FirstSibling.Index
   ' Place FirstSibling's text & linefeed in string variable.
   strText = Node.FirstSibling.Text & Chr(10)
   While n <> Node.LastSibling.Index
   ' While n is not the index of the last sibling, go to the
   ' next sibling and place its text into the string variable
      strText = strText & TreeView1.Nodes(n).Next.Text & Chr(10)
   ' Set n to the next node's index.
      n = TreeView1.Nodes(n).Next.Index
   Wend
   MsgBox strText ' Display results.
End Sub
```

## LineStyle Property

See Also Example

Returns or sets the style of lines displayed between **Node** objects.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**LineStyle** [ = *number*]

The **LineStyle** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TreeView** control. |
| *number* | A value or constant that specifies the line style as shown in Settings. |

### Settings

The settings for *number* are:

| Constant | Value | Description |
|---|---|---|
| **tvwTreeLines** | 0 | (Default) Tree lines. Displays lines between **Node** siblings and their parent **Node**. |
| **tvwRootLines** | 1 | Root Lines. In addition to displaying lines between **Node** siblings and their parent **Node**, also displays lines between the root nodes. |

### Remarks

You must set the **Style** property to a style that includes tree lines.

**See Also**

**Style** Property (**TreeView** Control)
**TreeView** Control
**TreeView** Control Constants

▪

## LineStyle Property Example

This example adds several **Node** objects with images to a **TreeView** control. You can change the **LineStyle** and **Style** properties by selecting the alternate styles in two **ComboBox** controls. To try the example, place a **TreeView** control, an **ImageList** control, and two **ComboBox** controls on a form, and paste the code into the form's Declarations section. Run the example, and click either **ComboBox** to change the **LineStyle** and **Style** properties.

```
Private Sub Form_Load()
   ' Add an image to the ImageList control.
   Dim imgX As ListImage
   Set imgX = ImageList1.ListImages. _
   Add(,,LoadPicture("bitmaps\outline\leaf.bmp"))

   TreeView1.ImageList = ImageList1 ' Initialize ImageList.

   With combo1        ' Populate ComboBox with line styles.
   .AddItem "Tree lines"
   .AddItem "Root lines"
   .ListIndex = 0     ' The default is TreeLines.
   End With

   With Combo2        ' Populate ComboBox with all styles.
   .AddItem "Text only"                          ' 0
   .AddItem "Image & text"                       ' 1
   .AddItem "Plus/minus & text"                  ' 2
   .AddItem "Plus/minus, image & text"           ' 3
   .AddItem "Lines & text"                       ' 4
   .AddItem "Lines, image & Text"                ' 5
   .AddItem "Lines, plus/minus & Text"           ' 6
   .AddItem "Lines, plus/minus, image & text"    ' 7
   .ListIndex = 7
   End With


   Dim nodX As Node
   Dim i as Integer
   ' Create root node.
   Set nodX = TreeView1.Nodes.Add(,,,"Node " & "1",1)

   For i = 1 to 5          ' Add 5 nodes.
      Set nodX = TreeView1.Nodes. _
      Add(i,tvwChild,,"Node " & CStr(i + 1),1)
   Next I
   nodX.EnsureVisible      ' Show all nodes.
End Sub

Private Sub combo1_Click()
   ' Change line style from ComboBox.
   TreeView1.LineStyle = combo1.ListIndex
End Sub

Sub combo2_Click()         ' Change Style with ComboBox.
   TreeView1.Style = Combo2.ListIndex
   Form1.Caption = "Indentation = " & Combo1.Text
End Sub
```

## Next Property

See Also  Example

Returns a reference to the next sibling **Node** of a **TreeView** control's **Node** object.

---

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**Next**

The *object* placeholder represents an object expression that evaluates to a **Node** object.

### Remarks

The **Child**, **FirstSibling**, **LastSibling**, **Previous**, **Parent**, **Next**, and **Root** properties all return a reference to another **Node** object. Therefore you can simultaneously reference and perform operations on a **Node**, as follows:

```
With TreeView1.Nodes(x).Child
   .Text = "New text"
   .Key = "New key"
   .SelectedImage = 3
End With
```

You can also set an object variable to the referenced **Node**, as follows:

```
Dim NodChild As Node
' Get a reference to the child of Node x.
Set NodChild = TreeView1.Nodes(x).Child
' Use this reference to perform operations on the child Node.
With nodChild
   .Text = "New text"      '  Change the text.
   .Key = "New key"        ' Change key.
   .SelectedImage = 3      ' Change SelectedImage.
End With
```

**See Also**

**FirstSibling** Property
**LastSibling** Property
**Previous** Property
**TreeView** Control

▪

## Next Property Example

This example adds several **Node** objects to a **TreeView** control. The **LastSibling** property, in conjunction with the **Next** property and the **FirstSibling** property, is used to navigate through a clicked **Node** object's hierarchy level. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, and click the various nodes to see what is returned.

```
Private Sub Form_Load()
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,"dad","Mike")
   Set nodX = TreeView1.Nodes.Add(,,"mom","Carol")
   ' Alice is the LastSibling.
   Set nodX = TreeView1.Nodes.Add(,,,"Alice")

   Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Marsha")
   Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Jan")
   ' Cindy is the LastSibling.
   Set nodX = TreeView1.Nodes.Add("mom",tvwChild,,"Cindy")
   nodX.EnsureVisible ' Show all nodes.

   Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Greg")
   Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Peter")
   ' Bobby is the LastSibling.
   Set nodX = TreeView1.Nodes.Add("dad",tvwChild,,"Bobby")
   nodX.EnsureVisible ' Show all nodes.
End Sub


Private Sub TreeView1_NodeClick(ByVal Node As Node)
   Dim strText As String
   Dim n As Integer
   ' Set n to FirstSibling's index.
   n = Node.FirstSibling.Index
   ' Place FirstSibling's text & linefeed in string variable.
   strText = Node.FirstSibling.Text & Chr(10)
   ' While n is not the index of the last sibling, go to the
   ' next sibling and place its text into the string variable.
   While n <> Node.LastSibling.Index
      strText = strText & TreeView1.Nodes(n).Next.Text & Chr(10)
   ' Set n to the next node's index.
      n = TreeView1.Nodes(n).Next.Index
   Wend
   MsgBox strText ' Display results.
End Sub
```

## NodeClick Event

See Also Example

Occurs when a **Node** object is clicked.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

**Private Sub** *object*_**NodeClick(ByVal** *node* **As Node)**

The NodeClick event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TreeView** control. |
| *node* | A reference to the clicked **Node** object. |

### Remarks

The standard Click event is generated when the user clicks any part of the **TreeView** control. The NodeClick event is generated when the user clicks a particular **Node** object; the NodeClick event also returns a reference to a particular **Node** object which can be used to validate the **Node** before further action.

The NodeClick event occurs before the standard Click event.

**See Also**

**Node** Object, **Nodes** Collection
**SelectedItem** Property
**TreeView** Control

■

## NodeClick Event Example

This example adds several **Node** objects to a **TreeView** control. When a **Node** is clicked, the NodeClick event is triggered and is used to get the **Node** object's index and text. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, and click any **Node**.

```
Private Sub Form_Load()
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,"R","Root")
   nodX.Expanded = True
   Set nodX = TreeView1.Nodes.Add(,,"P","Parent")
   nodX.Expanded = True
   Set nodX = TreeView1.Nodes.Add("R",tvwChild,,"Child 1")
   Set nodX = TreeView1.Nodes.Add("R",tvwChild,,"Child 2")
   Set nodX = TreeView1.Nodes.Add("R",tvwChild,,"Child 3")
   Set nodX = TreeView1.Nodes.Add("P",tvwChild,,"Child 4")
   Set nodX = TreeView1.Nodes.Add("P",tvwChild,,"Child 5")
   Set nodX = TreeView1.Nodes.Add("P",tvwChild,,"Child 6")
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
   Form1.Caption = "Index = " & Node.Index & " Text:" & Node.Text
End Sub
```

## Nodes Property

See Also Example

Returns a reference to a collection of **TreeView** control **Node** objects.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Nodes**

The *object* placeholder represents an object expression that evaluates to a **TreeView** control.

### Remarks

You can manipulate **Node** objects using standard collection methods (for example, the **Add** and **Remove** methods). Each element in the collection can be accessed by its index, or unique key which you store in the **Key** property.

**See Also**

**Add** Method (**Nodes** Collection)
**Clear** Method
**Index** Property
**Item** Method
**Key** Property
**Node** Object, **Nodes** Collection
**Remove** Method
**TreeView** Control

■

## Nodes Property Example

This example adds several **Node** objects to a **TreeView** control.   When the form is clicked, a reference to each **Node** is used to display each **Node** object's text.   To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section.   Run the example, and click the form.Private Sub Form_Load()

```
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,"R","Root")
   Set nodX = TreeView1.Nodes.Add("R", tvwChild,"C1","Child 1")
   Set nodX = TreeView1.Nodes.Add("R", tvwChild,"C2","Child 2")
   Set nodX = TreeView1.Nodes.Add("R", tvwChild,"C3","Child 3")
   Set nodX = TreeView1.Nodes.Add("R", tvwChild,"C4","Child 4")
   nodX.EnsureVisible
   TreeView1.Style = tvwTreelinesText ' Style 4.
   TreeView1.BorderStyle = vbFixedSingle
End Sub

Private Sub Form_Click()
   Dim i As Integer
   Dim strNodes As String
   For i = 1 To TreeView1.Nodes.Count
   strNodes = strNodes & TreeView1.Nodes(i).Index & " " & _
   "Key: " & TreeView1.Nodes(i).Key & " " & _
   "Text: " & TreeView1.Nodes(i).Text & Chr(10)
   Next i
   MsgBox strNodes
End Sub
```

## Parent Property (Node Object)

See Also Example

Returns or sets the parent object of a **Node** object. Available only at run time.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Parent**[ = *node*]

The **Parent** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Node** object. |
| *node* | A **Node** object that becomes the parent of the object. |

### Remarks

At run time, an error occurs if you set this property to an object that creates a loop. For example, you cannot set any **Node** to become a child **Node** of its own descendants.

The **Child**, **FirstSibling**, **LastSibling**, **Previous**, **Parent**, **Next**, and **Root** properties all return a reference to another **Node** object. Therefore, you can simultaneously reference and perform operations on a **Node**, as follows:

```
With TreeView1.Nodes(x).Child
   .Text = "New text"
   .Key = "New key"
   .SelectedImage = 3
End With
```

You can also set an object variable to the referenced **Node**, as follows:

```
Dim NodChild As Node
' Get a reference to the child of Node x.
Set NodChild = TreeView1.Nodes(x).Child
' Use this reference to perform operations on the child Node.
With nodChild
   .Text = "New text"       '  Change the text.
   .Key = "New key"         ' Change key.
   .SelectedImage = 3       ' Change SelectedImage.
End With
```

**See Also**

**Child** Property
**FirstSibling** Property
**LastSibling** Property
**Next** Property
**Node** Object, **Nodes** Collection
**Previous** Property
**Root** Property

▪

**Parent Property Example (Node Object)**

This example adds several **Node** objects to a **TreeView** control. After you select a **Node** object, you can then click and drag it to any other **Node** to make it a child of the target **Node**. To try the example, place **TreeView** and **ImageList** controls on a form and paste the code into the form's Declaration section. Run the example and drag **Node** objects onto other **Node** objects to see the result.

```
' Declare global variables.
Dim indrag As Boolean ' Flag that signals a Drag Drop operation.
Dim nodX As Object ' Item that is being dragged.

Private Sub Form_Load()
   ' Load a bitmap into an Imagelist control.
   Dim imgX As ListImage
   Dim BitmapPath As String
   BitmapPath = "icons\mail\mail01a.ico"
   Set imgX = ImageList1.ListImages.Add(, , LoadPicture(BitmapPath))

   ' Initialize TreeView control and create several nodes.
   TreeView1.ImageList = ImageList1
   Dim nodX As Node     ' Create a tree.
   Set nodX = TreeView1.Nodes.Add(, , , "Parent1", 1)
   Set nodX = TreeView1.Nodes.Add(, , , "Parent2", 1)
   Set nodX = TreeView1.Nodes.Add(1, tvwChild, , "Child 1", 1)
   Set nodX = TreeView1.Nodes.Add(1, tvwChild, , "Child 2", 1)
   Set nodX = TreeView1.Nodes.Add(2, tvwChild, , "Child 3", 1)
   Set nodX = TreeView1.Nodes.Add(2, tvwChild, , "Child 4", 1)
   Set nodX = TreeView1.Nodes.Add(3, tvwChild, , "Child 5", 1)
   nodX.EnsureVisible ' Expand tree to show all nodes.
End Sub

Private Sub TreeView1_MouseDown(Button As Integer, Shift As Integer, x As
Single, y As Single)
   Set nodX = TreeView1.SelectedItem ' Set the item being dragged.
   Set TreeView1.DropHighlight = Nothing
End Sub

Private Sub TreeView1_MouseMove _
(Button As Integer, Shift As Integer, x As Single, y As Single)
   If Button = vbLeftButton Then ' Signal a Drag operation.
      indrag = True ' Set the flag to true.
      ' Set the drag icon with the CreateDragImage method.
      TreeView1.DragIcon = TreeView1.SelectedItem.CreateDragImage
      TreeView1.Drag vbBeginDrag ' Drag operation.
   End If
End Sub

Private Sub TreeView1_DragDrop(Source As Control, x As Single, y As Single)
   ' If user didn't move mouse or released it over an invalid area.
   If TreeView1.DropHighlight Is Nothing Then
      indrag = False
      Exit Sub
   Else
      ' Set dragged node's parent property to the target node.
      On Error GoTo checkerror ' To prevent circular errors.
      Set nodX.Parent = TreeView1.DropHighlight
      Cls
      Print TreeView1.DropHighlight.Text & _
      " is parent of " & nodX.Text
      ' Release the DropHighlight reference.
      Set TreeView1.DropHighlight = Nothing
      indrag = False
      Exit Sub ' Exit if no errors occured.
   End If

checkerror:
   ' Define constants to represent Visual Basic errors code.
   Const CircularError = 35614
   If Err.Number = CircularError Then
      Dim msg As String
      msg = "A node can't be made a child of its own children."
      ' Display the message box with an exclamation mark icon
      ' and with OK and Cancel buttons.
      If MsgBox(msg, vbExclamation & vbOKCancel) = vbOK Then
         ' Release the DropHighlight reference.
         indrag = False
         Set TreeView1.DropHighlight = Nothing
         Exit Sub
      End If
   End If
End Sub

Private Sub TreeView1_DragOver(Source As Control, x As Single, y As Single,
State As Integer)
   Set TreeView1.DropHighlight = TreeView1.HitTest(x, y)
End Sub
```

## PathSeparator Property (TreeView Control)

See Also  Example

Returns or sets the delimiter string used for the path returned by the **FullPath** property.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**PathSeparator** [ = *string*]

The **PathSeparator** syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TreeView** control. |
| *string* | A string that determines the **PathSeparator**, usually a single character. |

### Remarks

The default character is "\."

**See Also**

**FullPath** Property

**TreeView** Control

■

**PathSeparator Property Example**

This example adds several **Node** objects to a **TreeView** control, and uses a **ComboBox** control to change the **PathSeparator** property.   To try the example, place a **TreeView** control and a **ComboBox** on a form, and paste the code into the form's Declarations section.   Run the example, select a **Node**, and click the form.   Change the **PathSeparator** property value using the **ComboBox**.

```
Private Sub Form_Load
   TreeView1.BorderStyle = vbFixedSingle    ' Show border.
   With combo1     ' Populate the ComboBox with alternate characters.
   .AddItem "/"
   .AddItem "-"
   .AddItem ":"
   .ListIndex = 1
   End With

   Dim nodX As Node
   Dim i As Integer
   Set nodX = TreeView1.Nodes.Add(,,,CStr(1))   ' Add first node.

   For i = 1 to 5                     ' Add other nodes.
      Set nodX = TreeView1.Nodes.Add(i,tvwChild,,CStr(i + 1))
   Next i

   nodX.EnsureVisible                 ' Ensure all are visible.
End Sub

Private Sub combo1_Click()            ' Change the delimiter character.
   TreeView1.PathSeparator = combo1.Text
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
   ' Show path in form's caption.
   Me.Caption = Node.FullPath
End Sub
```

## Previous Property (Node Object)

See Also Example

Returns a reference to the previous sibling of a **Node** object.

---

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*. **Previous**

The *object* placeholder represents an object expression that evaluates to a **Node** object.

### Remarks

The **Child**, **FirstSibling**, **LastSibling**, **Previous**, **Parent**, **Next**, and **Root** properties all return a reference to another **Node** object. Therefore you can simultaneously reference and perform operations on a **Node**, as follows:

```
With TreeView1.Nodes(x).Child
   .Text = "New text"
   .Key = "New key"
   .SelectedImage = 3
End With
```

You can also set an object variable to the referenced **Node**, as follows:

```
Dim NodChild As Node
' Get a reference to the child of Node x.
Set NodChild = TreeView1.Nodes(x).Child
' Use this reference to perform operations on the child Node.
With nodChild
   .Text = "New text"        '  Change the text.
   .Key = "New key"          ' Change key.
   .SelectedImage = 3        ' Change SelectedImage.
End With
```

**See Also**

**FirstSibling** Property
**LastSibling** Property
**Node** Object, **Nodes** Collection
**Parent** Property

■

## Previous Property Example

This example adds several nodes to a **TreeView** control. The **Previous** property, in conjunction with the **LastSibling** property and the **FirstSibling** property, is used to navigate through a clicked **Node** object's hierarchy level. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, and click the various nodes to see what is returned.

```
Private Sub Form_Load()
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(, , "r", "Root")
   Set nodX = TreeView1.Nodes.Add(, , "p", "parent")

   Set nodX = TreeView1.Nodes.Add("r", tvwChild, , "Child 1")
   Set nodX = TreeView1.Nodes.Add("r", tvwChild, , "Child 2")
   Set nodX = TreeView1.Nodes.Add("r", tvwChild, , "Child 3")
   nodX.EnsureVisible ' Show all nodes.

   Set nodX = TreeView1.Nodes.Add("p", tvwChild, , "Child 4")
   Set nodX = TreeView1.Nodes.Add("p", tvwChild, , "Child 5")
   Set nodX = TreeView1.Nodes.Add("p", tvwChild, , "Child 6")
   nodX.EnsureVisible ' Show all nodes.
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
   Dim strText As String
   Dim n As Integer
   ' Set n to LastSibling's index.
   n = Node.LastSibling.Index
   ' Place LastSibling's text & linefeed in string variable.
   strText = Node.LastSibling.Text & Chr(10)
   While n <> Node.FirstSibling.Index
      ' While n is not the index of the FirstSibling, go to the
      ' previous sibling and place its text into the string variable.
      strText = strText & TreeView1.Nodes(n).Previous.Text & Chr(10)
      ' Set n to the previous node's index.
      n = TreeView1.Nodes(n).Previous.Index
   Wend
   MsgBox strText ' Display results.
End Sub
```

## Root Property (Node Object)

See Also Example

Returns a reference to the root **Node** object of a selected **Node**.

---

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**Root**

The *object* placeholder represents an object expression that evaluates to a **Node** object.

### Remarks

The **Child**, **FirstSibling**, **LastSibling**, **Previous**, **Parent**, **Next**, and **Root** properties all return a reference to another **Node** object. Therefore, you can simultaneously reference and perform operations on a **Node**, as follows:

```
With TreeView1.Nodes(x).Child
   .Text = "New text"
   .Key = "New key"
   .SelectedImage = 3
End With
```

You can also set an object variable to the referenced **Node**, as follows:

```
Dim NodChild As Node
' Get a reference to the child of Node x.
Set NodChild = TreeView1.Nodes(x).Child
' Use this reference to perform operations on the child Node.
With nodChild
   .Text = "New text"      '  Change the text.
   .Key = "New key"        ' Change key.
   .SelectedImage = 3      ' Change SelectedImage.
End With
```

**See Also**

**FirstSibling** Property
**LastSibling** Property
**Node** Object, **Nodes** Collection
**Parent** Property (**Node** Object)
**SelectedItem** Property

▪

## Root Property Example

This example adds several **Node** objects to a **TreeView** control.   When you click a **Node**, the code navigates up the tree to the **Root** node, and displays the text of each **Parent** node.   To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section.  Run the example, and click a **Node**.

```
Private Sub Form_Load()
   Dim nodX As Node                ' Create a tree.
   Set nodX = TreeView1.Nodes.Add(,,"r", "Root")
   Set nodX = TreeView1.Nodes.Add(,,"p", "Parent")
   Set nodX = TreeView1.Nodes.Add("p",tvwChild,, "Child 1")
   nodX.EnsureVisible              ' Show all nodes.
   Set nodX = TreeView1.Nodes.Add("r",tvwChild,"C2", "Child 2")
   Set nodX = TreeView1.Nodes.Add("C2",tvwChild,"C3", "Child 3")
   Set nodX = TreeView1.Nodes.Add("C3",tvwChild,, "Child 4")
   Set nodX = TreeView1.Nodes.Add("C3",tvwChild,, "Child 5")
   nodX.EnsureVisible              ' Show all nodes.
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
   Dim n As Integer
   Dim strParents As String        ' Variable for information.
   n = Node.Index                  ' Set n to index of clicked node.
   strParents = Node.Text & Chr(10)
   While n <> Node.Root.Index
      strParents = strParents & _
      TreeView1.Nodes(n).Parent.Text & Chr(10)
      ' Set n to index of next parent Node.
      n = TreeView1.Nodes(n).Parent.Index
   Wend
   MsgBox strParents
End Sub
```

## Selected Property (Custom Controls)

See Also Example

Returns or sets a value that determines if a **ListItem**, **Node**, or **Tab** object is selected.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Selected** [ = *boolean*]

The **Selected** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListItem**, **Node**, or **Tab** object. |
| *boolean* | A Boolean expression that determines if an object is selected. |

### Remarks

The **Selected** property is used to programmatically select a specific **Node, ListItem** or **Tab** object. Once you have selected an object in this manner, you can perform various operations on it, such as setting properties and invoking methods.

To select a specific **Node** object, you must refer to it either by the value of its **Index** property or its **Key** property. The following example selects a specific **Node** object in a **TreeView** control:

```
Private Sub Command1_Click()
   ListView1.ListItems(3).Selected = True ' Selects an object.
   ' Use the SelectedItem property to get a reference to the object.
   ListView1.SelectedItem.Text = "Changed Text"
End Sub
```

In the **ListView** control, the **SelectedItem** property always refers to the first selected item. Therefore, if multiple items are selected, you must iterate through all of the items, checking each item's **Selected** property.

**See Also**

**ListItem** Object, **ListItems** Collection
**ListView** Control
**Node** Object, **Nodes** Collection
**SelectedItem** Property
**TreeView** Control

▪

## Selected Property Example

This example adds several **Node** objects to a **TreeView** control. When a **Node** is selected, a reference to the selected **Node** is used to display its key. To try the example, place a **TreeView** control on a form, and paste the code into the form's Declarations section. Run the example, select a **Node**, and click the form.

```
Private Sub Form_Load()
   Dim nodX As Node            ' Create a tree.
   Set nodX = TreeView1.Nodes.Add(,,"r","Root")
   Set nodX = TreeView1.Nodes.Add(,,"p","Parent")
   Set nodX = TreeView1.Nodes.Add("p",tvwChild,,"Child 1")
   nodX.EnsureVisible          ' Show all nodes.
   Set nodX = TreeView1.Nodes.Add("r",tvwChild,"C2","Child 2")
   Set nodX = TreeView1.Nodes.Add("C2",tvwChild,"C3","Child 3")
   Set nodX = TreeView1.Nodes.Add("C3",tvwChild,,"Child 4")
   Set nodX = TreeView1.Nodes.Add("C3",tvwChild,,"Child 5")
   nodX.EnsureVisible          ' Show all nodes.
End Sub

Private Sub Form_Click()
   Dim intX As Integer
   On Error Resume Next        ' If an integer isn't entered.
   intX = InputBox("Check Node",,TreeView1.SelectedItem.Index)
   If IsNumeric(intX) Then     ' Ensure an integer was entered.
      If TreeView1.Nodes(intX).Selected = True Then
         MsgBox TreeView1.Nodes(intX).Text & " is selected."
      Else
         MsgBox "Not selected"
      End If
   End If
End Sub
```

The following example adds three **ListItem** objects to a **ListView** control. When you click the form, the code uses the **Selected** property to determine if a specific **ListItem** object is selected. To try the example, place a **ListView** control on a form and paste the code into the form's Declarations section. Run the example, select a **ListItem**, and click the form.

```
Private Sub Form_Load()
   Listview1.BorderStyle = vbFixedSingle  ' Show the border.
   Dim itmX As ListViewItem
   Set itmX = ListView1.ListItems.Add(,,"Item 1")
   Set itmX = ListView1.ListItems.Add(,,"Item 2")
   Set itmX = ListView1.ListItems.Add(,,"Item 3")
End Sub

Private Sub Form_Click()
   Dim intX As Integer
   On Error Resume Next ' If an integer isn't entered.
   intX = InputBox("Check Item", , Listview1.SelectedItem.Index)
   If IsNumeric(intX) Then         ' Ensure an integer was entered.
      If ListView1.ListItems(intX).Selected = True Then
         MsgBox ListView1.ListItems(intX).Text & " is selected."
      Else
         MsgBox "Not selected"
      End If
   End If
End Sub
```

## SelectedImage Property

See Also

Returns or sets the index or key value of a **ListImage** object in an associated **ImageList** control; the **ListImage** is displayed when a **Node** object is selected.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**SelectedImage** [ = *index*]

The **SelectedImage** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Node** object. |
| *index* | An integer or unique string that identifies a **ListImage** object in an associated **ImageList** control. The integer is the value of the **ListImage** object's **Index** property; the string is the value of the **Key** property. |

### Remarks

If **Null**, the mask of the default image specified by the **Image** property is used.

**See Also**

**Image** Property
**ImageList** Control
**TreeView** Control

## SelectedItem Property

See Also Example

Returns a reference to a selected **ListItem, Node**, or **Tab** object.

### Syntax

*object*.**SelectedItem**

The *object* placeholder represents an object expression that evaluates to a **ListView**, **TabStrip**, or **TreeView** control.

### Remarks

The **SelectedItem** property returns a reference to an object that can be used to set properties and invoke methods on the selected object. This property is typically used to return a reference to a **ListItem, Node**, or **Tab** or object that the user has clicked or selected. With this reference, you can validate an object before allowing any further action, as demonstrated in the following code:

```
Command1_Click()
   ' If the selected object is not the root, then remove the Node.
   If TreeView1.SelectedItem.Index <> 1 Then
      Treeview1.Nodes.Remove TreeView1.SelectedItem.Index
   End If
End Sub
```

**See Also**

**ListView** Control
**Selected** Property
**TreeView** Control

▪

## SelectedItem Property Example

This example adds several **Node** objects to a **TreeView** control. After you select a **Node**, click the form to see various properties of the **Node**. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, select a **Node**, and click the form.

```
Private Sub Form_Load()
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(, , "r", "Root")
   Set nodX = TreeView1.Nodes.Add("r", tvwChild, "c1", "Child 1")
   Set nodX = TreeView1.Nodes.Add("r", tvwChild, "c2", "Child 2")
   Set nodX = TreeView1.Nodes.Add("r", tvwChild, "c3", "Child 3")
   Set nodX = TreeView1.Nodes.Add("c3", tvwChild, "c4", "Child 4")
   Set nodX = TreeView1.Nodes.Add("c3", tvwChild, "c5", "Child 5")
   Set nodX = TreeView1.Nodes.Add("c5", tvwChild, "c6", "Child 6")
   Set nodX = TreeView1.Nodes.Add("c5", tvwChild, "c7", "Child 7")
   nodX.EnsureVisible
   TreeView1.BorderStyle = vbFixedSingle
End Sub

Private Sub Form_Click()
   Dim nodX As Node
   ' Set the variable to the SelectedItem.
   Set nodX = TreeView1.SelectedItem
   Dim strProps As String
   ' Retrieve properties of the node.
   strProps = "Text: " & nodX.Text & Chr(10)
   strProps = strProps & "Key: " & nodX.Key & Chr(10)
   On Error Resume Next ' Root node doesn't have a parent.
   strProps = strProps & "Parent: " & nodX.Parent.Text & Chr(10)
   strProps = strProps & "FirstSibling: " & _
   nodX.FirstSibling.Text & Chr(10)
   strProps = strProps & "LastSibling: " & _
   nodX.LastSibling.Text & Chr(10)
   strProps = strProps & "Next: " & nodX.Next.Text & Chr(10)

   MsgBox strProps
End Sub
```

## Sorted Property (TreeView Control)

See Also Example

- Returns or sets a value that determines whether the child nodes of a **Node** object are sorted alphabetically.
- Returns or sets a value that determines whether the root level nodes of a **TreeView** control are sorted alphabetically.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Sorted** [ = *boolean*]

The **Sorted** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Node** object or **TreeView** control. |
| *boolean* | A Boolean expression specifying whether the **Node** objects are sorted, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | The **Node** objects are sorted alphabetically. |
| **False** | The **Node** objects are not sorted. |

### Remarks

The **Sorted** property can be used in two ways: first, to sort the **Node** objects at the root (top) level of a **TreeView** control and, second, to sort the immediate children of any individual **Node** object. For example, the following code sorts the root nodes of a **TreeView** control:

```
Private Sub Command1_Click()
   TreeView1.Sorted = True   ' Top level Node objects are sorted.
End Sub
```

The next example shows how to set the **Sorted** property for a **Node** object as it is created:

```
Private Sub Form_Load()
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,,"Parent Node")
   nodX.Sorted = True
End Sub
```

In either case, setting the **Sorted** property to **True** means any new **Node** objects added to a **Node** or **TreeView** control will be sorted automatically.

**See Also**

**Node** Object, **Nodes** Collection

▪

## Sorted Property (TreeView Control) Example

This example adds several **Node** objects to a tree.  When you click a **Node**, you are asked if you want to sort the **Node**.  To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section.  Run the example, and click a **Node** to sort it.

```
Private Sub Form_Load()
   ' Create a tree with several unsorted Node objects.
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(, , , "Adam")
   Set nodX = TreeView1.Nodes.Add(1, tvwChild, "z", "Zachariah")
   Set nodX = TreeView1.Nodes.Add(1, tvwChild, , "Noah")
   Set nodX = TreeView1.Nodes.Add(1, tvwChild, , "Abraham")
   Set nodX = TreeView1.Nodes.Add("z", tvwChild, , "Stan")
   Set nodX = TreeView1.Nodes.Add("z", tvwChild, , "Paul")
   Set nodX = TreeView1.Nodes.Add("z", tvwChild, "f", "Frances")
   Set nodX = TreeView1.Nodes.Add("f", tvwChild, , "Julie")
   Set nodX = TreeView1.Nodes.Add("f", tvwChild, "c", "Carol")
   Set nodX = TreeView1.Nodes.Add("f", tvwChild, , "Barry")
   Set nodX = TreeView1.Nodes.Add("c", tvwChild, , "Yale")
   Set nodX = TreeView1.Nodes.Add("c", tvwChild, , "Harvard")
   nodX.EnsureVisible
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
   Dim answer As Integer
   ' Check if there are children nodes.
   If Node.Children > 1 Then ' There are more than one children nodes.
      answer = MsgBox("Sort this node?", vbYesNo)   ' Prompt user.
      If answer = vbYes Then        ' User wants to sort.
         Node.Sorted = True
      End If
   End If
End Sub
```

## Style Property (TreeView Control)

See Also Example

Returns or sets the type of graphics (images, text, plus/minus, and lines) and text that appear for each **Node** object in a **TreeView** control.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Style** [ = *number*]

The **Style** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TreeView** control. |
| *number* | An integer specifying the style of the graphics, as described in Settings. |

### Settings

The settings for *number* are:

| Setting | Description |
|---|---|
| 0 | Text only. |
| 1 | Image and text. |
| 2 | Plus/minus and text. |
| 3 | Plus/minus, image, and text. |
| 4 | Lines and text. |
| 5 | Lines, image, and text. |
| 6 | Lines, plus/minus, and text. |
| 7 | (Default) Lines, plus/minus, image, and text. |

### Remarks

If the **Style** property is set to a value that includes lines, the **LineStyle** property determines the appearance of the lines. If the **Style** property is set to a value that does not include lines, the **LineStyle** property will be ignored.

**See Also**

[**LineStyle** Property]
[**TreeView** Control Constants]

▪

## StyleProperty Example (TreeView Control)

This example adds several **Node** objects with images to a **TreeView** control. You can change the **LineStyle** and **Style** properties by selecting the alternate styles in two **ComboBox** controls. To try the example, place a **TreeView** control, an **ImageList** control, and two **ComboBox** controls on a form, and paste the code into the form's Declarations section. Run the example, and click either **ComboBox** to change the **LineStyle** and **Style** properties.

```
Private Sub Form_Load()
   ' Add an image to the ImageList control.
   Dim imgX As ListImage
   Set imgX = ImageList1.ListImages. _
   Add(,,LoadPicture("bitmaps\outline\leaf.bmp"))

   TreeView1.ImageList = ImageList1  ' Initialize ImageList.

   With combo1                 ' Populate ComboBox with line styles.
   .AddItem "Tree lines"
   .AddItem "Root lines"
   .ListIndex = 0              ' The default is TreeLines.
   End With

   With Combo2      ' Populate ComboBox with all styles.
   .AddItem "Text Only"                           ' 0
   .AddItem "Image & text"                        ' 1
   .AddItem "Plus/minus & text"                   ' 2
   .AddItem "Plus/minus, image & Text"            ' 3
   .AddItem "Lines & Text"                        ' 4
   .AddItem "Lines, image & Text"                 ' 5
   .AddItem "Lines, Plus/minus & Text"            ' 6
   .AddItem "Lines, plus/minus, image & text"     ' 7
   .ListIndex = 7
   End With


   Dim nodX As Node
   Dim i as Integer
   ' Create root node.
   Set nodX = TreeView1.Nodes.Add(,,,"Node 1",1)

   For i = 1 to 5                    ' Add 5 nodes.
      Set nodX = TreeView1.Nodes. _
      Add(i,tvwChild,,"Node " & CStr(i + 1),1)
   Next I
   nodX.EnsureVisible                ' Show all nodes.
End Sub

Private Sub combo1_Click()
   ' Change line style from ComboBox.
   TreeView1.LineStyle = combo1.ListIndex
End Sub

Sub combo2_Click()       ' Change Style with ComboBox.
   TreeView1.Style = Combo2.ListIndex
   Form1.Caption = "Indentation = " & Combo1.Text
End Sub
```

## StartLabelEdit Method

See Also Example

Enables a user to edit a label.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**StartLabelEdit**

The *object* placeholder is an object expression that evaluates to a **ListView** or **TreeView** control.

### Remarks

The **StartLabelEdit** method must be used to initiate a label editing operation when the **LabelEdit** property is set to 1 (Manual).

When the **StartLabelEdit** method is invoked upon an object, the BeforeLabelEdit event is also generated.

**See Also**

AfterLabelEdit Event
BeforeLabelEdit Event
**LabelEdit** Property
**ListItem** Object, **ListItems** Collection
**ListView** Control
**Node** Object, **Nodes** Collection
**TreeView** Control

▪

## StartLabelEdit Method Example

This example adds several **Node** objects to a **TreeView** control.   After a **Node** is selected, click the form to begin editing it.   To try the example, place a **TreeView** control on a form, and paste the code into the form's Declarations section.   Run the example, select a **Node**, and click the form.

```
Private Sub Form_Load
   Dim nodX As Node

   Set nodX = TreeView1.Nodes.Add(,,,"Da Vinci") ' Root
   Set nodX = TreeView1.Nodes.Add(1,tvwChild,,"Titian")
   Set nodX = TreeView1.Nodes.Add(1,tvwChild,,"Rembrandt")
   Set nodX = TreeView1.Nodes.Add(1,tvwChild,,"Goya")
   Set nodX = TreeView1.Nodes.Add(1,tvwChild,,"David")
   nodX.EnsureVisible ' Expand tree to see all nodes.
End Sub

Private Sub Form_Click()
   ' If selected Node isn't the Root node then allow edits.
   If TreeView1.SelectedItem.Index <> 1 Then
      TreeView1.StartLabelEdit
   End If
End Sub
```

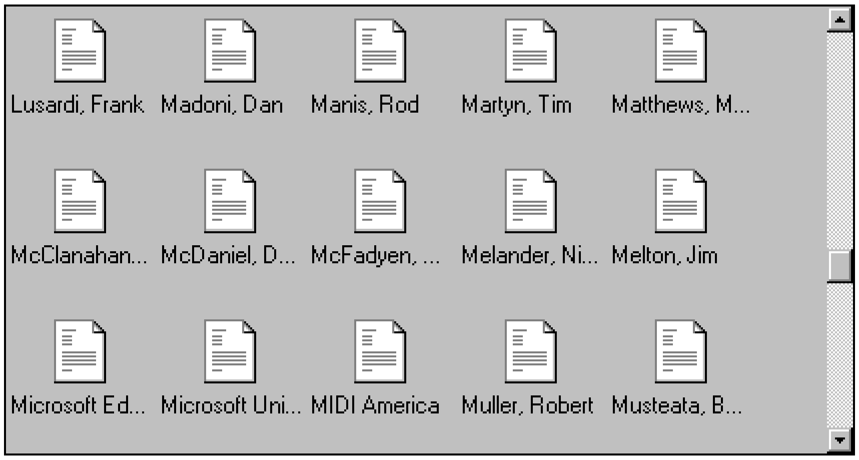
## Nodes

The **Nodes** keyword is used in these contexts:
**Nodes** Collection
**Nodes** Property

## ListView Control

See Also Properties Methods Events Constants

The **ListView** control displays items using one of four different views. You can arrange items into columns with or without column headings as well as display accompanying icons and text.



### Syntax

**ListView**

### Remarks

With a **ListView** control, you can organize list entries, called **ListItem** objects, into one of four different views:

- Large (standard) Icons
- Small Icons
- List
- Report

The **View** property determines which view the control uses to display the items in the list. You can also control whether the labels associated with items in the list wrap to more than one line using the **LabelWrap** property. In addition, you can manage how items in the list are sorted and how selected items appear.

The **ListView** control contains **ListItem** and **ColumnHeader** objects. A **ListItem** object defines the various characteristics of items in the **ListView** control, such as:

- A brief description of the item.
- Icons that may appear with the item, supplied by an **ImageList** control.
- Additional pieces of text, called subitems, associated with a **ListItem** object that you can display in Report view.

You can choose to display column headings in the **ListView** control using the **HideColumnHeaders** property. They can be added at both design and run time. At design time, you can use the Column Headers tab of the **ListView** Control Properties dialog box. At run time, use the **Add** method to add a **ColumnHeader** object to the **ColumnHeaders** collection.

**Distribution Note** The **ListView** control is a 32-bit custom control that can only run on Windows 95 and Windows NT 3.51 or higher. The **ListView** control is part of a group of custom controls that are found in the COMCTL32.OCX file. To use the **ListView** control in your application, you must add the COMCTL32.OCX file to the project. When distributing your application, install the COMCTL32.OCX file in the user's Microsoft Windows SYSTEM directory. For more information on how to add a custom control to a project, see the *Programmer's Guide*.

**See Also**

**Add** Method (**ColumnHeaders** Collection)
**ColumnHeader** Object, **ColumnHeaders** Collection
**ImageList** Control
**ListItem** Object, **ListItems** Collection
**TreeView** Control

Close

## ListView Control Properties

**Arrange** Property
**BackColor** Property
**BorderStyle** Property
**ColumnHeaders** Property
**Container** Property
**DragIcon** Property
**DragMode** Property
**DropHighLight** Property
**Enabled** Property
**Font** Property
**ForeColor** Property
**Height** Property
**HelpContextID** Property
**HideColumnHeaders** Property
**HideSelection** Property
**hWnd** Property
**Icons** Property
**Index** Property
**ListItems** Property
**LabelEdit** Property
**LabelWrap** Property
**Left** Property
**MouseIcon** Property
**MousePointer** Property
**MultiSelect** Property
**Name** Property
**Object** Property
**Parent** Property
**SelectedItem** Property
**SmallIcons** Property
**Sorted** Property (**ListView** Control**)**
**SortKey** Property
**SortOrder** Property
**TabIndex** Property
**TabStop** Property
**Tag** Property
**Top** Property
**View** Property
**Visible** Property
**WhatsThisHelpID** Property
**Width** Property

■
## ListView Control Methods

**Drag** Method
**FindItem** Method
**GetFirstVisible** Method
**HitTest** Method
**Move** Method
**Refresh** Method
**SetFocus** Method
**ShowWhatsThis** Method
**StartLabelEdit** Method
**ZOrder** Method

■
## ListView Control Events

AfterLabelEdit Event
BeforeLabelEdit Event
Click Event
ColumnClick Event
DblClick Event
DragDrop Event
DragOver Event
GotFocus Event
ItemClick Event
KeyDown Event
KeyPress Event
KeyUp Event
LostFocus Event
MouseDown Event
MouseMove Event
MouseUp Event

## ColumnHeader Object, ColumnHeaders Collection

See Also Properties Methods

- A **ColumnHeader** object is an item in a **ListView** control that contains heading text.
- A **ColumnHeaders** collection contains one or more **ColumnHeader** objects.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*listview.***ColumnHeaders**

*listview***.ColumnHeaders(***index***)**

The syntax lines above refer to the collection and to individual elements in the collection, respectively, according to the standard collection syntax.

The **ColumnHeader** object, **ColumnHeaders** collection syntax has these parts:

| Part | Description |
|---|---|
| *listview* | An object expression that evaluates to a **ListView** control. |
| *index* | Either an integer or string that uniquely identifies a member of an object collection. The integer is the value of the **Index** property; the string is the value of the **Key** property. |

### Remarks

You can view **ColumnHeader** objects in Report view only.

You can add **ColumnHeader** objects to a **ListView** control at both design time and run time.

With a **ColumnHeader** object, a user can:

- Click it to trigger the **ColumnClick** event and sort the items based on that data item.
- Grab the object's right border and drag it to adjust the width of the column.
- Hide **ColumnHeader** objects in Report view.

There is always one column in the **ListView** control, which is Column 1. This column contains the actual **ListItem** objects; not their subitems. The second column (Column 2) contains subitems. Therefore, you always have one more **ColumnHeader** object than subitems and the **ListItem** object's **SubItems** property is a 1-based array of size `ColumnHeaders.Count - 1`.

The number of **ColumnHeader** objects determines the number of subitems each **ListItem** object in the control can have. When you delete a **ColumnHeader** object, all of the subitems associated with the column are also deleted. Each **ListItem** object's subitem array shifts to update the indices of the **ColumnHeader**, causing the remaining column headers' **SubItemIndex** properties to change.

**See Also**

**Add** Method (**ColumnHeaders** Collection)
**Clear** Method
ColumnClick Event
**ColumnHeaders** Property
**HideColumnHeaders** Property
**Index** Property
**ListItem** Object, **ListItems** Collection
**Key** Property
**Remove** Method
**SubItemIndex** Property
**SubItems** Property (**ListItems** Object)

■

## ColumnHeader Object, ColumnHeaders Collection Properties

Legend

**Alignment** Property■

**Count** Property■
**Index** Property■
**Key** Property■
**Left** Property■
**SubItemIndex** Property■
**Tag** Property■
**Text** Property■
**Width** Property■

■

## ColumnHeader Object, ColumnHeaders Collection Methods

Legend

**Add** Method (**ColumnHeaders** Collection)■

**Clear** Method■
**Item** Method■
**Remove** Method■

## Add Method (ColumnHeaders Collection)

See Also Example

Adds a **ColumnHeader** object to a **ColumnHeaders** collection in a **ListView** control. Doesn't support named arguments.

**Important** This method requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Add(***index, key, text, width, alignment***)**

The **Add** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **ColumnHeaders** collection. |
| *index* | Optional. An integer that uniquely identifies a member of an object collection. |
| *key* | Optional. A unique string expression that can be used to access a member of the collection. |
| *text* | Optional. A string that appears in the **ColumnHeader** object. |
| *width* | Optional. A numeric expression specifying the width of the object using the scale units of the control's container. |
| *alignment* | Optional. An integer that determines the alignment of text in the **ColumnHeader** object. See the **Alignment** property for settings. |

### Remarks

The **Add** method returns a reference to the newly inserted **ColumnHeader** object.

Use the *index* argument to insert a column header in a specific position.

**See Also**

**Alignment** Property (**ColumnHeader** Object)
**Clear** Method
**ColumnHeader** Object**, ColumnHeaders** Collection
**Index** Property
**Key** Property
**ListView** Control
**Remove** Method
**SubItemIndex** Property
**SubItems** Property (**ListItems** Object)

## ListItem Object, ListItems Collection

See Also Properties Methods

- A **ListItem** consists of text, the index of an associated icon (**ListImage** object), and, in Report view, an array of strings representing subitems.
- A **ListItems** collection contains one or more **ListItem** objects.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*listview*.**ListItems**

*listview*.**ListItems(***index***)**

The syntax lines above refer to the collection and to individual elements in the collection, respectively, according to the standard collection syntax.

The **ListItem** object, **ListItems** collection syntax has these parts:

| Part | Description |
|---|---|
| *listview* | An object expression that evaluates to a **ListView** control. |
| *index* | Either an integer or string that uniquely identifies a member of a **ListItem** collection. The integer is the value of the **Index** property; the string is the value of the **Key** property. |

### Remarks

**ListItem** objects can contain both text and pictures. However, to use pictures, you must reference an **ImageList** control.

You can change the image by using the **Icon** or **SmallIcon** property.

The following example shows how to add **ColumnHeaders** and several **ListItem** objects with subitems to a **ListView** control.

```
Private Sub Form_Load()
   Dim clmX As ColumnHeader
   Dim itmX As ListItem
   Dim i As Integer

   For i = 1 To 3
      Set clmX = ListView1.ColumnHeaders.Add()
      clmX.Text = "Col" & i
   Next i

   For i = 1 To 10
      Set itmX = ListView1.ListItems.Add()
      itmX.SmallIcon = 1
      itmX.Text = "ListItem " & i
      itmX.SubItems(1) = "Subitem 1"
      itmX.SubItems(2) = "Subitem 2"
   Next i
End Sub
```

**See Also**

**Icon, SmallIcon** Properties
**ImageList** Control
**ListImage** Object, **ListImages** Collection
**ListView** Control

■
## ListItem Object, ListItems Collection Properties

Legend

**Count** Property■
**Ghosted** Property■
**Height** Property■
**Icon** Property■
**Index** Property■
**Key** Property■
**Left** Property■
**Selected** Property■
**SmallIcon** Property■
**SubItems** Property (**Listitems** Object)■
**Tag** Property■
**Text** Property■
**Top** Property■
**Width** Property■

■

**ListItem Object, ListItems Collection Methods**

Legend

**Add** Method (**ListItems** Collection)■

**Clear** Method■
**CreateDragImage** Method■
**EnsureVisible** Method■
**Item** Method■
**Remove** Method■

## Add Method (ListItems Collection)

See Also Example

Adds a **ListItem** object to a **ListItems** collection in a **ListView** control and returns a reference to the newly created object. Doesn't support named arguments.

**Important** This method requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Add(***index, key, text, icon, smallIcon***)**

The **Add** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **ListItems** collection. |
| *index* | Optional. An integer specifying the position where you want to insert the **ListItem**. If no index is specified, the **ListItem** is added to the end of the **ListItems** collection. |
| *key* | Optional. A unique string expression that can be used to access a member of the collection. |
| *text* | Optional. A string that is associated with the **ListItem** object control. |
| *icon* | Optional. An integer that sets the icon to be displayed from an **ImageList** control, when the **ListView** control is set to Icon view. |
| *smallIcon* | Optional. An integer that sets the icon to be displayed from an **ImageList** control, when the **ListView** control is set to SmallIcon view. |

### Remarks

Before setting either the **Icons** or **SmallIcons** properties, you must first initialize them. You can do this at design time with the General tab of the **ListView** Control Properties dialog box, or at run time with the following code:

```
ListView1.Icons = ImageList1   'Assuming the Imagelist is ImageList1.
ListView1.SmallIcons = ImageList2
```

If the list is not currently sorted, a **ListItem** object can be inserted in any position by using the *index* argument. If the list is sorted, the *index* argument is ignored and the **ListItem** object is inserted in the appropriate position based upon the sort order.

If *index* is not supplied, the **ListItem** object is added with an index that is equal to the greatest number + 1.

**See Also**

**Count** Property
**Ghosted** Property
**Index** Property
**Item** Method
**ListItem** Object, **ListItems** Collection
**Key** Property
**ListView** Control
**Selected** Property
**Sorted** Property (**ListView** Control)
**SubItems** Property

▪

## Add Method (ListItems, ColumnHeaders), ListItems Property, SubItems Property Example

The following example adds several **ListItem** objects with images from an **ImageList** control to a **ListView** control. To try this example, place a **ComboBox**, **ListView**, and two **ImageList** controls on a form and paste the code into the Declarations section. Note: the example will not run unless you add a reference to the Microsoft DAO 3.0 Object Library by using the References command on the Tools menu. Run the example.

```
Private Sub Form_Load()
   ' Create an object variable for the ColumnHeader object.
   Dim clmX As ColumnHeader
   ' Add ColumnHeaders.  The width of the columns is the width
   ' of the control divided by the number of ColumnHeader objects.
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Author", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Author ID", ListView1.Width / 3, lvwColumnCenter)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Birthdate", ListView1.Width / 3)

   ListView1.View = lvwReport ' Set View property to Report.

   ' Load one image into an ImageList control.
   Dim imgX As ListImage
   Set imgX = ImageList1.ListImages.Add _
   (,,LoadPicture("icons\Writing\Note06.ico"))
   Set imgX = ImageList2.ListImages.Add _
   (,,LoadPicture("bitmaps\assorted\w.bmp"))
   ' Set Icons property to ImageList1.
   ListView1.Icons = ImageList1
   ListView1.SmallIcons = ImageList2

   ' Add items to a ComboBox for switching views.
   With Combo1
      .AddItem "Icon"               ' 0
      .AddItem "SmallIcon"          ' 1
      .AddItem "List"               ' 2
      .AddItem "Report"             ' 3
      .ListIndex = 0
   End With

   ' Create object variables for the Data Access objects.
   Dim myDb As Database, myRs As Recordset
   ' Set the Database to the BIBLIO.MDB database.
   Set myDb = DBEngine.Workspaces(0).OpenDatabase("BIBLIO.MDB")
   ' Set the recordset to the "Authors" table.
   Set myRs = myDb.OpenRecordset("Authors", dbOpenDynaset)

   ' Create a variable to add ListItem objects.
   Dim itmX As ListItem

   ' While the record is not the last record, add a ListItem object.
   ' Use the author field for the ListItem object's text.
   ' Use the ID field 0 for the ListItem object's SubItem(1).
   ' Use the "Year of Birth" field for the ListItem object's SubItem(2).
   While Not myRs.EOF
      Set itmX = ListView1.ListItems. _
      Add(, , CStr(myRs!Author),1)   ' Author.

      ' If the AuthorID field is not null, then set SubItem 1 to it.
      If Not IsNull(myRs!Au_id) Then
         itmX.SubItems(1) = CStr(myRs!Au_id)    ' Author ID.
      End If

      ' If the birth field is not Null, set the SubItem 2 to it.
      If Not IsNull(myRs![Year Born]) Then
         itmX.SubItems(2) = myRs![Year Born]
      End If
      myRs.MoveNext                  ' Move to next record.
   Wend
End Sub

Private Sub combo1_Click()
   ' Switch ListView with the ComboBox.
   ListView1.View = combo1.ListIndex
End Sub
```

## Alignment Property (ColumnHeader Object)

See Also

Returns or sets the alignment of text in a **ColumnHeader** object.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Alignment** [= *integer*

The **Alignment** Property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ColumnHeader** object. |
| *integer* | An integer that determines the alignment, as described in Settings. |

### Settings

The settings for *integer* are:

| Constant | Value | Description |
|---|---|---|
| **lvwColumnLeft** | 0 | (Default) Left. Text is aligned left. |
| **lvwColumnRight** | 1 | Right. Text is aligned right. |
| **lvwColumnCenter** | 2 | Center. Text is centered. |

**See Also**

**Add** Method (**ColumnHeaders** Collection)
**ColumnHeader** Object, **ColumnHeaders** Collection
**ListView** Control

## Arrange Property

See Also Example

Returns or sets a value that determines how the icons in a **ListView** control's Icon or SmallIcon view are arranged. Only effective for Icon or SmallIcon view.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Arrange** [= *value*]

The **Arrange** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListView** control. |
| *value* | An integer or constant that determines how the icons or small icons are arranged, as described in Settings. |

### Settings

The settings for *value* are:

| Constant | Value | Description |
|---|---|---|
| **lvwNoArrange** | 0 | (Default) None. |
| **lvwAutoLeft** | 1 | Left. Items are aligned automatically along the left side of the control. |
| **lvwAutoTop** | 2 | Top. Items are aligned automatically along the top of the control. |

**See Also**

**Icons, SmallIcons** Properties (**ListView** Control)
**ListView** Control
**Sorted** Property (**ListView** Control)
**SortKey** Property
**SortOrder** Property
**View** Property

■

## Arrange Property Example

This example adds several **ListItem** objects and subitems to a **ListView** control. When you click on the **ComboBox** control, the **Arrange** property is set with the **ListIndex** value of the **ComboBox**. To try the example, place a **ComboBox**, **ListView**, and two **ImageList** controls on a form and paste the code into the form's Declarations section. Run the example and click on the **CombBox** to change the **Arrange** property.

```
Private Sub Combo1_Click()
   ' Set Arrange property to Combo1.ListIndex.
   ListView1.Arrange = Combo1.ListIndex
End Sub

Private Sub Form_Load()
   ' Populate ComboBox with Arrange choices.
   With Combo1
      .AddItem "No Arrange"         ' 0
      .AddItem "Align Auto Left"    ' 1
      .AddItem "Align Auto Top"     ' 2
      .ListIndex = 0
   End With

   ' Declare variables for creating ListView and ImageList objects.
   Dim i As Integer
   Dim itmX As ListItem      ' Object variable for ListItems.
   Dim imgX As ListImage     ' Object variable for ListImages.

   ' Add a ListImage object to an ImageList control.
   Set imgX = ImageList1.ListImages. _
   Add(,,LoadPicture("icons\mail\mail01a.ico"))

   ListView1.Icons = ImageList1     ' Associate an ImageList control.

   ' Add ten ListItem objects, each with an Icon.
   For i = 1 To 10
      Set itmX = ListView1.ListItems.Add()
      itmX.Icon = 1                 ' Icon.
      itmX.Text = "ListItem " & i
   Next i
End Sub
```

## ColumnClick Event

See Also Example

Occurs when a **ColumnHeader** object in a **ListView** control is clicked. Only available in Report view.

---

**Important** This event requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

**Private Sub** *object*_**ColumnClick**(**ByVal** *columnheader* **As ColumnHeader**)

The **ColumnClick** event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListView** control. |
| *columnheader* | A reference to the **ColumnHeader** object that was clicked. |

### Remarks

The **Sorted**, **SortKey**, and **SortOrder** properties are commonly used in code to sort the list using the clicked column header as the key.

**See Also**

Click Event
**ListView** Control
**Sorted** Property (**ListView** Control)
**SortKey** Property
**SortOrder** Property

## ColumnHeaders Property

See Also

Returns a reference to a collection of **ColumnHeader** objects.

---

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**ColumnHeaders**

The *object* placeholder represents an object expression that evaluates to a **ListView** control.

### Remarks

You can manipulate **ColumnHeader** objects using standard collection methods (for example, the **Remove** method). Each **ColumnHeader** in the collection can be accessed either by its index or by a unique key, stored in the **Key** property.

**See Also**

**Add** Method (**ColumnHeaders** Collection)
**ColumnHeader** Object, **ColumnHeaders** Collection
**Key** Property
**ListView** Control

## Ghosted Property

See Also Example

Returns or sets a value that determines whether a **ListItem** object in a **ListView** control is dimmed.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object.***Ghosted** [= *boolean*]

The **Ghosted** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListItem** object. |
| *boolean* | A Boolean expression specifying if the icon or small icon is ghosted, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | The **ListItem** object's icon is dimmed. |
| **False** | (Default) The **ListItem** isn't dimmed. |

### Remarks

The **Ghosted** property is typically used to show when a **ListItem** is cut.

When a ghosted **ListItem** is selected, the label is highlighted but its image is not.

**See Also**

- **ListItem** Method
- **ListView** Control
- **SelectedItem** Property

▪

## Ghosted, MultiSelect Properties Example

This example populates a **ListView** control with the contents of the Authors table from the BIBLIO.MDB database, and fills a **ComboBox** with **MultiSelect** property options.   You can select any item, or hold down the SHIFT Key and select multiple items.   Clicking on the **CommandButton** sets the **Ghosted** property of the selected items to **True**.   To try the example, place a **ComboBox**, **ListView**, **ImageList**, and **CommandButton** control on a form and paste the code into the form's Declarations section.   Note: the example will not run unless you add a reference to the Microsoft DAO 3.0 Object Library by using the References command on the Tools menu.   Run the example, select a **MultiSelect** option from the **ComboBox**, click on items to select them and click the **CommandButton** to ghost them.

```
Private Sub Command1_Click()
   Dim x As Object
   Dim i As Integer
   ' Ghost selected ListItem.
If ListView1.SelectedItem Is Nothing Then Exit Sub
   For i = 1 To ListView1.ListItems.Count
      If ListView1.ListItems(i).Selected = True Then
         ListView1.ListItems(i).Ghosted = True
      End If
   Next i
End Sub

Private Sub Form_Load()
   ' Create an object variable for the ColumnHeader object.
   Dim clmX As ColumnHeader
   ' Add ColumnHeaders.  The width of the columns is the width
   ' of the control divided by the number of ColumnHeader objects.
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Company", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Address", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Phone", ListView1.Width / 3)

   ' Populate Combobox with MultiSelect options.
   With Combo1
      .AddItem "No MultiSelect"
      .AddItem " MultiSelect"
      .ListIndex = 1             ' Set MultiSelect to True.
   End With

   ListView1.BorderStyle = ccFixedSingle ' Set BorderStyle property.
   ListView1.View = lvwReport       ' Set View property to Report.
   ' Add one image to ImageList control.
   Dim imgX As ListImage
   Set imgX = ImageList1.ListImages. _
   Add(, , LoadPicture("icons\mail\mail01a.ico"))
   ListView1.Icons = ImageList1

   ' Create object variables for the Data Access objects.
   Dim myDb As Database, myRs As Recordset
   ' Set the Database to the BIBLIO.MDB database.
   Set myDb = DBEngine.Workspaces(0).OpenDatabase("BIBLIO.MDB")
   ' Set the recordset to the Publishers table.
   Set myRs = myDb.OpenRecordset("Publishers", dbOpenDynaset)
   ' Create a variable to add ListItem objects.
   Dim itmX As ListItem

   ' While the record is not the last record, add a ListItem object.
   ' Use the Name field for the ListItem object's text.
   ' Use the Address field for the ListItem object's SubItem(1).
   ' Use the Phone field for the ListItem object's SubItem(2).

   While Not myRs.EOF
      Set itmX = ListView1.ListItems.Add(, , CStr(myRs!Name))
      itmX.Icon = 1  ' Set icon to the ImageList icon.

      ' If the Address field is not Null, set SubItem 1 to the field.
      If Not IsNull(myRs!Address) Then
         itmX.SubItems(1) = CStr(myRs!Address)  ' Address field.
      End If

      ' If the Phone field is not Null, set SubItem 2 to the field.
      If Not IsNull(myRs!Telephone) Then
         itmX.SubItems(2) = myRs!Telephone  ' Phone field.
      End If

      myRs.MoveNext  ' Move to next record.
   Wend

   ListView1.View = lvwIcon  ' Show Icons view.
   Command1.Caption = "Cut"  ' Set caption of the CommandButton.
   ' Add a caption to the form.
   Me.Caption = "Select any item(s) and click 'Cut'."
End Sub

Private Sub Combo1_Click()
   ListView1.MultiSelect = Combo1.ListIndex
End Sub
```

## HideColumnHeaders Property

See Also Example

Returns or sets whether **ColumnHeader** objects in a **ListView** control are hidden in Report view.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**HideColumnHeaders** [= *boolean*]

The **HideColumnHeaders** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListView** control. |
| *boolean* | A Boolean expression that specifies if the column headers are visible in Report view, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | The column headers are not visible. |
| **False** | (Default) The column headers are visible. |

### Remarks

The subitems remain visible even if the **HideColumnHeaders** property is set to **True**.

**See Also**

**ColumnHeaders** Property
**ListView** Control

▪
## HideColumnHeaders Property Example

This example adds several **ListItem** objects with subitems to a **ListView** control.   When you click on the **CommandButton**, the **HideColumnHeaders** property toggles between **True** (-1) and **False** (0).   To try the example, place **ListView** and **CommandButton** controls on a form and paste the code into the form's Declarations section.   Run the example and click the **CommandButton** to toggle the **HideColumnHeaders** property.

```
Private Sub Command1_Click()
   ' Toggle HideColumnHeaders property off and on.
   ListView1.HideColumnHeaders = Abs(ListView1.HideColumnHeaders) - 1
End Sub

Private Sub Form_Load()
   Dim clmX As ColumnHeader
   Dim itmX As ListItem
   Dim i As Integer
   Command1.Caption = "HideColumnHeaders"

   ' Add 3 ColumnHeader objects to the control.
   For i = 1 To 3
      Set clmX = ListView1.ColumnHeaders.Add()
      clmX.Text = "Col" & i
   Next I

   ' Set View to Report.
   ListView1.View = lvwReport

   ' Add 10 ListItems to the control.
   For i = 1 To 10
      Set itmX = ListView1.ListItems.Add()
      itmX.Text = "ListItem " & i
      itmX.SubItems(1) = "Subitem 1"
      itmX.SubItems(2) = "Subitem 2"
   Next i
End Sub
```

## Icon, SmallIcon Properties (ListItem Object)

See Also Example

Returns or sets the index or key value of an icon or small icon associated with a **ListItem** object in an **ImageList** control.

---

**Important** These properties require either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**Icon** [= *index*]

*object*.**SmallIcon** [= *index*]

The **Icon**, **SmallIcon** properties syntax has the following parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListItem** object. |
| *index* | An integer or unique string that identifies an icon or smallicon in an associated **ImageList** control. The integer is the value of the **ListItem** object's **Index** property; the string is the value of the **Key** property. |

### Remarks

Before you can use an icon in a **ListItem** object, you must associate an **ImageList** control with the **ListView** control containing the object. See the **Icons**, **SmallIcons** Properties (**ListView** Control) for more information. The example below shows the proper syntax:

```
ListView1.ListItems(1).SmallIcons=1
```

The images will appear when the **ListView** control is in SmallIcons view.

**See Also**

**Icons, SmallIcons** Properties (**ListView** Control)
**ImageList** Control
**Index** Property
**ListView** Control

## Icons, SmallIcons Properties

See Also Example

Returns or sets the **ImageList** controls associated with the Icon and SmallIcon views in a **ListView** control.

---

**Important** These properties require either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**Icons** [= *imagelist*]

*object.***SmallIcons** [= *imagelist*]

The **Icons**, **SmallIcons** properties syntax has the following parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to the **ListView** control. |
| *imagelist* | An object expression that evaluates to an **ImageList** control. |

### Remarks

To associate an **ImageList** control with a **ListView** control at run time, set these properties to the desired **ImageList** control.

Each **ListItem** object in the **ListView** control also has properties, **Icon** and **SmallIcon**, which index the **ListImage** objects and determine which image is displayed.

Once you associate an **ImageList** with the **ListView** control, you can use the value of either the **Index** or **Key** property to refer to a **ListImage** object in a procedure.

**See Also**

**Icon**, **SmallIcon** Properties (**ListItem** Object)
**ImageList** Control
**ListItem** Object, **ListItems** Collection
**ListView** Control

▪

## Icon, SmallIcon, Icons, SmallIcons, View Properties Example

This example populates a **ListView** control with the contents of the Publishers table in the BIBLIO.MDB database. A **ComboBox** control is populated with **View** property choices. You must place two **ImageList** controls on the form, one to contain images for the **Icon** property, and a second to contain images for the **SmallIcon** property of each **ListItem** object. To try the example, place a **ListView**, **ComboBox**, and two **ImageList** controls on a form and paste the code into the form's Declarations section. Note: the example will not run unless you add a reference to the Microsoft DAO 3.0 Object Library by using the References command on the Tools menu. Run the example and click on the **ComboBox** control to switch views.

```
Private Sub combo1_Click()
   ' Set the ListView control's View property to the
   ' ListIndex of Combo1.
   ListView1.View = combo1.ListIndex
End Sub

Private Sub Form_Load()
   ' Create an object variable for the ColumnHeader object.
   Dim clmX As ColumnHeader
   ' Add ColumnHeaders.  The width of the columns is the width
   ' of the control divided by the number of ColumnHeader objects.
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Company", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Address", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Phone", ListView1.Width / 3)

   ListView1.BorderStyle = ccFixedSingle ' Set BorderStyle property.
   ListView1.View = lvwReport ' Set View property to Report.

   ' Add one image to ImageList1--the Icons ImageList.
   Dim imgX As ListImage
   Set imgX = ImageList1.ListImages. _
   Add(, , LoadPicture("icons\mail\mail01a.ico"))
   ' Add an image to ImageList2--the SmallIcons ImageList.
   Set imgX = ImageList2.ListImages. _
   Add(, , LoadPicture("bitmaps\assorted\w.bmp"))

   ' To use ImageList controls with the ListView control, you must
   ' associate a particular ImageList control with the Icons and
   ' SmallIcons properties.
   ListView1.Icons = ImageList1
   ListView1.SmallIcons = ImageList2
   ' Populate ComboBox1 with View choices.
   With Combo1
      .AddItem "Icon"              ' 0
      .AddItem "SmallIcon"         ' 1
      .AddItem "List"              ' 2
      .AddItem "Report"            ' 3
      .ListIndex = 0               ' Set to Icon View.
   End With

   ' Create object variables for the Data Access objects.
   Dim myDb As Database, myRs As Recordset
   ' Set the Database to the BIBLIO.MDB database.
   Set myDb = DBEngine.Workspaces(0).OpenDatabase("BIBLIO.MDB")
   ' Set the recordset to the Publishers table.
   Set myRs = myDb.OpenRecordset("Publishers", dbOpenDynaset)

   ' Create a variable to add ListItem objects.
   Dim itmX As ListItem

   ' While the record is not the last record, add a ListItem object.
   ' Use the Name field for the ListItem object's text.
   ' Use the Address field for the ListItem object's SubItem(1)
   ' Use the Phone field for the ListItem object's SubItem(2)

   While Not myRs.EOF

      Set itmX = ListView1.ListItems.Add(, , CStr(myRs!Name))
      itmX.Icon = 1                ' Set an icon from ImageList1.
      itmX.SmallIcon = 1           ' Set an icon from ImageList2.

      ' If the Address field is not Null, set SubItem 1 to the field.
      If Not IsNull(myRs!Address) Then
         itmX.SubItems(1) = CStr(myRs!Address) ' Address field.
      End If

      ' If the Phone field is not Null, set SubItem 2 to the field.
      If Not IsNull(myRs!Telephone) Then
         itmX.SubItems(2) = myRs!Telephone  ' Phone field.
      End If

      myRs.MoveNext  ' Move to next record.
   Wend
End Sub
```

## ListItems Property

See Also     Example

Returns a reference to a collection of **ListItem** objects in a **ListView** control.

---

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**ListItems**

The *object* placeholder represents an object expression that evaluates to a **ListView** control.

### Remarks

**ListItem** obects can be manipulated using the standard collection methods. Each **ListItem** in the collection can be accessed by its unique key, which you create and store in the **Key** property.

You can also retrieve **ListItem** objects by their display position using the **Index** property.

**See Also**

**Icons**, **SmallIcons** Properties (**ListView** Control)
**Index** Property
**Key** Property
**ListItem** Object, **ListItems** Collection
**ListView** Control

## LabelWrap Property

See Also

Returns or sets a value that determines whether or not labels are wrapped when a **ListView** control is in Icon view.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**LabelWrap** [= *boolean*]

The **LabelWrap** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListView** control. |
| *boolean* | A Boolean expression specifying if the labels wrap, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) The labels wrap. |
| **False** | The labels don't wrap. |

### Remarks

The length of the label is determined by setting the icon spacing in the Control Panel. In Windows NT, use the Desktop option. In Windows 95, use the Appearance tab in the Display Properties Dialog box.

**See Also**

**ListView** Control

**View** Property

## MultiSelect Property

See Also Example

Returns or sets a value indicating whether a user can make multiple selections in the **ListView** control and how they can be made.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**MultiSelect** [= *boolean*]

The **MultiSelect** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListView** control. |
| *boolean* | A value specifying the type of selection, as described in Settings. |

### Settings

The settings for *boolean* are:

| Constant | Description |
|---|---|
| **False** | (Default) Multiple selection isn't allowed. |
| **True** | Multiple selection. Pressing SHIFT and clicking the mouse or pressing SHIFT and one of the arrow keys (UP ARROW, DOWN ARROW, LEFT ARROW, and RIGHT ARROW) extends the selection from the previously selected **ListItem** to the current **ListItem**. Pressing CTRL and clicking the mouse selects or deselects a **ListItem** in the list. |

**See Also**

**ListItem** Object**, ListItems** Collection

**ListView** Control

## SortKey Property

See Also Example

Returns or sets a value that determines how the **ListItem** objects in a **ListView** control are sorted.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**SortKey** [= *integer*]

The **SortKey** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListView** control. |
| *integer* | An integer specifying the sort key, as described in Settings. |

### Settings

The settings for *integer* are:

| Setting | Description |
|---|---|
| 0 | Sort using the **ListItem** object's **Text** property. |
| ≥ 1 | Sort using this subitem. |

### Remarks:

The **Sorted** property must be set to **True** before the change takes place.

It is common to sort a list when the column header is clicked.   For this reason, the **SortKey** property is commonly included in the **ColumnClick** event to sort the list using the clicked column, as determined by the sort key, and demonstrated in the following example:

```
Private Sub ListView1_ColumnClick (ByVal ColumnHeader as ColumnHeader)
   ListView1.SortKey=ColumnHeader.Index-1
End Sub
```

**See Also**

ColumnClick Event
**ListView** Control
**SortOrder** Property
**Text** Property

## SortOrder Property

See Also Example

Returns or sets a value that determines whether **ListItem** objects in a **ListView** control are sorted in ascending or descending order.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**SortOrder** [= *integer*]

The **SortOrder** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListView** control. |
| *integer* | An integer specifying the type of sort order, as described in Settings. |

### Settings

The settings for *integer* are:

| Constant | Value | Description |
|---|---|---|
| **lvwAscending** | 0 | (Default) Ascending order. Sorts from the beginning of the alphabet (A-Z), the earliest date, or the lowest number. |
| **lvwDescending** | 1 | Descending order. Sorts from the end of the alphabet (Z-A), the latest date, or the highest number. |

### Remarks

The **Sorted** property must be set to **True** before **SortOrder** can reorder the list.

**See Also**

**ListView** Control
**Sorted** Property (**ListView** Control)
**SortKey** Property

■

## SortKey, SortOrder, Sorted Properties, ColumnClick Event Example

This example adds three **ColumnHeader** objects to a **ListView** control and populates the control with the Publishers records of the BIBLIO.MDB database.   A **ComboBox** control contains the two choices for sorting records.   When you click on a **ColumnHeader**, the **ListView** control is sorted according to the **SortOrder** property, as determined by the **ComboBox** control.   To try the example, place a **ListView** and a **ComboBox** control on a form and paste the code into the form's Declarations section. Run the example and click on the **ColumnHeaders** to sort, and click on the **ComboBox** to switch the **SortOrder** property.   Also, the example will not run unless you add a reference to the Microsoft DAO 3.0 Object Library by using the References command on the Tools menu.

```
Private Sub Combo1_Click()
   ' This ComboBox has two items: Ascending (ListIndex 0),
   ' and Descending (ListIndex 1).  Clicking on one of these
   ' sets the SortOrder for the ListView control.
   ListView1.SortOrder = Combo1.ListIndex
   ListView1.Sorted = True ' Sort the List.
End Sub

Private Sub Form_Load()
   ' Create an object variable for the ColumnHeader object.
   Dim clmX As ColumnHeader
   ' Add ColumnHeaders.  The width of the columns is the width
   ' of the control divided by the number of ColumnHeader objects.
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Company", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Address", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Phone", ListView1.Width / 3)

   ListView1.BorderStyle = ccFixedSingle ' Set BorderStyle property.
   ListView1.View = lvwReport ' Set View property to Report.

   ' Populate ComboBox with SortOrder choices.
   With Combo1
      .AddItem "Ascending (A-Z)"
      .AddItem "Descending (Z-A)"
      .ListIndex = 0
   End With

   ' Create object variables for the Data Access objects.
   Dim myDb As Database, myRs As Recordset
   ' Set the Database to the BIBLIO.MDB database.
   Set myDb = DBEngine.Workspaces(0).OpenDatabase("BIBLIO.MDB")
   ' Set the recordset to the Publishers table.
   Set myRs = myDb.OpenRecordset("Publishers", dbOpenDynaset)

   ' Create a variable to add ListItem objects.
   Dim itmX As ListItem

   ' While the record is not the last record, add a ListItem object.
   ' Use the Name field for the ListItem object's text.
   ' Use the Address field for the ListItem object's subitem(1).
   ' Use the Phone field for the ListItem object's subitem(2).
   While Not myRs.EOF
      Set itmX = ListView1.ListItems.Add(, , CStr(myRs!Name))

      ' If the Address field is not Null, set subitem 1 to the field.
      If Not IsNull(myRs!Address) Then
         itmX.SubItems(1) = CStr(myRs!Address)   ' Address field.
      End If

      ' If the Phone field is not Null, set subitem 2 to the field.
      If Not IsNull(myRs!Telephone) Then
         itmX.SubItems(2) = myRs!Telephone   ' Phone field.
      End If

      myRs.MoveNext                ' Move to next record.
   Wend
End Sub

Private Sub ListView1_ColumnClick(ByVal ColumnHeader As ColumnHeader)
   ' When a ColumnHeader object is clicked, the ListView control is
   ' sorted by the subitems of that column.
   ' Set the SortKey to the Index of the ColumnHeader - 1
   ListView1.SortKey = ColumnHeader.Index - 1
   ' Set Sorted to True to sort the list.
   ListView1.Sorted = True
End Sub
```

## SubItemIndex Property

See Also    Example

Returns the index of the subitem associated with a **ColumnHeader** object in a **ListView** control.

---

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**SubItemIndex** [= *integer*

The **SubItemIndex** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ColumnHeader** object. |
| *integer* | An integer specifying the index of the subitem associated with the **ColumnHeader** object. |

### Remarks

Subitems are arrays of strings representing the **ListItem** object's data and are displayed in Report view.

The first column header always has a **SubItemIndex** property set to 0 because the small icon and the **ListItem** object's text always appear in the first column and are considered **ListItem** objects rather than subitems.

The number of column headers dictates the number of subitems. There is always exactly one more column header than there are subitems.

**See Also**

**ListView** Control

**SubItems** Property

■

## SubItemIndex Property Example

This example adds three **ColumnHeader** objects to a **ListView** control. The code then adds several **ListItem** and **Subitems** using the **SubItemIndex** to associate the **SubItems** string with the correct **ColumnHeader** object. To try the example, place a **ListView** control on a form and paste the code into the form's Declarations section. Run the example.

```
' Make sure ListView control is in report view.
ListView1.View = lvwReport

' Add three columns.
ListView1.ColumnHeaders.Add , "Name", "Name"
ListView1.ColumnHeaders.Add , "Address", "Address"
ListView1.ColumnHeaders.Add , "Phone", "Phone"

' Add ListItem objects to the control.
Dim itmX As ListItem
' Add names to column 1.
Set itmX= ListView1.ListItems.Add(1, "Mary", "Mary")
' Use the SubItemIndex to associate the SubItem with the correct
' ColumnHeader.  Use the key ("Address") to specify the right
' ColumnHeader.
itmX.SubItems(ListView1.ColumnHeaders("Address").SubItemIndex) _
= "212 Grunge Street"
' Use the ColumnHeader key to associate the SubItems string
' with the correct ColumnHeader.
itmX.SubItems(ListView1.ColumnHeaders("Phone").SubItemIndex) _
= "555-1212"

Set itmX = ListView1.ListItems.Add(2, "Bill", "Bill")
itmX.SubItems(ListView1.ColumnHeaders("Address").SubItemIndex) _
= "101 Pacific Way"
itmX.SubItems(ListView1.ColumnHeaders("Phone").SubItemIndex) _
= "555-7879"

Set itmX= ListView1.ListItems.Add(3, "Susan", "Susan")
itmX.SubItems(ListView1.ColumnHeaders("Address").SubItemIndex) = _
"800 Chicago Street"
itmX.SubItems(ListView1.ColumnHeaders("Phone").SubItemIndex) = _
"555-4537"

Set itmX= ListView1.ListItems.Add(4, "Tom", "Tom")
itmX.SubItems(ListView1.ColumnHeaders("Address").SubItemIndex) _
= "200 Ocean City"
itmX.SubItems(ListView1.ColumnHeaders("Phone").SubItemIndex) = _
"555-0348"
```

## View Property

See Also Example

Returns or sets the appearance of the **ListItem** objects in a **ListView** control.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**View** [= *value*]

The **View** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | The object expression that evaluates to a **ListView** control. |
| *value* | An integer or constant specifying the control's appearance, as described in Settings. |

### Settings

The settings for *value* are:

| Constant | Value | Description |
|---|---|---|
| **lvwIcon** | 0 | (Default) Icon. Each **ListItem** object is represented by a full-sized (standard) icon and a text label. |
| **lvwSmallIcon** | 1 | SmallIcon. Each **ListItem** object is represented by a small icon and a text label that appears to the right of the icon. The items appear horizontally. |
| **lvwList** | 2 | List. Each **ListItem** object is represented by a small icon and a text label that appears to the right of the icon. The **ListItem** objects are arranged vertically, each on its own line with information arranged in columns. |
| **lvwReport** | 3 | Report. Each **ListItem** object is displayed with its small icon and text labels. You can provide additional information about each **ListItem** object in a subitem. The icons, text labels, and information appear in columns with the leftmost column containing the small icon, followed by the text label. Additional columns display the text for each of the item's subitems. |

### Remarks

In Icon view only, use the **LabelWrap** property to specify if the **ListItem** object's labels are wrapped or not.

In Report view, you can hide the column headers by setting the **HideColumnHeaders** property to **True**. You can also use the ColumnClick event and sorting properties to sort the **ListItem** objects or subitems when a user clicks a column header. The user can change the size of the column by grabbing the right border of a column header and dragging it to the desired size.

**See Also**

ColumnClick Event
**HideColumnHeaders** Property
**LabelWrap** Property
**ListView** Control
**Sorted** Property (**ListView** Control)
**SortKey** Property
**SortOrder** Property

## FindItem Method

See Also Example

Finds and returns a reference to a **ListItem** object in a **ListView** control and returns a reference to that **ListItem**. Doesn't support named arguments.

**Important** This method requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**FindItem** (*string*, *value*, *index*, *match*)

The **FindItem** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **ListView** control. |
| *string* | Required. A string expression indicating the **ListItem** object to be found. |
| *value* | Optional. An integer or constant specifying whether the string will be matched to the **ListItem** object's **Text**, **Subitems**, or **Tag** property, as described in Settings. |
| *index* | Optional. An integer or string that uniquely identifies a member of an object collection and specifies the location from which to begin the search. The integer is the value of the **Index** property; the string is the value of the **Key** property. If no index is specified, the default is 1. |
| *match* | Optional. An integer or constant specifying that a match will occur if the item's **Text** property is the same as the string, as described in Settings. |

### Settings

The settings for *value* are:

| Constant | Value | Description |
|---|---|---|
| **lvwText** | 0 | (Default) Matches the string with a **ListItem** object's **Text** property. |
| **lvwSubitem** | 1 | Matches the string with any string in a **ListItem** object's **SubItems** property. |
| **lvwTag** | 2 | Matches the string with any **ListItem** object's **Tag** property. |

The settings for *match* are:

| Constant | Value | Description |
|---|---|---|
| **lvwWholeWord** | 0 | (Default) An integer or constant specifying that a match will occur if the item's **Text** property begins with the whole word being searched. Ignored if the criteria is not text. |
| **lvwPartial** | 1 | An integer or constant specifying that a match will occur if the item's **Text** property begins with the string being searched. Ignored if the criteria is not text. |

### Remarks

If you specify Text as the search criteria, you can use **lvwPartial** so that a match occurs when the **ListItem** object's **Text** property begins with the string you are searching for. For example, to find the **ListItem** whose text is "Autoexec.bat", use:

```
'Create a ListItem variable.
Dim itmX As ListItem
'Set the variable to the found item.
Set itmX = ListView1.FindItem("Auto",,,lvwpartial)
```

**See Also**

**Index** Property
**Key** Property
**ListView** Control
**SubItems** Property (**ListItems** Object)
**Tag** Property
**Text** Property

▪
## FindItem Method Example

This example populates a **ListView** control with the contents of the Publishers table of the BIBLIO.MDB database. A **ComboBox** control is also populated with three options for the **FindItem** method. A **CommandButton** contains the code for the **FindItem** method; when you click on the button, you are prompted to enter the string to search for, and the **FindItem** method searches the **ListView** control for the string. If the string is found, the control is scrolled using the **EnsureVisible** method to show the found **ListItem** object. To try the example, place a **ListView**, **ComboBox**, and a **CommandButton** control on a form and paste the code into the form's Declarations section. Run the example and click on the command button. Note: the example will not run unless you add a reference to the Microsoft DAO 3.0 Object Library by using the References command from the Tools menu.

```
Private Sub Form_Load()
   ' Create an object variable for the ColumnHeader object.
   Dim clmX As ColumnHeader
   ' Add ColumnHeaders.  The width of the columns is the width
   ' of the control divided by the number of ColumnHeader objects.
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Company", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Address", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Phone", ListView1.Width / 3)

   ListView1.BorderStyle = ccFixedSingle  ' Set BorderStyle property.
   ListView1.View = lvwReport  ' Set View property to Report.
   Command1.Caption = "&FindItem"

   ' Populate Combo1 to switch FindItem (FindWhere) arguments.
   With Combo1
      .AddItem "Text"               ' 0 Find in text.
      .AddItem "SubItem"            ' 1 Find in subitem.
      .AddItem "Tag"                ' 2 Find in tag.
      .ListIndex = 0                ' Set the ComboBox to show first item.
   End With

   ' Populate the ListView control with database records.
   ' Create object variables for the Data Access objects.
   Dim myDb As Database, myRs As Recordset
      ' Set the Database to the BIBLIO.MDB database.
   Set myDb = DBEngine.Workspaces(0).OpenDatabase("BIBLIO.MDB")
   ' Set the recordset to the Publishers table.
   Set myRs = myDb.OpenRecordset("Publishers", dbOpenDynaset)

   ' While the record is not the last record, add a ListItem object.
   ' Use the reference to the new object to set properties.
   ' Set the Text property to the Name field (myRS!Name).
   ' Set SubItem(1) to the Address field (myRS!Address).
   ' Set SubItem(7) to the Phone field (myRS!Telephone).

   While Not myRs.EOF
      Dim itmX As ListItem          ' A ListItem variable.
      Dim intCount As Integer       ' A counter variable.
      ' Use the Add method to add a new ListItem and set an object
      ' variable to the new reference. Use the reference to set
      ' properties.
      Set itmX = ListView1.ListItems.Add(, , CStr(myRs!Name))
      intCount = intCount + 1 ' Increment counter for the Tag property.
      itmX.Tag = "ListItem " & intCount  ' Set Tag with counter.

      ' If the Address field is not Null, set SubItem 1 to Address.
      If Not IsNull(myRs!Address) Then
         itmX.SubItems(1) = CStr(myRs!Address) ' Address field.
      End If

      ' If the Phone field is not Null, set SubItem 2 to Phone.
      If Not IsNull(myRs!Telephone) Then
         itmX.SubItems(2) = myRs!Telephone  ' Phone field.
      End If

      myRs.MoveNext                ' Move to next record.
   Wend
End Sub

Private Sub Command1_Click()
   ' FindItem method.
   ' Create a string variable called strFindMe. Use the InputBox
   ' to store the string to be found in the variable.  Use the
   ' FindItem method to find the string. Combo1 is used to
   ' switch the FindItem argument that determines where to look.

   Dim strFindMe As String
   strFindMe = InputBox("Find in " & Combo1.Text)
   ' FindItem method returns a reference to the found item, so
   ' you must create an object variable and set the found item
   ' to it.
   Dim itmFound As ListItem        ' FoundItem variable.

   Set itmFound = ListView1. _
   FindItem(strFindMe, Combo1.ListIndex, , lvwPartial)

   ' If no ListItem is found, then inform user and exit.  If a
   ' ListItem is found, scroll the control using the EnsureVisible
   ' method, and select the ListItem.
   If itmFound Is Nothing Then  ' If no match, inform user and exit.
      MsgBox "No match found"
      Exit Sub
   Else
        itmFound.EnsureVisible ' Scroll ListView to show found ListItem.
        itmFound.Selected = True    ' Select the ListItem.
      ' Return focus to the control to see selection.
        ListView1.SetFocus
   End If
End Sub

Private Sub ListView1_LostFocus()
   ' After the control loses focus, reset the Selected property
   ' of each ListItem to False.
   Dim i As Integer
   For i = 1 to ListView1.ListItems.Count
      ListView1.ListItems.Item(i).Selected = False
   Next i
```

```
End Sub
```

## GetFirstVisible Method

See Also Example

Retrieves a reference to the first **ListItem** object visible in the internal area of a **ListView** control.

---

**Important** This method requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**GetFirstVisible()**

The *object* placeholder represents an object expression that evaluates to a **ListView** control.

### Remarks

A **ListView** control can be scrolled to show more **ListItem** objects than can be seen in the internal area of the **ListView** control. You can use the reference to the first **ListItem** object in List and Report views, or the **Left** and **Top** properties of the first **ListItem** object in Icon and SmallIcon views, to determine the place to begin scrolling.

**See Also**

**EnsureVisible** Method
**Index** Property
**Key** Property
**Left**, **Top** Properties
**ListView** Control

▪

## GetFirstVisible Method Example

This example populates a **ListView** control with the contents of the Publishers table in the BIBLIO.MDB database.   When you click on the **CommandButton** control, the text of the first visible item is displayed.   Click on the column headers to change the **SortKey** property and click the **CommandButton** again.   To try the example, place a **ListView** and a **CommandButton** control on a form and paste the code into the form's Declaration section.   Also, the example will not run unless you add a reference to the Microsoft DAO 3.0 Object Library using the References command from the Tools menu.   Run the example.

```
Private Sub Command1_Click()
   ' Create a ListItem variable and set the variable to the object
   ' returned by the GetFirstVisible method.  Use the reference to
   ' display the text of the ListItem.
   Dim itmX As ListItem
   Set itmX = ListView1.GetFirstVisible
   MsgBox itmX.Text
End Sub

Private Sub Form_Load()
   ' Create an object variable for the ColumnHeader object.
   Dim clmX As ColumnHeader
   ' Add ColumnHeaders.  The width of the columns is the width
   ' of the control divided by the number of ColumnHeader objects.
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Company", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Address", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Phone", ListView1.Width / 3)

   ListView1.BorderStyle = ccFixedSingle ' Set BorderStyle property.

   ' Create object variables for the Data Access objects.
   Dim myDb As Database, myRs As Recordset
   ' Set the Database to the BIBLIO.MDB database.
   Set myDb = DBEngine.Workspaces(0).OpenDatabase("BIBLIO.MDB")
   ' Set the recordset to the Publishers table.
   Set myRs = myDb.OpenRecordset("Publishers", dbOpenDynaset)

   ' Create a variable to add ListItem objects.
   Dim itmX As ListItem

   ' While the record is not the last record, add a ListItem object.
   ' Use the Name field for the ListItem object's text.
   ' Use the Address field for the ListItem object's subitem(1).
   ' Use the Phone field for the ListItem object's subitem(2).

   While Not myRs.EOF

      Set itmX = ListView1.ListItems.Add(, , CStr(myRs!Name))

      ' If the Address field is not Null, set SubItem 1 to the field.
      If Not IsNull(myRs!Address) Then
         itmX.SubItems(1) = CStr(myRs!Address) ' Address field.
      End If
      ' If the Phone field is not Null, set the SubItem 2 to the field.
      If Not IsNull(myRs!Telephone) Then
         itmX.SubItems(2) = myRs!Telephone  ' Phone field.
      End If

      myRs.MoveNext                 ' Move to next record.
   Wend
   ListView1.View = lvwReport       ' Set view to Report.
End Sub

Private Sub ListView1_ColumnClick(ByVal ColumnHeader As ColumnHeader)
   ListView1.SortKey = ColumnHeader.Index - 1
   ListView1.Sorted = True
End Sub
```

## ColumnHeaders

The **ColumnHeaders** keyword is used in these contexts:
**ColumnHeaders** Collection
**ColumnHeaders** Property

## SubItems Property

See Also Example

Returns or sets an array of strings (a subitem) representing the **ListItem** object's data in a **ListView** control.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**SubItems(***index***)** [= *string*]

The **SubItems** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListItem** object. |
| *index* | An integer that identifies a subitem for the specified **ListItem**. |
| *string* | Text that describes the subitem. |

### Remarks

Subitems are arrays of strings representing the **ListItem** object's data and are displayed in Report view. For example, you could show the file size and the date last modified for a file.

A **ListItem** object can have any number of associated item data strings (subitems) but each **ListItem** object must have the same number of subitems. These strings become visible when the **ListView** control is in Report view.

There are corresponding column headers defined for each subitem.

You cannot add elements directly to the subitems array. Use the **Add** method of the **ColumnHeaders** collection to add subitems.

**See Also**

**Add** Method (**ColumnHeaders** Collection)
**Index** Property
**ListItem** Object, **ListItems** Collection
**Key** Property
**ListView** Control

## ListView Control Constants

See Also

### ListView Control Constants

| Constant | Value | Description |
|---|---|---|
| **lvwIcon** | 0 | (Default) Icon. Each **ListItem** object is represented by a full-sized (standard) icon and a text label. |
| **lvwSmallIcon** | 1 | SmallIcon. Each **ListItem** is represented by a small icon and a text label that appears to the right of the icon. The items appear horizontally. |
| **lvwList** | 2 | List. Each **ListItem** is represented by a small icon and a text label that appears to the right of the icon. Each **ListItem** appears vertically and on its own line with information arranged in columns. |
| **lvwReport** | 3 | Report. Each **ListItem** is displayed with its small icons and text labels. You can provide additional information about each **ListItem**. The icons, text labels, and information appear in columns with the leftmost column containing the small icon, followed by the text label. Additional columns display the text for each of the item's subitems. |

### ListArrange Constants

| Constant | Value | Description |
|---|---|---|
| **lvwNoArrange** | 0 | (Default) None. |
| **lvwAutoLeft** | 1 | Left. **ListItem** objects are aligned along the left side of the control. |
| **lvwTop** | 2 | Top. **ListItem** objects are aligned along the top of the control. |

### ListColumnAlignment Constants

| Constant | Value | Description |
|---|---|---|
| **lvwColumnLeft** | 0 | (Default) Left. Text is aligned left. |
| **lvwColumnRight** | 1 | Right. Text is aligned right. |
| **lvwColumnCenter** | 2 | Center. Text is centered. |

### ListLabelEdit Constants

| Constant | Value | Description |
|---|---|---|
| **lvwAutomatic** | 0 | (Default) Automatic. The BeforeLabelEdit event is generated when the user clicks the label of a selected node. |
| **lvwManual** | 1 | Manual. The BeforeLabelEdit event will be generated only when the **StartLabelEdit** method is invoked. |

### ListSortOrder Constants

| Constant | Value | Description |
|---|---|---|
| **lvwAscending** | 0 | (Default) Ascending order. Sorts from the beginning of the alphabet (A-Z), the earliest date, or the lowest number. |
| **lvwDescending** | 1 | Descending order. Sorts from the end of the alphabet (Z-A), the latest date, or the highest number. |

### ListFindItemWhere Constants

| Constant | Value | Description |
|---|---|---|
| **lvwText** | 0 | (Default) Text. Matches the string with a **ListItem** object's **Text** property. |
| **lvwSubItem** | 1 | SubItem. Matches the string with any string in a **ListItem** object's **SubItems** property. |
| **lvwTag** | 2 | Tag. Matches the string with any **ListItem** object's **Tag** property. |

### ListFindItemHow Constants

| Constant | Value | Description |
|---|---|---|
| **lvwWholeWord** | 0 | (Default) Whole word. Sets the search so that a match occurs if the item's **Text** property begins with the whole word being searched for. Ignored if the criteria is not text. |
| **lvwPartial** | 1 | Partial. Sets the search so that a match occurs if the item's **Text** property begins with the string being searched for. Ignored if the criteria is not text. |

**See Also**

**Arrange** Property
**Alignment** Property (**ColumnHeader** Object)
**FindItem** Method
**LabelEdit** Property
**SortOrder** Property
**View** Property
Visual Basic Custom Control Constants
Windows 95 Controls Constants

## Sorted Property (ListView Control)

See Also Example

Returns or sets a value that determines whether the **ListItem** objects in the Icon and SmallIcon views of a **ListView** control are sorted.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Sorted** [= *boolean*]

The **Sorted** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListView** control. |
| *boolean* | A Boolean expression specifying whether the **ListItem** objects are sorted, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | The list items are sorted alphabetically, according to the **SortOrder** property. |
| **False** | The list items are not sorted. |

### Remarks

The **Sorted** property must be set to **True** for the settings in the **SortOrder** and **SortKey** properties to take effect.

Each time the coordinates of a **ListItem** in the Icon and SmallIcon views change, the **Sorted** property becomes **False**.

**See Also**

**Height**, **Width** Properties
**Left**, **Top** Properties
**ListView** Control
**SortKey** Property
**SortOrder** Property

## ItemClick Event

See Also Example

Occurs when a **ListItem** object in a **ListView** control is clicked.

**Important** This event requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

**Private Sub***object*_**ItemClick(ByVal** *Item* **As ListItem)**

The **ItemClick** event syntax has these parts:

| **Part** | **Description** |
|---|---|
| *object* | An object expression that evaluates to a **ListView** control. |
| *listitem* | The **ListItem** object that was clicked. |

### Remarks

Use this event to determine which **ListItem** was clicked. This event is triggered before the **Click** event. The standard **Click** event is generated if the mouse is clicked on any part of the **ListView** control. The **ItemClick** event is generated only when the mouse is clicked on the text or image of a **ListItem** object.

**See Also**

Click Event
**ListItem** Object, **ListItems** Collection
**ListView** Control

▪

## ItemClick Event Example

This example populates a **ListView** control with contents of the Publishers table in the BIBLIO.MDB database. When a **ListItem** object is clicked, the code checks the value of the **Index** property. If the value is less than 15, nothing occurs. If the value is over 15, the **ListItem** object is ghosted. To try the example, place a **ListView** control on a form and paste the code into the form's Declarations section. Run the example and click on one of the items.

```
Private ListView1_ItemClick(ByVal Item As ListItem)
   Select Case Item.Index
   Case Is = <15
      Exit Sub
   Case Is => 15
      ' Toggle Ghosted property.
      Item.Ghosted = Abs(Item.Ghosted) - 1
   End Select
End Sub

Private Sub Form_Load()
   ' Create an object variable for the ColumnHeader object.
   Dim clmX As ColumnHeader
   ' Add ColumnHeaders.  The width of the columns is the width
   ' of the control divided by the number of ColumnHeader objects.
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Company", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Address", ListView1.Width / 3)
   Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Phone", ListView1.Width / 3)

   ListView1.BorderStyle = ccFixedSingle ' Set BorderStyle property.

   ' Create object variables for the Data Access objects.
   Dim myDb As Database, myRs As Recordset
   ' Set the Database to the BIBLIO.MDB database.
   Set myDb = DBEngine.Workspaces(0).OpenDatabase("BIBLIO.MDB")
   ' Set the recordset to the Publishers table.
   Set myRs = myDb.OpenRecordset("Publishers", dbOpenDynaset)

   ' Create a variable to add ListItem objects.
   Dim itmX As ListItem

   ' While the record is not the last record, add a ListItem object.
   ' Use the Name field for the ListItem object's text.
   ' Use the Address field for the ListItem object's SubItem(1).
   ' Use the Phone field for the ListItem object's SubItem(2).

   While Not myRs.EOF

      Set itmX = ListView1.ListItems.Add(, , CStr(myRs!Name))

      ' If the Address field is not Null, set SubItem 1 to the field.
      If Not IsNull(myRs!Address) Then
         itmX.SubItems(1) = CStr(myRs!Address) ' Address field.
      End If
      ' If the Phone field is not Null, set the SubItem 2 to the field.
      If Not IsNull(myRs!Telephone) Then
         itmX.SubItems(2) = myRs!Telephone  ' Phone field.
      End If

      myRs.MoveNext ' Move to next record.
   Wend
   ListView1.View = lvwReport      ' Set View to Report.
End Sub

Private Sub ListView1_ColumnClick(ByVal ColumnHeader As ColumnHeader)
   ListView1.SortKey = ColumnHeader.Index - 1
   ListView1.Sorted = True
End Sub
```

## ListItems

The **ListItems** keyword is used in these contexts:

**ListItems** Collection

**ListItems** Property

## ProgressBar Control

See Also Properties Methods Events

The **ProgressBar** control shows the progress of a lengthy operation by filling a rectangle with chunks from left to right.

### Syntax

**ProgressBar**

### Remarks

The **ProgressBar** control monitors an operation's progress toward completion. It functions like the **Gauge** control, but without the same precision.

- **ProgressBar**
- fills in chunks that approximate the relative progress of an operation.
- **Gauge**
- fills continuously and precisely measures a value, such as how much memory remains.

You can use the **Align** property with the **ProgressBar** control to automatically position it at the top or bottom of the form.

A **ProgressBar** control has a range and a current position. The range represents the entire duration of the operation. The current position represents the progress the application has made toward completing the operation. The **Max** and **Min** properties set the limits of the range. The **Value** property specifies the current position within that range. Because chunks are used to fill in the control, the amount filled in only approximates the **Value** property's current setting. Based on the control's size, the **Value** property determines when to display the next chunk.

The **ProgressBar** control's **Height** and **Width** properties determine the number and size of the chunks that fill the control. The more chunks, the more accurately the control portrays an operation's progress. To increase the number of chunks displayed, decrease the control's **Height** or increase its **Width**. The **BorderStyle** property setting also affects the number and size of the chunks. To accommodate a border, the chunk size becomes smaller.

---

**Tip** For a chunk size that best shows incremental progress, make a **ProgressBar** control at least 12 times wider than its height.

---

The following example shows how to use the **ProgressBar** control, named ProgressBar1, to show the progress of a lengthy operation of a large array. Put a **CommandButton** control and a **ProgressBar** control on a form. The **Align** property in the sample code positions the **ProgressBar** control along the bottom of the form. The **ProgressBar** control displays no text.

```
Private Sub Command1_Click()
   Dim Counter As Integer
   Dim Workarea(250) As String
   ProgressBar1.Min = LBound(Workarea)
   ProgressBar1.Max = UBound(Workarea)
   ProgressBar1.Visible = True

'Set the Progress's Value to Min.
   ProgressBar1.Value = ProgressBar1.Min

'Loop through the array.
   For Counter = LBound(Workarea) To UBound(Workarea)
      'Set initial values for each item in the array.
      Workarea(Counter) = "Initial value" & Counter
      ProgressBar1.Value = Counter
   Next Counter
   ProgressBar1.Visible = False
   ProgressBar1.Value = ProgressBar1.Min
End Sub

Private Sub Form_Load()
   ProgressBar1.Align = vbAlignBottom
   ProgressBar1.Visible = False
   Command1.Caption = "Initialize array"
End Sub
```

**Distribution Note** The **ProgressBar** control is a 32-bit custom control that can only run on Windows 95 and Windows NT 3.51 or higher. Additionally, the **ProgressBar** control is part of a group of custom controls that are found in the COMCTL32.OCX file. To use the **ProgressBar** control in your application, you must add the COMCTL32.OCX file to the project. When distributing your application, install the COMCTL32.OCX file in the user's Microsoft Windows SYSTEM directory. For more information on how to add a custom control to a project, see the *Programmer's Guide*.

**See Also**

**Align** Property
**BorderStyle** Property
**Gauge** Control
**Height, Width** Properties
**Max, Min** Properties (**ProgressBar, Slider** Controls)
**Value** Property (**ProgressBar, Slider** Controls)

■
## ProgressBar Control Properties

**Align** Property
**Appearance** Property
**BorderStyle** Property
**Container** Property
**DragIcon** Property
**DragMode** Property
**Enabled** Property
**Height** Property
**hWnd** Property
**Index** Property
**Left** Property
**Max** Property
**Min** Property
**MouseIcon** Property
**MousePointer** Property
**Name** Property
**Negotiate** Property
**Object** Property
**Parent** Property
**TabIndex** Property
**Tag** Property
**Top** Property
**Value** Property (**ProgressBar, Slider** Controls)
**Visible** Property
**WhatsThisHelpID** Property
**Width** Property

■
## ProgressBar Control Methods

**Drag** Method
**Move** Method
**ShowWhatsThis** Method
**ZOrder** Method

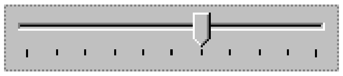
■
## ProgressBar Control Events

Click Event
DragDrop Event
DragOver Event
MouseDown Event
MouseMove Event
MouseUp Event

## Slider Control

Properties Methods Events Constants

A **Slider** control is a window containing a slider and optional tick marks. You can move the slider by dragging it, clicking the mouse to either side of the slider, or using the keyboard.



### Syntax

**Slider**

### Remarks

**Slider** controls are useful when you want to select a discrete value or a set of consecutive values in a range. For example, you could use a **Slider** to set the size of a displayed image by moving the slider to a given tick mark rather than by typing a number. To select a range of values, set the **SelectRange** property to **True**, and program the control to select a range when the SHIFT key is down.

The **Slider** control can be oriented either horizontally or vertically.

---

**Distribution Note** The **Slider** control is a 32-bit custom control that can only run on Windows 95 and Windows NT 3.51 or higher. Additionally, the **Slider** control is part of a group of custom controls that are found in the COMCTL32.OCX file. To use the **Slider** control in your application, you must add the COMCTL32.OCX file to the project. When distributing your application, install the COMCT32.OCX file in the user's Microsoft Windows SYSTEM directory. For more information on how to add a custom control to a project, see the *Programmer's Guide*.

---

■

**Slider Control Properties**

**BorderStyle** Property
**Container** Property
**DragIcon** Property
**DragMode** Property
**Enabled** Property
**Height** Property
**HelpContextID** Property
**hWnd Property**
**Index** Property
**LargeChange** Property
**Left** Property
**Max** Property
**Min** Property
**MouseIcon** Property
**MousePointer** Property
**Name** Property
**Orientation** Property
**Parent** Property
**SelectRange** Property
**SelLength** Property
**SelStart** Property
**SmallChange** Property
**TabIndex** Property
**TabStop** Property
**Tag** Property
**TickFrequency** Property
**TickStyle** Property
**Top** Property
**Value** Property
**Visible** Property
**WhatsThisHelpID** Property
**Width** Property

■
## Slider Control Methods

**ClearSel** Method
**Drag** Method
**GetNumTicks** Method
**Move** Method
**Refresh** Method
**SetFocus** Method
**ShowWhatsThis** Method
**ZOrder** Method

■

## Slider Control Events

- Change Event
- Click Event
- DragDrop Event
- DragOver Event
- GotFocus Event
- KeyDown Event
- KeyPress Event
- KeyUp Event
- LostFocus Event
- MouseDown Event
- MouseMove Event
- MouseUp Event
- Scroll Event

## LargeChange, SmallChange Properties (Slider Control)

See Also Example

- The **LargeChange** property sets the number of ticks the slider will move when you press the PAGEUP or PAGEDOWN keys, or when you click the mouse to the left or right of the slider.
- The **SmallChange** property sets the number of ticks the slider will move when you press the left or right arrow keys.

### Syntax

*object*.**LargeChange** = *number*

*object*.**SmallChange** = *number*

The **LargeChange** and **SmallChange** property syntaxes have these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Slider** control. |
| *number* | A long integer specifying how many ticks the slider moves. |

### Remarks

The default for the **LargeChange** property is 5. The default for the **SmallChange** property is 1.

**See Also**

**Max, Min** Properties
**Value** Property
**Slider** Control

■

## LargeChange, SmallChange Properties Example

This example matches a **TextBox** control's width to that of a **Slider** control. While the **Slider** control's **Value** property is above a certain value, the **TextBox** control's width matches the **Slider** control's value. The **SmallChange** and **LargeChange** properties depend on the value of the **Slider** control's **Max** property. To try the example, place a **Slider** control and a **TextBox** control on a form and paste the code into the form's Declarations section. Run the example and press the PAGEDOWN, PAGEUP, and LEFT and RIGHT ARROW keys.

```
Sub Form_Load()
   Text1.Width = 4500 ' Set a minimum width for the TextBox.
   Slider1.Left = Text1.Left ' Align the Slider to the TextBox.
   ' Match the width of the Slider to the TextBox.
   Slider1.Max = Text1.Width
   ' Place the Slider a little below the Textbox.
   Slider1.Top = Text1.Top + 600
   ' Set TickFrequency to a fraction of the Max value.
   Slider1.TickFrequency = Slider1.Max * 0.1
   ' Set LargeChange and SmallChange value to a fraction of Max.
   Slider1.LargeChange = Slider1.Max * 0.1
   Slider1.SmallChange = Slider1.Max * 0.01
End Sub

Private Sub Slider1_Change()
   ' If the slider is under 1/3 the size of the textbox, no change.
   ' Else, match the width of the textbox to the Slider's value.
   If Slider1.Value > Slider1.Max / 3 Then
      Text1.Width = Slider1.Value
   End If
End Sub
```

## Orientation Property (Slider Control)

See Also Example

Sets a value that determines whether the **Slider** control is oriented horizontally or vertically.

### Syntax

*object*.**Orientation** = *number*

The **Orientation** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Slider** control. |
| *number* | A constant or value specifying the orientation, as described in Settings. |

### Settings

The settings for *number* are:

| Constant | Value | Description |
|---|---|---|
| **sldHorizontal** | 0 | (Default) Horizontal. The slider moves horizontally and tick marks can be placed on either the top or bottom, both, or neither. |
| **sldVertical** | 1 | Vertical. The slider moves vertically and tick marks can be placed on either the left or right sides, both, or neither. |

**See Also**

**Max, Min**Properties

**Slider** Control

**Slider** Control Constants

▪

## Orientation Property Example

This example toggles the orientation of a **Slider** control on a form. To try the example, place a **Slider** control onto a form and paste the code into the form's Declarations section, and then run the example. Click the form to toggle the **Slider** control's orientation.

```
Private Sub Form_Click()
   If Slider1.Orientation = 0 Then
      Slider1.Orientation = 1
   Else
      Slider1.Orientation = 0
   End If
End Sub
```

## Scroll Event (Slider Control)

See Also

Occurs when you move the slider on a **Slider** control, either by clicking on the control or using keyboard commands.

### Syntax

**Private Sub** *object*_**Scroll( )**

The object placeholder represents an object expression that evaluates to a **Slider** control.

### Remarks

The Scroll Event occurs before the Click event.

The Scroll Event continuously returns the value of the **Value** property as the slider is moved.   You can use this event to perform calculations to manipulate controls that must be coordinated with ongoing changes in the **Slider** control.   In contrast, use the Change event when you want an update to occur only once, after a **Slider** control's **Value** property has changed.

---

**Note**    Avoid using a **MsgBox** statement or function in this event.

---

**See Also**

**LargeChange**, **SmallChange** Properties
**Value** Property
**Slider** Control

## SelectRange Property

See Also Example

Sets a value that determines if a **Slider** control can have a selected range.

### Syntax

*object*.**SelectRange** = *boolean*

The **SelectRange** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Slider** control. |
| *boolean* | A Boolean expression that determines whether or not the **Slider** can have a selected range, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | The **Slider** can have a selected range. |
| **False** | The **Slider** can't have a selected range. |

### Remarks

If **SelectRange** is set to **False**, then the **SelStart** property setting is the same as the **Value** property setting. Setting the **SelStart** property also changes the **Value** property, and vice-versa, which will be reflected in the position of the slider on the control. Setting **SelLength** when the **SelectRange** property is **False** has no effect.

**See Also**

**ClearSel** Method
**SelLength, SelStart** Properties
**Slider** Control
**Value** Property

▪

## SelectRange Property Example

This example allows the user to select a range when the SHIFT key is held down.   To try the example, place a **Slider** control on a form and paste the code into the form's Declarations section.   Run the example and select a range by holding down the SHIFT key and dragging or clicking the mouse on the **Slider** control.

```
Private Sub Form_Load()
   'Set slider control settings
   Slider1.Max = 20
End Sub

Private Sub Slider1_MouseDown(Button As Integer, Shift As Integer, x As
Single, y As Single)
   If Shift = 1 Then                 ' If Shift button is down then
      Slider1.SelectRange = True     ' turn SelectRange on.
      Slider1.SelStart = Slider1.Value  ' Set the SelStart value
      Slider1.SelLength = 0 ' Set previous SelLength (if any) to 0.
   Else
      Exit Sub
   End If
End Sub

Private Sub Slider1_MouseUp(Button As Integer, Shift As Integer, x As Single,
y As Single)

   If Shift = 1 Then
   ' If user selects backwards from a point, an error will occur.
   On Error Resume Next
   ' Else set SelLength using SelStart and current value.
      Slider1.SelLength = Slider1.Value - Slider1.SelStart
   Else
      Slider1.SelectRange = False ' If user lifts SHIFT key.
   Exit Sub
   End If
End Sub
```

## SelLength, SelStart Properties (Slider Control)

See Also Example

- **SelLength** returns or sets the length of a selected range in a **Slider** control.
- **SelStart** returns or sets the start of a selected range in a **Slider** control.

### Syntax

*object*.**SelLength** [= *value*]

*object*.**SelStart** [= *value*]

The **SelLength** and **SelStart** property syntaxes have these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Slider** control. |
| *value* | A value that falls within the **Min** and **Max** properties. |

### Remarks

The **SelLength** and **SelStart** properties are used together to select a range of contiguous values on a **Slider** control.   The **Slider** control then has the additional advantage of being a visual analogue of the range of possible values.

The **SelLength** property can't be less than 0, and the sum of **SelLength** and **SelStart** can't be greater than the **Max** property.

**See Also**

**ClearSel** Method
**SelectRange** Property
**Slider** Control

▪

## SelLength, SelStart Properties Example

This example selects a range on a **Slider** control. To try this example, place a **Slider** control onto a form with three **TextBox** controls, named Text1, Text2, and Text3. The **Slider** control's **SelectRange** property must be set to **True**. Paste the code below into the form's Declarations section, and run the example. While holding down the SHIFT key, you can select a range on the slider, and the various values will be displayed in the text boxes.

```
Private Sub Form_Load()
   ' Make sure SelectRange is True so selection can occur.
   Slider1.SelectRange = True
End Sub

Private Sub Slider1_MouseDown(Button As Integer, Shift As Integer, x As
Single, y As Single)
   If Shift = 1 Then ' If SHIFT is down, begin the range selection.
      Slider1.ClearSel                ' Clear any previous selection.
      Slider1.SelStart = Slider1.Value
      Text2.Text = Slider1.SelStart    ' Show the beginning
                                      ' of the range in the textbox.
   Else
      Slider1.ClearSel                ' Clear any previous selection.
End If
End Sub

Private Sub Slider1_MouseUp(Button As Integer, Shift As Integer, x As Single,
y As Single)
   ' When SHIFT is down and SelectRange is True,
   ' this event is triggered.
   If Shift = 1 And Slider1.SelectRange = True Then
      ' Make sure the current value is larger than SelStart or
      ' an error will occur--SelLength can't be negative.
      If Slider1.Value >= Slider1.SelStart Then
         Slider1.SelLength = Slider1.Value - Slider1.SelStart
         Text1.Text = Slider1.Value  ' To see the end of the range.
         ' Text3 is the difference between the end and start values.
         Text3.Text = Slider1.SelLength
      End If
   End If
End Sub
```

## TickFrequency Property

See Also        Example

Returns or sets the frequency of tick marks on a **Slider** control in relation to its range.   For example, if the range is 100, and the **TickFrequency** property is set to 2, there will be one tick for every 2 increments in the range.

### Syntax

*object*.**TickFrequency** [= *number*]

The **TickFrequency** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Slider** control. |
| *number* | A numeric expression specifying the frequency of tick marks. |

**See Also**

**GetNumTicks** Method
**Max, Min** Properties
**Slider** Control

■

## TickFreqency Property Example

This example matches a **TextBox** control's width to that of a **Slider** control. While the **Slider** control's **Value** property is above a certain value, the **TextBox** control's width matches the **Slider** control's value. The **TickFrequency** depends on the value of the **Slider** control's **Max** property. To try the example, place a **Slider** and a **TextBox** control on a form and paste the code into the form's Declarations section. Run the example and click the slider several times.

```
Sub Form_Load()
   Text1.Width = 4500 ' Set a minimum width for the TextBox.
   Slider1.Left = Text1.Left ' Align the Slider to the TextBox.
   ' Match the width of the Slider to the TextBox.
   Slider1.Max = Text1.Width
   ' Place the Slider a little below the Textbox.
   Slider1.Top = Text1.Top + 600
   ' Set TickFrequency to a fraction of the Max value.
   Slider1.TickFrequency = Slider1.Max * 0.1
   ' Set LargeChange and SmallChange value to a fraction of Max.
   Slider1.LargeChange = Slider1.Max * 0.1
   Slider1.SmallChange = Slider1.Max * 0.01
End Sub

Private Sub Slider1_Change()
   ' If the slider is under 1/3 the size of the textbox, no change.
   ' Else, match the width of the textbox to the Slider's value.
   If Slider1.Value > Slider1.Max / 3 Then
      Text1.Width = Slider1.Value
   End If
End Sub
```

## TickStyle Property

See Also Example

Returns or sets the style (or positioning) of the tick marks displayed on the **Slider** control.

### Syntax

*object*.**TickStyle** [= *number*]

The **TickStyle** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Slider** control. |
| *number* | A constant or integer that specifies the **TickStyle** property, as described in Settings. |

### Settings

The settings for *number* are:

| Constant | Value | Description |
|---|---|---|
| **sldBottomRight** | 0 | (Default) Bottom/Right. Tick marks are positioned along the bottom of the **Slider** if the control is oriented horizontally, or along the right side if it is oriented vertically. |
| **sldTopLeft** | 1 | Top/Left. Tick marks are positioned along the top of the **Slider** if the control is oriented horizontally, or along the left side if it is oriented vertically. |
| **sldBoth** | 2 | Both. Tick marks are positioned on both sides or top and bottom of the **Slider**. |
| **sldNoTicks** | 3 | None. No tick marks appear on the **Slider**. |

**See Also**

**GetNumTicks** Method
**Max, Min** Properties
**Slider** Control
**Slider** Control Constants

■

## TickStyle Property Example

This example allows you to see the various tick styles available in a drop-down list. To try the example, place a **Slider** control and a **ComboBox** control on a form. Paste the code into the Declarations section of the form, and run the example. Click on the **ComboBox** to change the **TickStyle** property value.

```
Sub Form_Load()
   With combo1
      .AddItem "Bottom/Right"
      .AddItem "Top/Left"
      .AddItem "Both"
      .AddItem "None"
      .ListIndex = 0
   End With
End Sub

Private Sub combo1_Click()
   Slider1.TickStyle = combo1.ListIndex
End Sub
```

## ClearSel Method

See Also

Clears the current selection of a **Slider** control.

### Syntax

*object*.**ClearSel**

The object placeholder represents an object expression that evaluates to a **Slider** control.

### Remarks

This method sets the **SelStart** property to the value of the **Value** property and sets the **SelLength** property to 0.

**See Also**

- **SelectRange** Property
- **SelLength, SelStart** Properties (**Slider** Control)
- **Slider** Control
- **Value** Property

## GetNumTicks Method

See Also Example

Returns the number of ticks between the **Min** and **Max** properties of the **Slider** control.

### Syntax

*object*.**GetNumTicks**

The object placeholder represents an object expression that evaluates to a **Slider** control.

### Remarks

To change the number of ticks, reset the **Min** or **Max** properties or the **TickFrequency** property.

**See Also**

**Max, Min** Properties
**TickFrequency** Property
**Slider** Control

■

## GetNumTicks Method Example

This example displays the current number of ticks on a **Slider** control, then increments the **Max** property by 10.   To try this example, place a **Slider** control onto a form and paste the code into the form's Declarations section.   Run the example, and click the **Slider** control to get the number of ticks.   Every click on the control increases the ticks.

```
Sub Slider1_Click()
   MsgBox Slider1.GetNumTicks
   Slider1.Max = Slider1.Max + 10
End Sub
```

## Slider Constants

See Also

### Orientation Constants

| Constant | Value | Description |
|---|---|---|
| **sldHorizontal** | 0 | Horizontal orientation. |
| **sldVertical** | 1 | Vertical orientation. |

### TickStyle Constants

| Constant | Value | Description |
|---|---|---|
| **sldBottomRight** | 0 | Bottom/Right. Tick marks are positioned along the bottom of the **Slider** if the control is oriented horizontally, or along the right side if it is oriented vertically. |
| **sldTopLeft** | 1 | Top/Left. Tick marks are positioned along the top of the **Slider** if the control is oriented horizontally, or along the left side if it is oriented vertically. |
| **sldBoth** | 2 | Both. Tick marks are positioned on both sides or top and bottom of the **Slider**. |
| **sldNoTicks** | 3 | None. No tick marks appear on the **Slider**. |

**See Also**

**TickStyle** Property
**Orientation** Property
**Slider** Control
Visual Basic Custom Control Constants
Windows 95 Controls Constants

## Image Property (Custom Controls)

See Also

Returns or sets a value that specifies which **ListImage** object in a **ImageList** control to use with another object.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Image** [= *index*]

The **Image** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Button**, **Node**, or **Tab** object. |
| *index* | An integer or unique string specifying the **ListImage** object to use with *object*. The integer is the value of the **Index** property; the string is the value of the **Key** property. |

### Remarks

Before setting the **Image** property, you must associate an **ImageList** control with a **Toolbar**, **TreeView**, or **TabStrip** control by setting each control's **ImageList** property to an **ImageList** control.

At design time, put an **ImageList** control on the form and load it with images, each of which is a **ListImage** object assigned an index number in a **ListImages** collection. On the General tab in the control's properties dialog box, select the **ImageList** you want from the **ImageList** list box, such as `ImageList1`. For **Tab** and **Button** objects, you can also specify the image you want to associate with these objects by typing the index number of the specific **ListImage** in the Image field on the Tabs or Buttons tab.

At run time, use code like the following to associate an **ImageList** to a control and then a **ListImage** to a specific object:

```
Set TabStrip1.ImageList=ImageList1
TabStrip1.Tabs(1).Image=2
```

Use the **Key** property to specify an **ImageList** control's **ListImage** object when you wish your code to be self-documenting, as follows:

```
' Assuming there is a ListImage object with the Key property value =
' "close," use that image for a Toolbar button.
Toolbar1.Buttons(1).Image = "close"

' This is easier to read than just specifying an Index value, as below:
Toolbar1.Buttons(1).Image = 4 ' Requires that the ListImage object
' with Index property = 4 is the "close" image.
```

Additionally, when you use the **Key** property to specify a **ListImage** object, you can ignore the **Index** property, which may change if **ListImage** objects are added or deleted from a collection.

If there are no images for a **Tabs** collection, the value of *index* is -1.

**See Also**

**Add** Method (**ListImages** Collection)
**ImageList** Control
**Index** Property
**Key** Property

## Index Property (Custom Controls)

See Also

Returns or sets the number that uniquely identifies an object in a collection.

### Syntax

*object*.**Index**

The object placeholder is an object expression that evaluates to a **Button**, **ColumnHeader**, **ListImage**, **ListItem**, **Node**, **Panel**, or **Tab** object.

**See Also**

**Key** Property

## Key Property (Custom Controls)

See Also

Returns or sets a string that uniquely identifies a member in a collection.

### Syntax

*object*.**Key** [= *string*]

The **Key** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Button, ColumnHeader, ListImage, ListItem, Node, Panel,** or **Tab** object. |
| *string* | A unique string identifying a member in a collection. |

### Remarks

If the string is not unique, an error will occur.

You can set the **Key** property when you use the **Add** method to add an object to a collection.

**See Also**

**Add** Method
**Index** Property

## Max, Min Properties (Custom Controls)

- **Max**

- returns or sets a control's maximum value.
- **Min**

- returns or sets a control's minimum value.

**Syntax**

*object*.**Max** [= *integer*]

*object*.**Min** [= *integer*]

The **Max** and **Min** properties syntaxes have these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ProgressBar** or **Slider** control. |
| *integer* | An integer specifying the maximum or minimum value. |

**Remarks**

The **Max** and **Min** properties define the range of a control.

Setting the **Min** property greater than the **Max** property produces an error.

For the **ProgressBar** control, you can't set the **Min** property equal to the **Max** property, and the **Min** property must be greater than or equal to 0.  By default, the **ProgressBar** control sets the **Max** property to 100 and the **Min** property to 0.  This range represents the duration of the operation.

## ToolTipText Property (Custom Controls)

See Also

Returns or sets a ToolTip.

### Syntax

*object*.**ToolTipText** [= *string*]

The **ToolTipText** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Button** or **Tab** object. |
| *string* | A string associated with a **Tab** or **Button** that appears in a small rectangle below the object when the user's cursor hovers over the object at run time for about one second. |

### Remarks

If you use only an image to label an object, you can use this property to explain each object with a few words.   The **ShowTips** property must be set to **True** for a **ToolTipText** property string to appear with the object at run time.

At design time you can set the **ToolTipText** property string on the Buttons (**Toolbar**) or Tabs (**TabStrip**) tab in the control's properties dialog box.

**See Also**

**ShowTips** Property

## ShowTips Property (Custom Controls)

See Also

Returns or sets a value that determines whether ToolTips are displayed for an object.

### Syntax

*object*.**ShowTips** [= *value*]

The **ShowTips** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TabStrip** or **Toolbar** control. |
| *value* | A Boolean expression specifying whether ToolTips are displayed, as described in Settings. |

### Settings

The settings for *value* are:

| Setting | Description |
|---|---|
| **True** | (Default) Each object in the control may display an associated string, which is the setting of the **ToolTipText** property, in a small rectangle below the object.   This ToolTip appears when the user's cursor hovers over the object at run time for about one second. |
| **False** | An object will not display a ToolTip at run time. |

### Remarks

At design time you can set the **ShowTips** property on the General tab in the control's properties dialog box.

**See Also**

**ToolTipText** Property

## ImageList Property (Custom Controls)

See Also

Returns or sets the **ImageList** control, if any, that is associated with another control**.**

### Syntax

*object*.**ImageList** [= *imagelist*]

The **ImageList** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TabStrip**, **Toolbar**, or **TreeView** control. |
| *imagelist* | An object reference that specifies which **ImageList** control to use. |

### Remarks

For the control to use the **ImageList** property, you must put an **ImageList** control on the form.   Then, at design time, you can set the **ImageList** property in the associated control's properties dialog box from the combo box containing the names of all the **ImageList** controls currently on the form.   To associate an **ImageList** with a control at run time, set the control's **ImageList** property to the **ImageList** control you want to use, as in this example:

```
Set TabStrip1.ImageList = ImageList1
```

**See Also**

**ImageList** Control

## Clear Method (Custom Controls)

See Also Example

Clears all objects in a collection.

### Syntax

*object*.**Clear**

The *object* placeholder represents an object expression that evaluates to one of the following collections: **Buttons**, **ColumnHeaders**, **ListImages**, **ListItems**, **Nodes**, **Panels**, or **Tabs**.

### Remarks

To remove one object from a collection, use the **Remove** method.

**See Also**

**Remove** Method

■

## Clear Method (Custom Controls) Example

This example adds six **Panel** objects to a **StatusBar** control, creating a total of seven **Panel** objects. A click on the form clears all **Panel** objects when their number reaches seven. If the number of **Panel** objects is less than seven, each click on the form will add a new **Panel** object to the control until the number seven is once again reached. To try the example, place a **StatusBar** control on a form and paste the code into the Declarations section. Run the example and click on the form to clear all **Panel** objects and subsequently add **Panel** objects.

```
Private Sub Form_Load()
   Dim pnlX As Panel ' Declare object variable for Panel objects.
   Dim I As Integer

   ' Add 6 Panel objects to the single default Panel object,
   ' making 7 Panel objects.
   For I = 1 to 6
      Set pnlX = StatusBar1.Panels.Add
   Next I
End Sub

Private Sub Form_Click()
   ' If the Count of the collection is 7, then clear the collection.
   ' Otherwise, add one Panel and use the collection's Count property
   ' to set its Style.
   If StatusBar1.Panels.Count = 7 Then
      StatusBar1.Panels.Clear
   Else
      Dim pnlX As Panel
      Set pnlX = StatusBar1.Panels.Add( , , "simple", 0)
      ' The Style property is enumerated from 0 to 6.  Use the Panels
      ' Count property -1 to set the Style property for the new Panel.
      ' Display all panels regardless of form width.
      pnlX.minwidth = TextWidth("simple")
      pnlX.AutoSize = sbrSpring
      pnlX.Style = Statusbar1.Panels.Count - 1
   End If
End Sub
```

## Remove Method (Custom Controls)

See Also Example

Removes a specific member from a collection.

### Syntax

*object*.**Remove** *index*

The **Remove** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to one of the following collections: **Buttons**, **ColumnHeaders**, **ListImages**, **ListItems**, **Nodes**, **Panels, Tabs**. |
| *index* | An integer or string that uniquely identifies the object in the collection. An integer specifies the value of the **Index** property; a string specifies the value of the **Key** property. |

### Remarks

To remove all the members of a collection, use the **Clear** method.

**See Also**

**Clear** Method
**Index** Property
**Key** Property

■

## Remove Method (Custom Controls) Example

This example adds six **Panel** objects to a **StatusBar** control, creating a total of seven **Panel** objects. When you click on the form, the code checks to see how many **Panel** objects there are.   If there is only one **Panel** object, the code adds six **Panel** objects.   Otherwise, it removes the first panel.   To try the example, place a **StatusBar** control on a form and paste the code into the Declarations section.   Run the example and click on the form to remove one **Panel** object at a time, and subsequently add **Panel** objects.

```
Private Sub Form_Load()
   Dim pnlX As Panel    ' Declare object variable for Panel objects.
   Dim i As Integer

   ' Add 6 Panel objects to the single default Panel object,
   ' making 7 Panel objects.
   For i = 1 To 6
      Set pnlX = StatusBar1.Panels.Add(, , , i)
      pnlX.AutoSize = sbrSpring
   Next i
End Sub

Private Sub Form_Click()
   ' If the Count of the collection is 1, add 6 Panel objects.
   ' Otherwise, remove the first panel from the collection.
   If StatusBar1.Panels.Count = 1 Then
      Dim sbrX As Panel
      Dim i As Integer
      For i = 1 To 6 ' Each panel has its style set by i.
         Set sbrX = StatusBar1.Panels.Add(, , , i)
         sbrX.AutoSize = sbrSpring
      Next i
   Else ' Remove the first panel.
      StatusBar1.Panels.Remove 1
   End If
End Sub
```

## Value Property (Custom Controls)

See Also Example

Returns or sets the value of an object. See Remarks for more specific information.

**Syntax**

*object*.**Value** [= *integer*]

The **Value** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a Button object, 3D check box, 3D command button, 3D group push button, 3D option button, Slider, or ProgressBar control. |
| *integer* | For a Slider control, a long integer that specifies the current position of the slider. For the ProgressBar control, an integer that specifies the value of the ProgressBar control. For other controls, see Settings below. |

**Settings**

For the Button object, the settings for *integer* are:

| Constant | Value | Description |
|---|---|---|
| **tbrPressed** | 0 | (Default). The button is not currently pressed or checked. |
| **tbrUnpressed** | 1 | The button is currently pressed or checked. |

For the 3D check box, 3D command button, and 3D group push button controls, the settings for *integer* are:

| Value | Description |
|---|---|
| **True** | The button is pressed. |
| **False** | (Default). The button is not pressed. |

For the 3D option button control, the settings for *integer* are:

| Value | Description |
|---|---|
| **True** | The button is selected. |
| **False** | (Default). The button is not selected. |

**Remarks**

■ Slider control
■returns or sets the current position of the slider. **Value** is always between the values for the **Max** and **Min** properties, inclusive, for a Slider control.
■ ProgressBar
■returns or sets a value indicating an operation's approximate progress toward completion. Incrementing the **Value** property doesn't change the appearance of the ProgressBar control by the exact value of the **Value** property. **Value** is always in the range between the values for the **Max** and **Min** properties, inclusive. Not available at design time.
■ 3D command button control
■returns or sets a value indicating whether the button is chosen; not available at design time. Setting the **Value** property to **True** in code invokes the button's Click event.

**See Also**

**Max**, **Min** Properties

■
## Value Property Example

This example uses the **Value** property to determine which icon from an associated **ImageList** control is displayed on the **Toolbar** control.   To try the example, place a **Toolbar** control on a form and paste the code into the form's Declarations section.   Then run the example.

```
Private Sub Toolbar1_ButtonClick(ByVal Button As Button)
   ' Use the Key value to determine which button has been clicked.
   Select Case Button.Key

   Case Is "Done"                    ' A check button.
      If Button.Value = 0 Then       ' The button is unchecked.
         Button.Value = 1            ' Check the button.
         ' Assuming there is a ListImage object with key "down."
         Button.Icon = "down"
      Else
         Button.Value = 0            ' Uncheck the button
         ' Assuming there is a ListImage object with key " up."
         Button.Icon = "up"
      End If

   ' More Cases are possible.
   End Select
End Sub
```

## HideSelection Property (Custom Controls)

Returns or sets a value that specifies if the selected item remains highlighted when a control loses focus.

### Syntax

*object* .**HideSelection** [ = *boolean*]

The **HideSelection** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListView**, **RichTextBox**, or **TreeView** control. |
| *boolean* | A Boolean expression specifying how a control is displayed when it loses the focus, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) The items in the control are no longer selected when the control loses the focus. |
| **False** | The items are still selected after the control loses focus. |

### Remarks

Normally, the selected items in a control are hidden when the control loses focus.   This is the default action of the property.

If you want the selected items to remain selected after the control loses focus, set the **HideSelection** property to **False**.

## Text Property (Custom Controls)

Example

Returns or sets the text contained in an object.

**Syntax**

*object*.**Text** [= *string*]

The **Text** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ColumnHeader**, **ListItem**, **Panel**, or **Node** object, or a **RichTextBox** control. |
| *string* | A string expression specifying the text appearing in the object. |

■

## Text Property (Custom Controls) Example

This example populates a **TreeView** control with the titles of files in a **FileListBox** control. When an item in the **TreeView** control is clicked, the **Text** property is displayed in a message box. To try the example, place **TreeView** and **FileListBox** controls on a form and paste the code into the form's Declarations section. Run the example and click on any item to see its **Text** property.

```
Private Sub Form_Load()
   Dim nodX As Node   ' Declare an object variable for the Node.
   Dim i As Integer   ' Declare a variable for use as a counter.

   ' Add one Node to the TreeView control, and use the Path
   ' property of the FileListBox as its Text property.
   Set nodX = TreeView1.Nodes.Add()
   nodX.Text = File1.Path

   ' Add child nodes to the first Node object. Use the
   ' FileListBox to populate the control.
   For i = 0 To File1.ListCount - 1
      Set nodX = TreeView1.Nodes.Add(1, tvwChild)
      nodX.Text = File1.List(i)
      nodX.EnsureVisible     ' Make sure all nodes are visible.
   Next i
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
   ' Display the clicked Node object's Text property.
   MsgBox Node.Text
End Sub
```

## ImageList Control

See Also     Properties     Methods     Constants

An **ImageList** control contains a collection of **ListImage** objects, each of which can be referred to by its index or key.   The **ImageList** control is not meant to be used alone, but as a central   repository to conveniently supply other controls with images.

### Syntax

**ImageList**

### Remarks

You can use the **ImageList** control with any control that assigns a **Picture** object to a **Picture** property. For example, the following code assigns the first **ListImage** object in a **ListImages** collection to the **Picture** property of a newly created **StatusBar** panel:

```
Dim pnlX As Panel
Set pnlX = StatusBar1.Panels.Add() ' Add a new Panel object.
Set pnlX.Picture = ImageList1.ListImages(1).Picture ' Set Picture.
```

**Note**   You must use the **Set** statement when assigning an image to a **Picture** object.

The **ImageList** control can load either or both bitmaps and icons together as long as all images are of the same size.   You are not limited to any particular image size, but the total number of images that can be loaded is limited by the amount of available memory.

At design time, you can add images using the General tab of the ImageList Control Properties dialog box.   At run time, you can add images using the **Add** method for the **ListImages** collection.

Besides storing **Picture** objects, the **ImageList** control can also perform graphical operations on images before assigning them to other controls.   For example, the **Overlay** method creates a composite image from two disparate images.

Additionally, you can bind one or more **ImageList** controls to certain other Windows 95 common controls to conserve system resources.   These include the **ListView**, **ToolBar**, **TabStrip**, and **TreeView** controls.   In order to use an **ImageList** with one of these controls, you must associate a particular **ImageList** with the control through an appropriate property.   For the **ListView** control, you must set the **Icons** and **SmallIcons** properties to **ImageList** controls.   For the **TreeView**, **TabStrip**, and **Toolbar** controls, you must set the **ImageList** property to an **ImageList** control.

For these controls, you can specify an **ImageList** at design time using the Custom Properties dialog box.   At run time, you can also specify an **ImageList** which sets a **TreeView** control's **ImageList** property, as in the following example:

```
TreeView1.ImageList = ImageList1  ' Specify ImageList
```

Once you associate an **ImageList** with a control, you can use the value of either the **Index** or **Key** property to refer to a **ListImage** object in a procedure.   The following example sets the **Image** property of a **TreeView** control's third **Node** object to the first **ListImage** object in an **ImageList** control:

```
' Use the value of the Index property of ImageList1.
TreeView1.Nodes(3).Image = 1
' Or use the value of the Key property.
TreeView1.Nodes(3).Image = "image 1"   ' Assuming Key is "image 1."
```

**Distribution Note**   The **ImageList** control is a 32-bit custom control that can only run on 32-bit systems such as Windows 95 and Windows NT 3.51 or higher.   Additionally, the **ImageList** control is part of a group of custom controls that are found in the COMCTL32.OCX file.   To use the **ImageList** control in your application, you must add the COMCTL32.OCX file to the project.   When distributing your application, install the COMCTL32.OCX file in the user's Microsoft Windows SYSTEM directory.   For more information on how to add a custom control to a project, see the *Programmer's Guide*.

**See Also**

**Add** Method (**ListImages** Collection)
**Icons** Property
**ImageList** Property
**Index** Property
**Key** Property
**ListImage** Object, **ListImages** Collection
**ListView** Control
**SmallIcons** Property
**TabStrip** Control
**Toolbar** Control
**TreeView** Control

■
## ImageList Control Properties

**BackColor** Property
**ImageHeight** Property
**ImageWidth** Property
**Index** Property
**ListImages** Property
**MaskColor** Property
**Name** Property
**Object** Property
**Parent** Property
**Tag** Property

■
## ImageList Control Methods

**Overlay** Method

## ListImage Object, ListImages Collection

See Also Properties Methods

- A **ListImage** object is a bitmap of any size that can be used in other controls.
- A **ListImages** collection is a collection of **ListImage** objects.

### Syntax

*imagelist*.**ListImages**

*imagelist*.**ListImages(***index***)**

The syntax lines above refer to the collection and to individual elements in the collection, respectively, according to standard collection syntax.

The **ListImage** Object, **ListImages** Collection syntaxes have these parts:

| Part | Description |
|---|---|
| *imagelist* | An object expression that evaluates to an **ImageList** control. |
| *index* | An integer or string that uniquely identifies the object in the collection. The integer is the value of the **Index** property; the string is the value of the **Key** property. |

### Remarks

The **ListImages** collection is a 1-based collection.

You can add and remove a **ListImage** at design time using the General tab of the ImageList Control Properties page, or at run time using the **Add** method for **ListImage** objects.

Each item in the collection can be accessed by its index or unique key. For example, to get a reference to the third **ListImage** object in a collection, use the following syntax:

```
Dim imgX As ListImage
   ' Reference by index number.
Set imgX = ImageList.ListImages(3)
   ' Or reference by unique key.
Set imgX = ImageList1.ListImages("third") ' Assuming Key is "third."
   ' Or use Item method.
Set imgX = ImageList1.ListImages.Item(3)
```

Each **ListImage** object has a corresponding mask that is generated automatically using the **MaskColor** property. This mask is not used directly, but is applied to the original bitmap in graphical operations such as the **Overlay** and **Draw** methods.

**See Also**

**ImageList** Control
**ImageList** Property
**MaskColor** Property
**Overlay** Method

■

## ListImage Object, ListImages Collection Properties

Legend

**Count** Property■
**Index** Property■
**Key** Property■
**Picture** Property■

■

## ListImage Object, ListImages Collection Methods

Legend

**Add** Method■
**Clear** Method■
**Draw** Method■
**ExtractIcon** Method■
**Item** Method■
**Remove** Method■

## Add Method (ListImages Collection)

See Also Example

Adds a **ListImage** object to a **ListImages** collection. Doesn't support named arguments.

### Syntax

*object*.**Add(***index*, *key*, *picture***)**

The **Add** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListImages** collection. |
| *index* | Optional. An integer specifying the position where you want to insert the **ListImage**. If no *index* is specified, the **ListImage** is added to the end of the **ListImages** collection. |
| *key* | Optional. A unique string that identifies the **ListImage** object. Use this value to retrieve a specific **ListImage** object. NOTE: An error occurs if the key is not unique. |
| *picture* | Required. Specifies the picture to be added to the collection. |

### Remarks

The **ListImages** collection is a 1-based collection.

You can load either bitmaps or icons into a **ListImage** object as long as all images are of the same size. To load a bitmap or icon, you must use the **LoadPicture** function, as follows:

```
Set imgX = ImageList1.ListImages.Add(,,LoadPicture("file name"))
```

You can also load a **Picture** object directly into the **ListImage** object. For example, this example loads a **PictureBox** control's picture into the **ListImage** object:

```
Set imgX = ImageList1.ListImages.Add(,,Picture1.Picture)
```

If no **ListImage** objects have been added to a **ListImages** collection, you can set the **ImageHeight** and **ImageWidth** properties before adding the first **ListImage** object. The first **ListImage** object you add to a collection can be any size. However, all subsequent **ListImage** objects must be the same size as the first **ListImage** object. Once a **ListImage** object has been added to the collection, the **ImageHeight** and **ImageWidth** properties become read-only properties, and any image added to the collection must have the same **ImageHeight** and **ImageWidth** values.

You should use the **Key** property to reference a **ListImage** object if you expect the value of the **Index** property to change. For example, if you allow users to add and delete their own images to the collection, the value of the **Index** property may change.

When a **ListImage** object is added to the collection, a reference to the newly created object is returned. You can use the reference to set other properties of the **ListImage**, as follows:

```
Dim imgX As ListImage
Dim I As Integer
   Set imgX = ImageList1.ListImages. _
   Add(,,LoadPicture("icons\comm\net01.ico"))
   imgX.Key = "net connect" ' Use the new reference to assign Key.
```

**See Also**

**Clear** Method
**Count** Property
**ListImage** Object, **ListImages** Collection
**ImageHeight, ImageWidth** Properties
**Item** Method
**Remove** Method

■

## Add Method (ListImages Collection) Example

This example adds several images to a **ListImages** collection, and then uses the images in a **TreeView** control.   To try the example, place **ImageList** and **TreeView** controls on a form, and paste the code into the form's Declarations section.   Run the example to see the **TreeView** populated with pictures from the **ImageList**.

```
Private Sub Form_Load()
   Dim imgX As ListImage
   ' Load three icons into the ImageList control's collection.
   Set imgX = ImageList1.ListImages. _
   Add(,"rocket", LoadPicture("icons\industry\rocket.ico"))
   Set imgX = ImageList1.ListImages. _
   Add(,"plane",LoadPicture("icons\industry\plane.ico"))
   Set imgX = ImageList1.ListImages. _
   Add(,"car",LoadPicture("icons\industry\cars.ico"))

   ' Set TreeView control's ImageList property.
   Set TreeView1.ImageList = ImageList1

   ' Create a Treeview, and use ListImage objects for its images.
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,,"Rocket")
   nodX.Image = 1            ' Use the Index property of image 1.
   Set nodX = TreeView1.Nodes.Add(,,,"Plane")
   nodX.Image = "plane"      ' Use the Key property of image 2.
   Set nodX = TreeView1.Nodes.Add(,,,"Car")
   nodX.Image = "car"        ' Use the Key property of image 3.
End Sub
```

## Draw Method

See Also     Example

Draws an image into a destination device context (DC), such as a **PictureBox** control, after performing a graphical operation on the image. Doesn't support named arguments.

### Syntax

*object*.**Draw (***hDC*, *x*,*y*, *style***)**

The **Draw** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **ListImage** object. |
| *hDC* | Required. A value set to the target object's **hDC** property. |
| *x,y* | Optional. The coordinates used to specify the location within the device context where the image will be drawn. If you don't specify these, the image is drawn at the origin of the DC. |
| *style* | Optional. Specifies the operation performed on the image, as described in Settings. |

### Settings

The settings for *style* are:

| Constant | Value | Description |
|---|---|---|
| **imlNormal** | 0 | (Default) Normal. Draws the image with no change. |
| **imlTransparent** | 1 | Transparent. Draws the image using the **MaskColor** property to determine which color of the image will be transparent. |
| **imlSelected** | 2 | Selected. Draws the image dithered with the system highlight color. |
| **imlFocus** | 3 | Focus. Draws the image dithered and striped with the highlight color creating a hatched effect to indicate the image has the focus. |

### Remarks

The **hDC** property is a handle (a number) that the Windows operating system uses for internal reference to an object. You can paint in the internal area of any control that has an **hDC** property. In Visual Basic, these include the **Form** object, **PictureBox** control, and **Printer** object.

Because an object's **hDC** can change while an application is running, it is better to specify the **hDC** property rather than an actual value. For example, the following code ensures that the correct **hDC** value is always supplied to the **ImageList** control:

```
ImageList1.ListImages(1).Draw Form1.hDC
```

**See Also**

**ExtractIcon** Method
**ImageList** Control
**MaskColor** Property
**Overlay** Method

■

## Draw Method Example

This example loads an image into an **ImageList** control.   When you click the form, the image is drawn on the form in four different styles.   To try the example, place an **ImageList** control on a form and paste the code into the form's Declarations section.   Run the example and click the form.

```
Private Sub Form_Load()
   Dim X As ListImage
   'Load one image into the ImageList.
   Set X = ImageList1.ListImages. _
   Add(, , LoadPicture("bitmaps\assorted\intl_no.bmp"))
End Sub

Private Sub Form_Click()
   Dim space, intW As Integer      ' Create spacing variables.

   ' Use the ImageWidth property for spacing.
   intW = ImageList1.ImageWidth
   space = Form1.Font.Size * 2 ' Use the Font.Size for height spacing.

   ScaleMode = vbPoints            ' Set ScaleMode to points.
   Cls ' Clear the form.

   ' Draw the image with Normal style.
   ImageList1.ListImages(1).Draw Form1.hDC, , space,imlNormal
   ' Set MaskColor to red, which will become transparent.
   ImageList1.MaskColor = vbRed
   ' Draw the image with red (MaskColor) the transparent color.
   ImageList1.ListImages(1).Draw Form1.hDC, intW, space,imlTransparent
   ' Draw image with the Selected style.
   ImageList1.ListImages(1).Draw Form1.hDC, intW * 2,space,imlSelected
   ' Draw image with Focus style.
   ImageList1.ListImages(1).Draw Form1.hDC, intW * 3, space,imlFocus

   ' Print a caption for the images.
   Print _
   "Normal              Transparent           Selected          Focus"

End Sub
```

## ExtractIcon Method

See Also Example

Creates an icon from a bitmap in a **ListImage** object of an **ImageList** control and returns a reference to the newly created icon.

### Syntax

*object*.**ExtractIcon**

The *object* placeholder represents an object expression that evaluates to a **ListImage** object.

### Remarks

You can use the icon created with the **ExtractIcon** method like any other icon. For example, you can use it as a setting for the **MouseIcon** property, as the following code illustrates:

```
Set Command1.MouseIcon = ImageList1.ListImages(1).ExtractIcon
```

**See Also**

**Add** Method (**ListImages** Collection)
**ImageList** Control
**ListImage** Object, **ListImages** Collection

▪

## ExtractIcon Method Example

This example loads a bitmap into an **ImageList** control. When the user clicks the form, the **ExtractIcon** method is used to create an icon from the bitmap, and that icon is used as a setting in the **Form** object's **MouseIcon** property. To try the example, place an **ImageList** control on a form and paste the code into the form's Declarations section. Run the example and click the form.

```
Private Sub Form_Load()
   Dim imgX As ListImage
   Set imgX = ImageList1.ListImages. _
   Add(, , LoadPicture("bitmaps\assorted\balloon.bmp"))
End Sub

Private Sub Form_Click()
   Dim picX As Picture
   Set picX = ImageList1.ListImages(1).ExtractIcon  ' Make an icon.

   With Form1
   .MouseIcon =  picX             ' Set new icon.
   .MousePointer = vbCustom       ' Set to custom icon.
   End With
End Sub
```

## ImageHeight, ImageWidth Properties

See Also Example

- The **ImageHeight** property returns or sets the height of **ListImage** objects in an **ImageList** control.
- The **ImageWidth** property returns or sets the width of **ListImage** objects in an **ImageList** control.

### Syntax

*object*.**ImageHeight**

*object*.**ImageWidth**

The *object* placeholder represents an object expression that evaluates to an **ImageList** control.

### Remarks

Both height and width are measured in pixels. All images in a **ListImages** collection have the same height and width properties.

When an **ImageList** contains no **ListImage** objects, you can set both **ImageHeight** and **ImageWidth** properties. However, once a **ListImage** object has been added, all subsequent images must be of the same height and width as the first object. If you try to add an image of a different size, an error is returned.

**See Also**

**Add** Method (**ListImages** Collection)

**ImageList** Control

▪

## ImageHeight, ImageWidth Properties Example

This example loads an icon into an **ImageList** control, and uses the image in a **ListView** control. When the user clicks the form, the code uses the **ImageHeight** property to adjust the height of the **ListView** control to accommodate the **ListImage** object. To try the example, place **ImageList** and **ListView** controls on a form and paste the code into the form's Declarations section. Run the example and click the form.

```
Private Sub Form_Load()
   ' Create variables for ImageList and ListView objects.
   Dim imgX As ListImage
   Dim itmX As ListItem

   Form1.ScaleMode = vbPixels ' Make sure ScaleMode is set to pixels.

   ListView1.BorderStyle = FixedSingle ' Show border.
   ' Shorten ListView control so later contrast is more obvious.
   ListView1.Height = 50

   ' Put a large bitmap into the ImageList.
   Set imgX = ImageList1.ListImages. _
   Add(,, LoadPicture("bitmaps\gauge\vert.bmp"))

   ListView1.Icons = ImageList1 ' Set Icons property.

   ' Add an item to the ListView control.
   Set itmX = ListView1.ListItems.Add()
   itmX.Icon = 1                ' Set Icon property to ListImage 1 of
ImageList.
   itmX.Text = "Thermometer"    ' Set text of ListView ListItem object.
End Sub

Private Sub Form_Click()
   Dim strHW As String

   strHW = "Height: " & ImageList1.ImageHeight & _
   "   Width: " & ImageList1.ImageWidth
   caption = strHW                  ' Show dimensions.
   ' Enlarge ListView to accommodate the tallest image.
   ListView1.Height = ImageList1.ImageHeight + 50
End Sub
```

## ListImages Property

See Also Example

Returns a reference to a collection of **ListImage** objects in an **ImageList** control.

### Syntax

*object*.**ListImages**

The *object* placeholder represents an object expression that evaluates to an **ImageList** control.

### Remarks

You can manipulate **ListImage** objects using standard collection methods (for example, the **Add** and **Clear** methods). Each member of the collection can be accessed by its index or unique key. These are stored in the **Index** and **Key** properties, respectively, when **ListImage** is added to a collection.

**See Also**

**Add** Method (**ListImages** Collection)
**Clear** Method
**Count** Property
**Item** Method
**ListImage** Object, **ListImages** Collection
**Remove** Method

■

## ListImages Property Example

This example adds three **ListImage** objects to a **ListImages** collection and uses them in a **ListView** control.   The code refers to the **ListImage** objects using both their **Key** and **Item** properties.   To try the example, place **ImageList** and **ListView** controls on a form and paste the code into the form's Declarations section.   Run the example.

```
Private Sub Form_Load()
   Dim imgX As ListImage
   ' Add images to ListImages collection.
   Set imgX = ImageList1. _
   ListImages.Add(,"rocket",LoadPicture("icons\industry\rocket.ico"))
   Set imgX = ImageList1. _
   ListImages.Add(,"jet",LoadPicture("icons\industry\plane.ico"))
   Set imgX = ImageList1. _
   ListImages.Add(,"car",LoadPicture("icons\industry\cars.ico"))

   ListView1.Icons = ImageList1 ' Set Icons property.

   ' Add Item objects to the ListView control.
   Dim itmX as ListItem
   Set itmX = ListView1.ListItems.Add()
   ' Reference by index.
   itmX.Icon = 1
   itmX.Text = "Rocket"     ' Set Text string.
   Set itmX = ListView1.ListItems.Add()
   ' Reference by key ("jet").
   itmX.Icon = "jet"
   itmX.Text = "Jet"        ' Set Text string.
   Set itmX = ListView1.ListItems.Add()
   itmX.Icon = "car"
   itmX.Text = "Car"        ' Set Text string.
End Sub
```

## MaskColor Property

See Also Example

Returns or sets the color used to create masks for an **ImageList** control.

### Syntax

*object*.**MaskColor** [ = *color*]

The **MaskColor** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **ImageList** control. |
| *color* | A value or constant that determines the color used to create masks. You can specify colors using either Visual Basic intrinsic constants, the QBColor function, or the RGB function. |

### Remarks

Every image in a **ListImages** collection has a corresponding mask associated with it. The mask is a monochrome image derived from the image itself, automatically generated using the **MaskColor** property as the specific color of the mask. This mask is not used directly, but is applied to the original bitmap in graphical operations such as the **Overlay** and **Draw** methods. For example, the **MaskColor** property determines which color of an image will be transparent in the **Overlay** method.

**See Also**

**Draw** Method
**ImageList** Control
**Overlay** Method

▪

## MaskColor Property Example

This example loads several bitmaps into an **ImageList** control. As you click the form, one **ListImage** object is overlaid over one of the other **ListImage** objects. To try the example, place an **ImageList** control and a **Picture** control on a form and paste the code into the form's Declarations section. Run the program and click the form.

```
Private Sub Form_Load()
   Dim imgX As ListImage

   ' Load bitmaps.
   Set imgX = ImageList1.ListImages. _
   Add(, "No", LoadPicture("bitmaps\assorted\Intl_No.bmp"))
   Set imgX = ImageList1.ListImages. _
   Add(, , LoadPicture("bitmaps\assorted\smokes.bmp"))
   Set imgX = ImageList1.ListImages. _
   Add(, , LoadPicture("bitmaps\assorted\beany.bmp"))

   ScaleMode = vbPixels
   ' Set MaskColor property.
   ImageList1.MaskColor = vbGreen
   ' Set the form's BackColor to white.
   Form1.BackColor = vbWhite
End Sub

Private Sub Form_Click()
   Static intCount As Integer ' Static variable to count images.

   ' Reset variable to 2 if it is over the ListImages.Count value.
   If intCount > ImageList1.ListImages.Count Or intCount < 1 Then
      intCount = 2 ' Reset to second image
   End If

   ' Overlay ListImage(1) over ListImages 2-3.
   Picture1.Picture = ImageList1.Overlay(intCount, 1)
   ' Increment count.
   intCount = intCount + 1

   ' Create variable to hold ImageList.ImageWidth value.
   Dim intW
   intW = ImageList1.ImageWidth

   ' Draw images onto the form for reference. Use the ImageWidth
   ' value to space the images.
   ImageList1.ListImages(1).Draw Form1.hDC, 0, 0, imlNormal
   ImageList1.ListImages(2).Draw Form1.hDC, 0, intW, imlNormal
   ImageList1.ListImages(3).Draw Form1.hDC, 0, intW * 2, imlNormal
End Sub
```

## Overlay Method

See Also Example

Draws one image from a **ListImages** collection over another, and returns the result. Doesn't support named arguments.

### Syntax

*object*.**Overlay (***index1*, *index2***)**

The **Overlay** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **ImageList** control. |
| *index1* | An integer (**Index** property) or unique string (**Key** property) that specifies the image to be overlaid. |
| *index2* | An integer (**Index** property) or unique string (**Key** property) that specifies the image to be drawn over the object specified in *index1*. Note: The color of the image that matches the **MaskColor** property will be made transparent. If no color matches, the image will be drawn opaquely over the other image. |

### Remarks

Use the **Overlay** method in conjunction with the **MaskColor** property to create a single image from two disparate images. The **Overlay** method imposes one bitmap over another to create a third, composite image. The **MaskColor** property determines which color of the overlaying image is transparent.

The *index* can be either an index or a key. For example, to overlay the first picture in the collection with the second:

```
Set Picture1.Picture = ImageList1.Overlay(1,2) ' Reference by Index.
   'Or reference by Key property.
Set Picture1.Picture = ImageList1.Overlay("First", "Second")
```

**See Also**

**Draw** Method
**Index** Property
**ListImage** Object, **ListImages** Collection
**MaskColor** Property

▪

## Overlay Method Example

This example loads five **ListImage** objects into an **ImageList** control and displays any two images in two **PictureBox** controls. For each **PictureBox**, select an image to display from one of the two **ComboBox** controls. When you click the form, the code uses the **Overlay** method to create a third image that is displayed in a third **PictureBox** control. To try the example, place an **ImageList** control, two **ComboBox** controls, and three **PictureBox** controls on a form and paste the code into the form's Declarations section. Run the example and click the form.

```
Private Sub Form_Load()
   Dim X As ListImage
   ' Add 5 images to a ListImages collection.
   Set X = ImageList1.ListImages. _
    Add(, , LoadPicture("icons\elements\moon05.ico"))
   Set X = ImageList1.ListImages. _
    Add(, , LoadPicture("icons\elements\snow.ico"))
   Set X = ImageList1.ListImages. _
    Add(, , LoadPicture("icons\writing\erase02.ico"))
   Set X = ImageList1.ListImages. _
    Add(, , LoadPicture("icons\writing\note06.ico"))
   Set X = ImageList1.ListImages. _
    Add(, , LoadPicture("icons\flags\flgfran.ico"))

   With combo1               ' Populate the first ComboBox.
      .AddItem "Moon"
      .AddItem "Snowflake"
      .AddItem "Pencil"
      .AddItem "Note"
      .AddItem "Flag"
      .ListIndex = 0
   End With

   With combo2               ' Populate the second ComboBox.
      .AddItem "Moon"
      .AddItem "Snowflake"
      .AddItem "Pencil"
      .AddItem "Note"
      .AddItem "Flag"
      .ListIndex = 2
   End With

   Picture1.BackColor = vbWhite    ' Make BackColor white.
   Picture2.BackColor = vbWhite
   Picture3.BackColor = vbWhite
End Sub

Private Sub Form_Click()
   ' Overlay the two images, and display in PictureBox3.
   Set Picture3.Picture = ImageList1. _
    Overlay(combo1.ListIndex + 1, combo2.ListIndex + 1)
End Sub

Private Sub combo1_Click()
   ' Change PictureBox to reflect ComboBox selection.
   Set Picture1.Picture = ImageList1. _
    ListImages(combo1.ListIndex + 1).ExtractIcon
```

```
End Sub

Private Sub combo2_Click()
   ' Change PictureBox to reflect ComboBox selection.
   Set Picture2.Picture = ImageList1. _
    ListImages(combo2.ListIndex + 1).ExtractIcon
End Sub
```

## ListImages

The **ListImages** keyword is used in these contexts:

**ListImages** Collection

**ListImages** Property

## ImageList Control Constants

See Also

| Constant | Value | Description |
|---|---|---|
| **imlNormal** | 0 | Image is drawn with no change. |
| **imlTransparent** | 1 | Image is drawn transparently. |
| **imlSelected** | 2 | Image is drawn selected. |
| **imlFocus** | 3 | Image is drawn with focus. |

**See Also**

**Draw** Method

**ImageList** Control

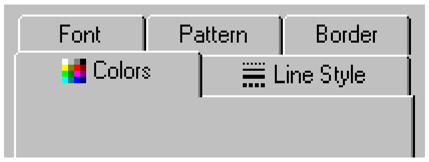
Visual Basic Custom Control Constants

Windows 95 Controls Constants

# TabStrip Control

See Also Properties Methods Events Constants

A **TabStrip** is like the dividers in a notebook or the labels on a group of file folders. By using a **TabStrip** control, you can define multiple pages for the same area of a window or dialog box in your application.



## Syntax

**TabStrip**

## Remarks

The control consists of one or more **Tab** objects in a **Tabs** collection. At both design time and run time, you can affect the **Tab** object's appearance by setting properties, and at run time, by invoking methods to add and remove **Tab** objects.

The **Style** property determines whether the **TabStrip** control looks like push buttons (Buttons) or notebook tabs (Tabs). At design time when you put a **TabStrip** control on a form, it has one notebook tab. If you choose the **Tabs** setting for the **Style** property, a three-dimensional border defines the **TabStrip** control's internal area. Even though the **Style** property's **Buttons** setting shows no border around the internal area, that area still exists.

To set the overall size of the **TabStrip** control, use its drag handles and/or set the **Top**, **Left**, **Height**, and **Width** properties. Based on the control's overall size at run time, Visual Basic automatically determines the size and position of the internal area and returns the Client-coordinate properties▪**ClientLeft**, **ClientTop**, **ClientHeight**, and **ClientWidth**. The **MultiRow** property determines whether the control can have more than one row of tabs, the **TabWidthStyle** property determines the appearance of each row, and, if **TabWidthStyle** is set to **Fixed**, you can use the **TabFixedHeight** and **TabFixedWidth** properties to set the same height and width for all tabs in the **TabStrip** control.

The **TabStrip** control is not a container. To contain the actual pages and their objects, you must use **PictureBox** controls or other containers that match the size of the internal area which is shared by all **Tab** objects in the control. If you use a control array for the container, you can associate each item in the array with a specific **Tab** object, as in the following example:

```
' This code puts the selected tab's picture container on top.
Picture1(TabStrip1.SelectedItem.Index - 1).ZOrder 0
```

The **Tabs** property of the **TabStrip** control is the collection of all the **Tab** objects. Each **Tab** object has properties associated with its current state and appearance. For example, you can associate an **ImageList** control with the **TabStrip** control, and then use images on individual tabs. You can also associate a ToolTip with each **Tab** object.

**Distribution Note** The **TabStrip** control is a 32-bit custom control that can only run on Windows 95 and Windows NT 3.51 or higher. Additionally, the **TabStrip** control is part of a group of custom controls that are found in the COMCTL32.OCX file. To use the **TabStrip** control in your application, you must add the COMCTL32.OCX file to the project. When distributing your application, install the COMCTL32.OCX file in the user's Microsoft Windows SYSTEM directory. For more information on how to add a custom control to a project, see the *Programmer's Guide*.

**See Also**

**ImageList** Control
**SelectedItem** Property
**Selected** Property
**Tab** Object, **Tabs** Collection
**ZOrder** Method

■

**TabStrip Control Properties**

**ClientHeight** Property
**ClientLeft** Property
**ClientTop** Property
**ClientWidth** Property
**Container** Property
**DragIcon** Property
**DragMode** Property
**Enabled** Property
**Font** Property
**Height** Property
**HelpContextID** Property
**hWnd** Property
**ImageList** Property
**Index** Property
**Left** Property
**MouseIcon** Property
**MousePointer** Property
**MultiRow** Property
**Name** Property
**Object** Property
**Parent** Property
**SelectedItem** Property
**ShowTips** Property
**Style** Property
**TabFixedHeight** Property
**TabFixedWidth** Property
**TabIndex** Property
**Tabs** Property
**TabStop** Property
**TabWidthStyle** Property
**Tag** Property
**Top** Property
**Visible** Property
**WhatsThisHelpID** Property
**Width** Property

■
## TabStrip Control Methods

**Drag** Method
**Move** Method
**Refresh** Method
**SetFocus** Method
**ShowWhatsThis** Method
**ZOrder** Method

■

## TabStrip Control Events

BeforeClick Event
Click Event
DblClick Event
DragDrop Event
DragOver Event
GotFocus Event
KeyDown Event
KeyPress Event
KeyUp Event
LostFocus Event
MouseDown Event
MouseMove Event
MouseUp Event

## Tab Object, Tabs Collection

See Also Properties Methods

A **Tabs** collection contains all the **Tab** objects in a **TabStrip** control. A **Tab** object is analogous to a divider in a notebook.

**Important** This object requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*tabstrip*.**Tabs**

*tabstrip*.**Tabs(***index***)**

The syntax lines above refer to the collection and to individual elements in the collection, respectively, according to standard collection syntax.

The **Tab** object, **Tabs** collection syntax has these parts:

| Part | Description |
|---|---|
| *tabstrip* | An object expression that evaluates to a **TabStrip** control. |
| *index* | An integer or string that uniquely identifies a member of an object collection. The integer is the value of the **Index** property; the string is the value of the **Key** property. |

### Remarks

For each **Tab** object, you can use various properties to specify its appearance, and you can specify its state with the **Selected** property.

At design time, use the Insert Tab and Remove Tab buttons on the Tabs tab in the TabStrip Control Properties dialog box to insert and remove tabs, and use the text boxes to specify any of these properties for a **Tab** object: **Caption**, **Image**, **ToolTipText**, **Tag**, **Index**, and/or **Key**. You can also specify these properties at run time.

Use the **Caption** and **Image** properties, separately or together, to label or put an icon on a tab.

- To use the **Caption** property, in the Caption text box on the Tabs tab in the TabStrip Control Properties dialog box, type the text you want to appear on the tab or button at run time.
- To use the **Image** property, put an **ImageList** control on the form and fill the **ListImages** collection with **ListImage** objects, which each get an index number. On the General tab in the TabStrip Control Properties dialog box, select that **ImageList** to associate it with the **TabStrip** control. In the Image text box on the Tabs tab, type the index number of the **ListImage** object that should appear on the **Tab** object.

Use the **ToolTipText** property to temporarily display a string of text in a small rectangular box at run time when the user's cursor hovers over the tab. To set the **ToolTipText** property at design time, select the **ShowTips** checkbox on the General tab, and then in the ToolTipText text box on the Tabs tab, type the ToolTip string.

At run time, use the **Index** and/or **Key** properties to retrieve a **Tab** object from the **Tabs** collection.

To return a reference to a **Tab** object a user has selected, use the **SelectedItem** property; to determine whether a specific tab is selected, use the **Selected** property. These properties are useful in conjunction with the BeforeClick event to verify or record data associated with the currently-selected tab before displaying the next tab the user selects.

Each **Tab** object also has read-only properties you can use to reference a single **Tab** object in the **Tabs** collection: **Left**, **Top**, **Height** and **Width**.

The **Tabs** collection uses the **Count** property to return the number of tabs in the collection. To manipulate the **Tab** objects in the **Tabs** collection, use these methods at run time:

- **Add**

■adds **Tab** objects to the **TabStrip** control.
- **Item**

■retrieves the **Tab** identified by its **Key** or **Index** from the collection.
- **Clear**

■removes all **Tab** objects from the collection.

- **Remove**

- removes the **Tab** identified by its **Key** or **Index** from the collection.

**See Also**

**SelectedItem** Property

**TabStrip** Control

■

## Tab Object, Tabs Collection Properties

Legend

**Caption** Property■
**Count** Property■
**Height** Property■
**Image** Property■
**Index** Property■
**Left** Property■
**Key** Property■
**Selected** Property■
**Tag** Property■
**ToolTipText** Property■
**Top** Property■
**Width** Property■

■

## Tab Object, Tabs Collection Methods

Legend

**Add** Method■
**Clear** Method■
**Item** Method■
**Remove** Method■

## TabStrip Control Constants

See Also

### TabStyle Constants

| Constant | Value | Description |
|---|---|---|
| **tabTabs** | 0 | Tabs appear as notebook tabs, and the internal area has a three-dimensional border enclosing it. |
| **tabButtons** | 1 | Tabs appear as push buttons, and the internal area has no border around it. |

### TabWidthStyle Constants

| Constant | Value | Description |
|---|---|---|
| **tabJustified** | 0 | Each tab is wide enough to accommodate its contents, and the width of each tab is increased, if needed, so that each row of tabs spans the width of the control.   If there is only a single row of tabs, this style has no effect. |
| **tabNonJustified** | 1 | Each tab is just wide enough to accommodate its contents. The rows are not justified, so multiple rows of tabs are jagged. |
| **tabFixed** | 2 | The height and width of all tabs are identical, and are set by the **TabFixedHeight** and **TabFixedWidth** properties. |

**See Also**

**MultiRow** Property
**Style** Property (**TabStrip** Control)
**TabFixedHeight**, **TabFixedWidth** Properties
**Tab** Object, **Tabs** Collection
**TabStrip** Control
**TabWidthStyle** Property

## Add Method (Tabs Collection)

See Also Example

Adds a **Tab** object to a **Tabs** collection in a **TabStrip** control. Doesn't support named arguments.

---

**Important** This method requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**Add(***index, key, caption, image***)**

The **Add** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Tabs** collection. |
| *index* | Optional. An integer specifying the position where you want to insert the **Tab**. If you don't specify an index, the **Tab** is added to the end of the **Tabs** collection. |
| *key* | Optional. A unique string that can be used to retrieve the **Tab** with the **Item** method or remove the **Tab** with the **Remove** method. |
| *caption* | Optional. The string that appears on the **Tab**. |
| *image* | Optional. The index of an image in an associated **ImageList** control. This image is displayed on the tab. |

### Remarks

To add tabs to the **TabStrip** control at design time, click the Insert Tab button on the Tab tab in the TabStrip Control Properties dialog box, and then fill in the appropriate fields for the new tab.

To add tabs to the **TabStrip** control at run time, use the **Add** method, which returns a reference to the newly inserted **Tab** object. For example, the following code adds a tab with the *caption*, "Howdy!" whose *key* is "MyTab," as the second tab (its *index* is 2):

```
Set X = TabStrip1.Tabs.Add(2,"MyTab","Howdy!")
```

**See Also**

**Caption** Property (**Tabs** Object)
**Clear** Method
**Image** Property
**Index** Property
**Item** Method
**Key** Property
**Remove** Method
**Tab** Object, **Tabs** Collection

■

## Add Method (Tabs Collection) Example

This example adds three **Tab** objects, each with captions and images from an **ImageList** control, to a **TabStrip** control.   To try this example, put an **ImageList** and a **TabStrip** control on a form.   The **ImageList** control supplies the images for the **Tab** objects.   Paste the following code into the Load event of the Form object, and run the program.

```
Private Sub Form_Load()
   Dim X As Integer
   Set TabStrip1.ImageList = ImageList1
   TabStrip1.Tabs(1).Caption = "Time"
   TabStrip1.Tabs.Add 2, , "Date"
   TabStrip1.Tabs.Add 3, , "Mail"
   For X = 1 To TabStrip1.Tabs.Count
      TabStrip1.Tabs(X).Image = X
   Next X
End Sub
```

## BeforeClick Event

See Also Example

Generated when a **Tab** object in a **TabStrip** control is clicked, or a **Tab** object's **Selected** setting has changed.

**Important** This event requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

**Private Sub** *object*_**BeforeClick(***cancel* **As Integer)**

The BeforeClick event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TabStrip** control. |
| *cancel* | Evaluates to an integer with values of 0 (False) and -1 (True). The initial value is 0. |

### Remarks

Use the BeforeClick event to validate the information on the old **Tab** object before actually generating a **Click** event that selects the new **Tab** object. The *cancel* argument allows you to stop a change to the new selection.

**Note** If you use the **MsgBox** or **InputBox** functions during the BeforeClick event procedure, the **TabStrip** control will not receive a Click event, regardless of the setting of the *cancel* argument.

**See Also**

**Click** Event
**SelectedItem** Property
**TabStrip** Control

▪

**BeforeClick Event Example**

This example uses the BeforeClick event to demonstrate how to prevent a Click event from occurring. This is useful when you want to verify information on the current tab before displaying the newly selected tab.

To try this example, place a **TabStrip** control and a two-element **PictureBox** control array on the form. In the first **PictureBox** control, add a **CheckBox** control and in the second, add a **TextBox**. Paste the following code into the Load event of the Form object, and run the program. Click the tab labeled Text after you select/deselect the CheckBox on the tab labeled Check.

```
Private Sub Form_Load()
Dim i As Integer
Dim Tabx As Object
' Sets the caption of the first tab to "Check."
TabStrip1.Tabs(1).Caption = "Check"
' Adds a second tab with "Text" as its caption.
Set Tabx = TabStrip1.Tabs.Add(2, , "Text")
' Labels the checkbox.
Check1.Caption = "Cancel tab switch"
   ' Aligns the picture boxes with the internal area
   ' of the Tabstrip Control.
   For i = 0 To 1
      Picture1(i).Left = TabStrip1.ClientLeft
      Picture1(i).Top = TabStrip1.ClientTop
      Picture1(i).Height = TabStrip1.ClientHeight
      Picture1(i).Width = TabStrip1.ClientWidth
   Next
   ' Puts the first tab's picture box container on top.
   Picture1(0).ZOrder 0
End Sub

' The BeforeClick event verifies the check box value
' to determine whether to proceed with the Click event.
Private Sub TabStrip1_BeforeClick(Cancel As Integer)
   If TabStrip1.Tabs(1).Selected Then
      If Check1.Value = 1 Then Cancel = True
   End If
End Sub

Private Sub TabStrip1_Click()
   Picture1(TabStrip1.SelectedItem.Index-1).ZOrder 0
End Sub
```

## Caption Property (Tab Object)

See Also        Example

Returns or sets the caption that appears on the tab or button of a **Tab** object in a **TabStrip** control.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Caption** [= *string*]

The **Caption** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Tab** object. |
| *string* | A string expression that evaluates to the text displayed as the caption. |

### Remarks

You can set the **Caption** property for a **Tab** object in the **TabStrip** control at design time or at run time.

- Design time
- On the Tab tab in the TabStrip Control properties dialog box, type the caption string in the Caption text box.
- Run time
- Set the caption as follows:

```
TabStrip1.Tabs(1).Caption = "First Tab"
```

Or

```
TabStrip1.Tabs.Add 2, , "Second Tab"
```

**See Also**

**Add** Method (**Tabs** Collection)
**Image** Property
**Tab** Object, **Tabs** Collection
**TabStrip** Control

■

## Caption Property (Tab Object) Example

This example sets the **Caption** property for each of three **Tab** objects it adds to a **TabStrip** control. The caption strings are "Time," "Date," and "Mail."   Each **Tab** object also displays an image from an **ImageList** control.   To try this example, place an **ImageList** and a **TabStrip** control on a form.   Place three sample bitmaps in the **ImageList** control.   The **ImageList** control supplies the images for the **Tab** objects.   Paste the following code into the Load event of the Form object, and run the program.

```
Private Sub Form_Load()
   Dim X As Integer
   ' Associate an ImageList with the TabStrip control.
   Set TabStrip1.ImageList = ImageList1
   ' Set the captions.
   Set TabStrip1.Tabs(1).Caption = "Time"
   TabStrip1.Tabs.Add 2, , "Date"
   TabStrip1.Tabs.Add 3, , "Mail"
   For X = 1 To TabStrip1.Tabs.Count
      ' Associate an image with a tab.
      TabStrip1.Tabs(X).Image = X
   Next X
End Sub
```

## ClientHeight, ClientWidth, ClientLeft, ClientTop Properties

See Also Example

Return the coordinates of the internal area (display area) of the **TabStrip** control. Read-only at run time; not available at design time.

---

**Important** These properties require either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

---

### Syntax

*object*.**ClientHeight**

*object*.**ClientWidth**

*object*.**ClientLeft**

*object*.**ClientTop**

The *object* placeholder represents an object expression that evaluates to a **TabStrip** control.

### Remarks

At run time, the client-coordinate properties■**ClientLeft**, **ClientTop**, **ClientHeight**, and **ClientWidth**

■automatically store the coordinates of the **TabStrip** control's internal area, which is shared by all **Tab** objects in the control. So that the controls associated with a specific **Tab** appear when that **Tab** object is selected, place the **Tab** object's controls inside a container, such as a **PictureBox** control, whose size and position match the client-coordinate properties. To associate a container (and its controls) with a **Tab** object, create a control array, such as a **PictureBox** control array.
All client-coordinate properties use the scale mode of the parent form. To place a **PictureBox** control so it fits perfectly in the internal area, use the following code:

```
Picture1.Left = TabStrip1.ClientLeft
Picture1.Top = TabStrip1.ClientTop
Picture1.Width = TabStrip1.ClientWidth
Picture1.Height = TabStrip1.ClientHeight
```

To create the effect of placing a new tab and its associated container on top when the tab is selected:

- Set the size and location of the container in the **TabStrip** control's internal area to the client-coordinate properties; and
- Program the **Visible** property to manually show and hide the tab-specific container and its controls; or
- Use the **ZOrder** method to place the controls you specify at the front or back of the z-order.

**See Also**

**Tab** Object, **Tabs** Collection
**TabStrip** Control
**Visible** Property
**ZOrder** Method

■

## ClientHeight, ClientWidth, ClientLeft, ClientTop Properties Example

The following example demonstrates using the Client-coordinate properties■**ClientLeft**, **ClientTop**, **ClientWidth**, and **ClientHeight**

■along with a **PictureBox** control array to display tab-specific objects in the internal area of the **TabStrip** control when switching tabs. The example uses the **ZOrder** method to display the appropriate **PictureBox** control and the objects it contains.

To try this example, place a **TabStrip** control and a three-element **PictureBox** control array on the form. In one **PictureBox** control, place a **CheckBox** control, in another, place a **CommandButton** control, and in the third, place a **TextBox** control. Paste the following code into the Load event of the Form object, and run the program. Click the various tabs to select them and their contents.

```
Private Sub Form_Load()
Dim Tabx As Object
Dim i As Integer
   ' Sets the caption of the first tab to "Check."
   TabStrip1.Tabs(1).Caption = "Check"
   ' Adds a second tab with "Command" as its caption.
   Set Tabx = TabStrip1.Tabs.Add(2, , "Command")
   ' Adds a third tab with "Text" as its caption.
   Set Tabx = TabStrip1.Tabs.Add(3, , "Text")

   ' Aligns the picture boxes with the internal area
   ' of the TabStrip control.
   For i = 0 To 2
      Picture1(i).Left = TabStrip1.ClientLeft
      Picture1(i).Top = TabStrip1.ClientTop
      Picture1(i).Height = TabStrip1.ClientHeight
      Picture1(i).Width = TabStrip1.ClientWidth
   Next
   ' Puts the first tab's picture box container on top
   ' at startup.
   Picture1(0).ZOrder 0
End Sub

Private Sub TabStrip1_Click()
   Picture1(TabStrip1.SelectedItem.Index - 1).ZOrder 0
End Sub
```

## MultiRow Property

See Also

Returns or sets a value indicating whether a **TabStrip** control can display more than one row of tabs.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object.***MultiRow** [= *boolean*]

The **MultiRow** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TabStrip** control. |
| *boolean* | A boolean expression that specifies whether the control has more than one row of tabs, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | Allows more than one row of tabs. |
| **False** | Restricts tabs to a single row. |

### Remarks

The number of rows is automatically set by the width and number of the tabs. The number of rows can change if the control is resized, which ensures that the tab wraps to the next row. If **MultiRow** is set to **False**, and the last tab exceeds the width of the control, a horizontal spin control is added at the right end of the **TabStrip** control.

At design time, set the **MultiRow** property on the General tab in the TabStrip Properties dialog box. At run time, use code like the following to set the **MultiRow** property:

```
'Allows more than one row of tabs in the TabStrip control.
TabStrip1.MultiRow = TRUE
```

**See Also**

**ClientHeight**, **ClientWidth**, **ClientLeft**, **ClientTop** Properties
**TabFixedHeight**, **TabFixedWidth** Properties
**TabStrip** Control
**TabStrip** Control Constants
**TabWidthStyle** Property

## Style Property (TabStrip Control)

See Also

Returns or sets the appearance▪tabs or buttons
▪of a **TabStrip** control.

**Important**  This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Style** [= *value*]

The **Style** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TabStrip** control. |
| *value* | A constant or integer that determines the appearance of the tabbed dialog box, as described in Settings. |

### Settings

The settings for *value* are:

| Constant | Value | Description |
|---|---|---|
| **tabTabs** | 0 | (Default) Tabs. The tabs appear as notebook tabs, and the internal area has a three-dimensional border around it. |
| **tabButtons** | 1 | Buttons. The tabs appear as regular push buttons, and the internal area has no border around it. |

### Remarks

At design time, select the **Style** property you want▪tabs or buttons
▪from the Style list on the General tab of the TabStrip Control Properties dialog box.
At run time, use code like the following to set the **Style** property:

```
' Style property set to the Tabs style.
TabStrip1.Style = tabTabs

' Style property set to the Buttons style:
TabStrip1.Style = tabButtons
```

**See Also**

**ClientHeight, ClientWidth, ClientLeft, ClientTop** Properties
**MultiRow** Property
**TabFixedHeight**, **TabFixedWidth** Properties
**TabStrip** Control
**TabStrip** Control Constants
**TabWidthStyle** Property

## Tabs Property (TabStrip Control)

See Also

Returns a reference to the collection of **Tab** objects in a **TabStrip** control.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**Tabs(***index***)**

The **Tabs** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TabStrip** control. |
| *index* | A value that identifies a **Tab** object in the **Tabs** collection. |

### Remarks

The **Tabs** collection can be accessed by using the standard collection methods, such as the **Item** method.

**See Also**

**Add** Method (**Tabs** Collection)
**Clear** Method (Collection Objects)
**Index** Property
**Item** Method
**Key** Property
**Remove** Method
**Tab** Object, **Tabs** Collection
**TabStrip** Control

## TabFixedHeight, TabFixedWidth Properties

See Also

Return or set the fixed height and width of all **Tab** objects in a **TabStrip** control, but only if the **TabWidthStyle** property is set to **tabFixed**.

**Important** These properties require either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**TabFixedHeight** [= *integer*]

*object*.**TabFixedWidth** [= *integer*]

The **TabFixedHeight** and **TabFixedWidth** properties syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TabStrip** control. |
| *integer* | The number of pixels or twips of the height or width of a **TabStrip** control. |

### Remarks

The **TabFixedHeight** property applies to all **Tab** objects in the **TabStrip** control. It defaults either to the height of the font as specified in the **Font** property, or the height of the **ListImage** object specified by the **Image** property, whichever is higher, plus a few extra pixels as a border. If the **TabWidthStyle** property is set to **tabFixed**, and the value of the **TabFixedWidth** property is set, the width of each **Tab** object remains the same whether you add or delete **Tab** objects in the control.

**See Also**

**ClientHeight**, **ClientWidth**, **ClientLeft**, **ClientTop** Properties
**Font** Property
**Image** Property
**ImageList** Property
**TabStrip** Control
**TabStrip** Control Constants
**TabWidthStyle** Property

## TabWidthStyle Property

See Also

Returns or sets a value that determines the justification or width of all **Tab** objects in a **TabStrip** control.

**Important** This property requires either Microsoft Windows 95 or Microsoft Windows NT version 3.51 or higher.

### Syntax

*object*.**TabWidthStyle** [=*value*]

The **TabWidthStyle** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TabStrip** control. |
| *value* | An integer or constant that determines whether tabs are justified or set to a fixed width, as described in Settings. |

### Settings

The settings for *value* are:

| Constant | Value | Description |
|---|---|---|
| **tabJustified** | 0 | (Default) Justified. If the **MultiRow** property is set to **True**, each tab is wide enough to accommodate its contents and, if needed, the width of each tab is increased so that each row of tabs spans the width of the control. If the **MultiRow** property is set to **False**, or if there is only a single row of tabs, this setting has no effect. |
| **tabNonJustified** | 1 | Nonjustified. Each tab is just wide enough to accommodate its contents. The rows are not justified, so multiple rows of tabs are jagged. |
| **tabFixed** | 2 | Fixed. All tabs have an identical width which is determined by the **TabFixedWidth** property. |

### Remarks

At design time you can set the **TabWidthStyle** property on the General tab of the **TabStrip** Control Properties dialog box. The setting of the **TabWidthStyle** property affects how wide each **Tab** object appears at run time.

At run time, you can set the **TabWidthStyle** property as follows:

```
' Justifies all the tabs in a row to fit the width of the control.
TabStrip1.MultiRow = True
TabStrip1.TabWidthStyle = tabJustified

' Creates ragged rows of tabs.
TabStrip1.MultiRow = True
TabStrip1.TabWidthStyle = tabNonJustified

' Sets the same width for all tabs.
TabStrip1.TabFixedWidth = 500
TabStrip1.TabWidthStyle = tabFixed
```

**See Also**

**ClientHeight**, **ClientWidth**, **ClientLeft**, **ClientTop** Properties
**MultiRow** Property
**TabFixedHeight**, **TabFixedWidth** Properties
**TabStrip** Control

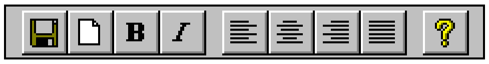
## Tabs

The **Tabs** keyword is used in these contexts:
**Tabs** Collection
**Tabs** Property (**TabStrip** Control)

## Toolbar Control

See Also Properties Methods Events Constants Example

A **Toolbar** control contains a collection of **Button** objects used to create a toolbar that is associated with an application.



### Syntax

**Toolbar**

### Remarks

Typically, a toolbar contains buttons that correspond to items in an application's menu, providing a graphic interface for the user to access an application's most frequently used functions and commands.

The **Toolbar** control allows you to create toolbars by adding **Button** objects to a **Buttons** collection; each **Button** object can have optional text and/or an image, supplied by an associated **ImageList** control. Set text with the **Caption** property, and an image with the **Image** property for each **Button** object. At design time, you can add **Button** objects to the control with the Toolbar Control Properties dialog box. At run time, you can add or remove buttons from the **Buttons** collection using **Add** and **Remove** methods.

To program the **Toolbar**, the ButtonClick event allows individual **Button** objects to respond to the user's actions. You can also determine the behavior and appearance of each **Button** object using the **Style** property. For example, if four buttons are assigned the ButtonGroup style, only one button can be pressed at any time and at least one button is always pressed.

You can place another control on a toolbar by assigning a **Button** object the PlaceHolder style. For example, to place a drop-down text box on a toolbar at design time, add a **Button** object with the PlaceHolder style and size it to the size of a **ComboBox** control. Then place a **ComboBox** on the placeholder.

Double clicking a toolbar at run time invokes the Customize Toolbar dialog box, which allows the user to hide, display, or rearrange toolbar buttons. To enable or disable the dialog box, use the **AllowCustomize** property. You can also invoke the Customize Toolbar dialog box using the **Customize** method. If you wish to save and restore the state of a toolbar, or allow the user to do so, two methods are provided: the **SaveToolbar** and **RestoreToolbar** methods. The Change event, generated when a toolbar is altered, is typically used to invoke the **SaveToolbar** method.

Usability is further enhanced by programming **ToolTipText** descriptions of each **Button** object. To display ToolTips, the **ShowTips Property** must be set to **True**. When the user invokes the Customize Toolbar dialog box, clicking a button causes a description of the button to be displayed in the dialog box; this description can be programmed by setting the **Description** property.

---

**Distribution Note** The **Toolbar** control is a 32-bit custom control that can only run on 32-bit systems such as Windows 95 and Windows NT version 3.51 or higher. Additionally, the **Toolbar** control is part of a group of custom controls that are found in the COMCTL32.OCX file. To use the **Toolbar** control in your application, you must add the COMCTL32.OCX file to the project. When distributing your application, install the COMCTL32.OCX file in the user's Microsoft Windows SYSTEM directory. For more information on how to add a custom control to a project, see the *Programmer's Guide*.

---

**See Also**

**Add** Method (**Buttons** Collection)
**Button** Object, **Buttons** Collection
**ImageList** Control

▪

**Toolbar Control Properties**

**Align** Property
**AllowCustomize** Property
**ButtonHeight** Property
**Buttons** Property
**ButtonWidth** Property
**Container** Property
**DragIcon** Property
**DragMode** Property
**Enabled** Property
**Font** Property
**Height** Property
**hWnd Property**
**ImageList** Property
**Index** Property
**Left** Property
**MouseIcon** Property
**MousePointer** Property
**Name** Property
**Negotiate** Property
**Object** Property
**Parent** Property
**ShowTips Property**
**TabIndex** Property
**Tag** Property
**Top** Property
**Visible** Property
**WhatsThisHelpID** Property
**Width** Property
**Wrappable** Property

■

**Toolbar Control Methods**

**Customize** Method
**Drag** Method
**Move** Method
**Refresh** Method
**RestoreToolbar** Method
**SaveToolbar** Method
**ShowWhatsThis** Method
**ZOrder** Method

■
## Toolbar Control Events

▪
## Toolbar Control Example

This example adds several **Button** objects to a **Toolbar** control using the **Add** method and assigns images supplied by the **ImageList** control.   The behavior of each button is determined by the **Style** property.   The code creates buttons to open and save files for a **RichTextBox** control, set text alignment, and set font color.   Two **CommandButton** controls allow you to edit and restore the state of the toolbar. To try the example, place a **Toolbar**, **RichTextBox**, **ImageList**, **CommonDialog**, **ComboBox**, and two **CommandButton** controls on a form and paste the code into the form's Declarations section.   Run the example, click the various buttons, and type into the **RichTextBox**.

```
' SaveToolbar method constants.
Const SaveToolbarKey = 1
Const SaveToolbarSubKey = "MyToolbar"
Const SaveToolbarVal = "True"

Private Sub Form_Load()
   ' Create object variable for the ImageList.
   Dim imgX As ListImage

   ' Load pictures into the ImageList control.
   Set imgX = ImageList1.ListImages. _
   Add(, "open", LoadPicture("bitmaps\tlbr_w95\open.bmp"))    ' 1
   Set imgX = ImageList1.ListImages. _
   Add(, "save", LoadPicture("bitmaps\tlbr_w95\save.bmp"))    ' 2
   Set imgX = ImageList1.ListImages. _
   Add(, "left", LoadPicture("bitmaps\tlbr_w95\lft.bmp"))   ' 3
   Set imgX = ImageList1.ListImages. _
   Add(, "right", LoadPicture("bitmaps\tlbr_w95\rt.bmp"))   ' 4
   Set imgX = ImageList1.ListImages. _
   Add(, "center", LoadPicture("bitmaps\tlbr_w95\cnt.bmp"))   ' 5
   Set imgX = ImageList1.ListImages. _
   Add(, "justify", LoadPicture("bitmaps\tlbr_w95\jst.bmp")) ' 6
   Set imgX = ImageList1.ListImages. _
   Add(, "bold", LoadPicture("bitmaps\tlbr_w95\bld.bmp"))   ' 7
   Set imgX = ImageList1.ListImages. _
   Add(, "italic", LoadPicture("bitmaps\tlbr_w95\Itl.bmp"))   ' 8
   Toolbar1.ImageList = ImageList1

   ' Create object variable for the Toolbar.
   Dim btnX As Button
   ' Add button objects to Buttons collection using the
   ' Add method. After creating each button, set both
   ' Description and ToolTipText properties.
   Set btnX = Toolbar1.Buttons.Add(, , , tbrSeparator)
   Set btnX = Toolbar1.Buttons.Add(, "open", , tbrDefault, "open")
   btnX.ToolTipText = "Open File"
   btnX.Description = btnX.ToolTipText
   Set btnX = Toolbar1.Buttons.Add(, "save", , tbrDefault, "save")
   btnX.ToolTipText = "Save File"
   btnX.Description = btnX.ToolTipText
   Set btnX = Toolbar1.Buttons.Add(, , , tbrSeparator)
   Set btnX = Toolbar1.Buttons.Add(, "left", , tbrButtonGroup,"left")
   btnX.ToolTipText = "Align Left"
   btnX.Description = btnX.ToolTipText
   Set btnX = Toolbar1.Buttons.Add(,"center", ,tbrButtonGroup,"center")
   btnX.ToolTipText = "Center"
   btnX.Description = btnX.ToolTipText
   Set btnX = Toolbar1.Buttons.Add(, "right", ,tbrButtonGroup,"right")
   btnX.ToolTipText = "Align Right"
   btnX.Description = btnX.ToolTipText
   Set btnX = Toolbar1.Buttons.Add(, , , tbrSeparator)
   Set btnX = Toolbar1.Buttons.Add(, "bold", , tbrCheck, "bold")
   btnX.ToolTipText = "Bold"
   btnX.Description = btnX.ToolTipText
   Set btnX = Toolbar1.Buttons.Add(, "italic", , tbrCheck, "italic")
   btnX.ToolTipText = "Italic"
   btnX.Description = btnX.ToolTipText
   Set btnX = Toolbar1.Buttons.Add(, , , tbrSeparator)

   ' The next button has the Placeholder style. A ComboBox control
   ' will be placed on top of this button.
   Set btnX = Toolbar1.Buttons.Add(, "combo1", , tbrPlaceholder)
   btnX.Width = 2000 ' Placeholder width to accommodate a combobox.

   Show ' Show form to continue configuring ComboBox.

   ' Configure ComboBox control to be at same location as the
   ' Button object with the PlaceHolder style (key = "combo1").
   With Combo1
      .Width = Toolbar1.Buttons("combo1").Width
      .Top = Toolbar1.Buttons("combo1").Top
      .Left = Toolbar1.Buttons("combo1").Left
      .AddItem "Black" ' Add colors for text.
      .AddItem "Blue"
      .AddItem "Red"
      .ListIndex = 0
   End With

   With Toolbar1
      .Wrappable = True ' Buttons can wrap.
      ' Prevent customization except by clicking Command1.
      .AllowCustomize = False
   End With

   ' Configure commondialog1 for opening and saving files.
   With CommonDialog1
      .DefaultExt = ".rtf"
      .Filter = "RTF file (*.RTF)|*.RTF"
   End With

   'Configure CommandButton 1 to be positioned just below the toolbar.
   With Command1
      .Left = Toolbar1.Buttons(2).Left
      .Top = Toolbar1.Top + Toolbar1.Height + 100
      .Width = 1500
      .Height = 300
      .Caption = "Customize Toolbar"
   End With

   'Configure CommandButton 2 to be positioned to right of Command1.
   With Command2
      .Left = Command1.Left + Command1.Width + 50
```

```
      .Top = Command1.Top
      .Width = 1500
      .Height = 300
      .Caption = "Restore Toolbar"
   End With

   ' Set margin of the RichTextBox to the width of the control.
   richtextbox1.RightMargin = richtextbox1.Width

End Sub

Private Sub Form_Resize()
   ' Configure ComboBox control.
   With Combo1
      .Width = Toolbar1.Buttons("combo1").Width
      .Top = Toolbar1.Buttons("combo1").Top
      .Left = Toolbar1.Buttons("combo1").Left
   End With

End Sub

Private Sub richtextbox1_SelChange()
   ' When the insertion point changes, set the Toolbar buttons
   ' to reflect the attributes of the text where the cursor is located.
   ' Use the Select Case statement.
   ' The SelAlignment property returns either 0, 1, 2, or Null.
   Select Case richtextbox1.SelAlignment
   Case Is = rtfLeft ' 0
      Toolbar1.Buttons("left").VALUE = tbrPressed
   Case Is = rtfRight '1
      Toolbar1.Buttons("right").VALUE = tbrPressed
   Case Is = rtfCenter '2
      Toolbar1.Buttons("center").VALUE = tbrPressed
   Case Else ' Null -- No buttons are shown in the up position.
      Toolbar1.Buttons("left").VALUE = tbrUnpressed
      Toolbar1.Buttons("right").VALUE = tbrUnpressed
      Toolbar1.Buttons("center").VALUE = tbrUnpressed
   End Select

   ' SelBold returns 0, -1, or Null.  If it's Null then set
   ' the MixedState property to True.
   Select Case richtextbox1.SelBold
   Case 0 ' Not bold.
      Toolbar1.Buttons("bold").VALUE = tbrUnpressed
   Case -1 ' Bold.
      Toolbar1.Buttons("bold").VALUE = tbrPressed
   Case Else ' Mixed state.
      Toolbar1.Buttons("bold").MixedState = True
   End Select

   ' SelItalic returns 0, -1, or Null.  If it's Null then set
   ' the MixedState property to True.
   Select Case richtextbox1.SelItalic
   Case 0 ' Not italic.
      Toolbar1.Buttons("italic").VALUE = tbrUnpressed
   Case -1 ' Italic.
```

```
      Toolbar1.Buttons("italic").VALUE = tbrPressed
   Case Else ' Mixed State.
      Toolbar1.Buttons("italic").MixedState = True
   End Select
End Sub

Private Sub toolbar1_ButtonClick(ByVal Button As Button)
   ' Use the Key property with the SelectCase statement to specify
   ' an action.
   Select Case Button.KEY
   Case Is = "open"           ' Open file.
      Dim strOpen As String   ' String variable for file name.
      CommonDialog1.ShowOpen  ' Show Open File dialog box.
      strOpen = CommonDialog1.filename ' Set variable to filename.
      richtextbox1.LoadFile strOpen, 0 ' Use LoadFile method.
   Case Is = "save"              ' Save file.
      Dim strNewFile As String   ' String variable for new file name.
      CommonDialog1.ShowSave     ' Show Save dialog box.
      strNewFile = CommonDialog1.filename ' Set variable to file name.
      richtextbox1.SaveFile strNewFile, 0 ' Use SaveFile method.
   Case Is = "left"
      richtextbox1.SelAlignment = rtfLeft
   Case Is = "center"
      richtextbox1.SelAlignment = rtfCenter
   Case Is = "right"
      richtextbox1.SelAlignment = rtfRight
   Case Is = "bold"
      ' Test to see if the MixedState property is True. If so,
      ' then set it to false before doing anything else.
      If Button.MixedState = True Then
            Button.MixedState = False
      End If
      ' Toggle the SelBold property.
      richtextbox1.SelBold = Abs(richtextbox1.SelBold) - 1
   Case Is = "italic"
      ' Test to see if the MixedState property is True. If so,
      ' then set it to false before doing anything else.
      If Button.MixedState = True Then
         Button.MixedState = False
      End If
      ' Toggle the SelItalic property.
      richtextbox1.SelItalic = Abs(richtextbox1.SelItalic) - 1
   End Select
End Sub

Private Sub Combo1_Click()
   ' Change font colors of text using the ComboBox.
   With richtextbox1
      Select Case Combo1.ListIndex
      Case 0
      .SelColor = vbBlack
      Case 1
      .SelColor = vbBlue
      Case 2
      .SelColor = vbRed
      End Select
   End With
   ' Return focus to the RichTextbox control.
   richtextbox1.SetFocus
End Sub

Private Sub command1_Click()
   ' Save the state of Toolbar1 before allowing further customization.
   With Toolbar1
      .SaveToolbar SaveToolbarKey, SaveToolbarSubKey, SaveToolbarVal
      .AllowCustomize = True ' AllowCustomize must be True.
      .Customize     ' Customize method invokes Customize Dialog box.
      .AllowCustomize = False ' After customization, set this to False.
   End With
End Sub

Private Sub Command2_Click()
   ' Restore state of Toolbar1 using Constants.
   Toolbar1.RestoreToolbar SaveToolbarKey, _
   SaveToolbarSubKey, SaveToolbarVal
End Sub
```

## Button Object, Buttons Collection

See Also Properties Methods

- A **Button** object contains an image and a caption, both of which are optional.
- A **Buttons** collection is a collection of **Button** objects for a **Toolbar** control.

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*toolbar.***Buttons**

*toolbar.***Buttons**(*index*)

The syntax lines above refer to the collection and to individual elements in the collection, respectively, according to standard collection syntax.

The **Button** object, **Buttons** collection syntax has these parts:

| Part | Description |
|---|---|
| *toolbar* | An object expression that evaluates to a **Toolbar** control. |
| *index* | An integer or string that uniquely identifies the object in the collection. The integer is the value of the **Index** property; the string is the value of the **Key** property. |

### Remarks

The **Buttons** collection is a 1-based collection, which means the collection's **Index** property begins with the number 1 (versus 0 in a 0-based collection).

Each item in the collection can be accessed by its index or unique key. For example, to get a reference to the third **Button** object in a collection, use the following syntax:

```
Dim btnX As Button
   ' Reference by index number.
Set btnX = Toolbar1.Buttons(3)
   ' Or reference by unique key.
Set btnX = Toolbar1.Buttons("third") ' Assuming Key is "third."
   ' Or use Item method.
Set btnX = Toolbar1.Buttons.Item(3)
```

**See Also**

**ImageList** Control
**Index** Property
**Key** Property
**Toolbar** Control

■

## Button Object , Buttons Collection Properties

Legend

**Caption** Property■

**Count** Property■
**Description** Property (**Button** Object)■
**Enabled** Property■
**Height** Property■
**Image** Property■
**Index** Property■
**Key** Property■
**Left** Property■
**MixedState** Property■
**Style** Property■
**Tag** Property■
**ToolTipText Property**■
**Top** Property■
**Value** Property■
**Visible** Property■
**Width** Property■

■

## Button Object, Buttons Collection Methods

Legend

**Add** Method (**Buttons** Collection)■

**Clear** Method■

**Item** Method■

**Remove** Method■

## Add Method (Buttons Collection)

See Also

Adds a **Button** object to a **Buttons** collection and returns a reference to the newly created object. Doesn't support named arguments.

**Important** This method requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object*.**Add(***index, key, caption, style, image***)**

The **Add** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **Buttons** collection. |
| *index* | Optional. An integer specifying the position where you want to insert the **Button** object. If no *index* is specified, the **Button** is added to the end of the **Buttons** collection. |
| *key* | Optional. A unique string that identifies the **Button** object. Use this value to retrieve a specific **Button** object. |
| *caption* | Optional. A string that will appear beneath the **Button** object. |
| *style* | Optional. The style of the **Button** object. The available styles are detailed in the **Style** Property (**Button** Object). |
| *image* | Optional. An integer or unique key that specifies a **ListImage** object in an associated **ImageList** control. |

### Remarks

You can add **Button** objects at design time using the Buttons tab of the Toolbar Control Properties dialog box. At run time, use the **Add** method to supplement the buttons already present on the toolbar.

You associate an **ImageList** control with the **Toolbar** through the **Toolbar** control's **ImageList** property.

**See Also**

**Caption** Property
**ImageList** Control
**ImageList** Property
**Index** Property
**Key** Property
**Style** Property (**Button** Object)
**Toolbar** Control

## AllowCustomize Property

See Also

Returns or sets a value determining if a **Toolbar** control can be customized by the end user with the Customize Toolbar dialog box.

**Important** This property requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object*.**AllowCustomize** [= *boolean*]

The **AllowCustomize** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Toolbar** control. |
| *boolean* | A constant or value that determines if the user can customize a **Toolbar** control, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | Allows the end user to invoke the Customize Toolbar dialog box by double clicking a **Toolbar** control. |
| **False** | Customization of the **Toolbar** control with the Customize Toolbar dialog box is not allowed. |

### Remarks

If the **AllowCustomize** property is set to **True**, double-clicking a **Toolbar** control at run time invokes the Customize Toolbar dialog box.

The Customize Toolbar can also be invoked with the **Customize** method. However, the **AllowCustomize** property must be set to **True** before the method can be invoked.

**See Also**

**Customize** Method
**RestoreToolbar** Method
**SaveToolbar** Method
**Toolbar** Control

## ButtonClick Event

See Also

Occurs when the user clicks on a **Button** object in a **Toolbar** control.

**Important** This event requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

**Private Sub** *object*_**ButtonClick**(**ByVal** *button* **As Button**)

The ButtonClick event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Toolbar** control. |
| *button* | A reference to the clicked **Button** object. |

### Remarks

To program an individual **Button** object's response to the ButtonClick event, use the value of the *button* argument. For example, the following code uses the **Key** property of the **Button** object to determine the appropriate action.

```
Private Sub Toolbar1_ButtonClick(ByVal Button As Button)
   Select Case Button.Key
   Case "Open"
      CommandDialog1.ShowOpen
   Case "Save"
      CommandDialog.ShowSave
   End Select
End Sub
```

**Note** Because the user can rearrange **Button** objects using the Customize Toolbar dialog box, the value of the **Index** property may not always indicate the position of the button. Therefore, it's preferable to use the value of the **Key** property to retrieve a **Button** object.

**See Also**

**Toolbar** Control

**Button** Object, **Buttons** Collection

**Value** Property

## ButtonHeight, ButtonWidth Properties

See Also

Return or set the height and width of a **Toolbar** control's buttons.

**Important** These properties require either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object*.**ButtonHeight** [= *number*]

*object*.**ButtonWidth** [= *number*]

The **ButtonHeight, ButtonWidth** properties syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Toolbar** control. |
| *number* | A numeric expression specifying the dimensions of all buttons on the control that have the Button, Check, or ButtonGroup style. |

### Remarks

**ButtonHeight** and **ButtonWidth** use the scale units of the **Toolbar** control's container, which is determined by the **ScaleMode** property of the container.

The **ButtonWidth** is automatically updated to accommodate the longest string (**Caption** property of the **Button** object) on a **Toolbar** control.

**See Also**

**Toolbar** Control

**Height**, **Width** Properties

## Buttons Property

See Also

Returns a reference to a **Toolbar** control's collection of **Button** objects.

---

**Important** This property requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

---

### Syntax

*object*.**Buttons**

The *object* placeholder is an object expression that evaluates to a **Toolbar** control.

### Remarks

You can manipulate **Button** objects using standard collection methods (for example, the **Add** and **Remove** methods). Each element in the collection can be accessed by its index, the value of the **Index** property, or a unique key, the value of the **Key** property.

**See Also**

**Add** Method (**Buttons** Collection)
**Clear** Method
**Count** Property
**Item** Method
**Remove** Method

## Change Event (Toolbar Control)

See Also

Generated after the end user customizes a **Toolbar** control's appearance using the Customize Toolbar dialog box.

---

**Important** This event requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

---

### Syntax

**Private Sub** *object*_**Change()**

The *object* placeholder is an object expression that evaluates to a **Toolbar** control.

### Remarks

The Change event is typically used in conjunction with the **SaveToolbar** method to save changes to a toolbar, as follows:

```
Private Sub Toolbar1_Change()
   ' Save the changes to the Windows registry whenever
   ' the Toolbar changes.
   Toolbar1.SaveToolbar(1, "Custom1", "MyToolbar")
End Sub
```

**See Also**

**AllowCustomize** Property

**Customize** Method

**RestoreToolbar** Method

**SaveToolbar** Method

## Customize Method

See Also

Invokes the Customize Toolbar dialog box which allows the end user to rearrange or hide **Button** objects on a **Toolbar** control.

---

**Important** This object requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

---

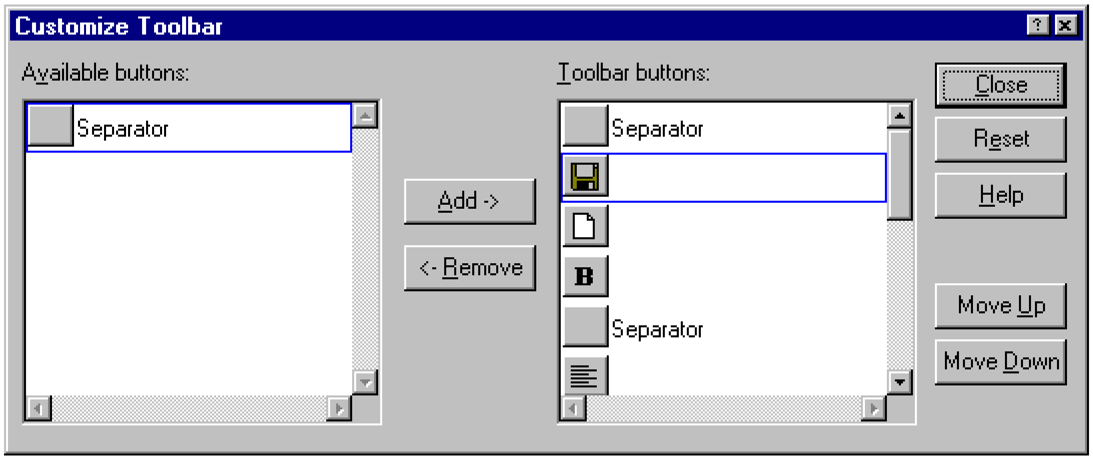
### Syntax

*object*.**Customize**

The *object* placeholder is an object expression that evaluates to a **Toolbar** control.

### Remarks

The **Toolbar** control contains a built-in dialog box that allows the user to hide, display, or rearrange buttons on a toolbar. When the **Toolbar** control's **AllowCustomize** property is set to **True**, double-clicking the toolbar calls the **Customize** method, which invokes the dialog box illustrated below:



Use the **Customize** method when you wish to restrict the alteration of the toolbar. For example, the code below allows the user to customize the toolbar only if a password is given:

```
Private Sub Command1_Click()
   If InputBox("Password:") = "Chorus&Line9" Then
      Toolbar1.AllowCustomize = True   ' Allow customization.
      Toolbar1.Customize               ' Invoke Customize method.
   End If
End Sub
```

To preserve the state of a **Toolbar** control, the **SaveToolbar** method writes to the Windows registry. You can restore a **Toolbar** control to a previous state using the **RestoreToolbar** method to read the information previously saved in the registry.

**See Also**

**AllowCustomize** Property
**Description** Property
**RestoreToolbar** Method
**SaveToolbar** Method
**Toolbar** Control

## Description Property (Button Object)

See Also

Returns or sets the text for a **Button** object's description, which is displayed in the Customize Toolbar dialog box.

**Important** This property requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object*.**Description** [= *string*]

The **Description** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Button** object. |
| *string* | The string displayed in the Customize Toolbar dialog box when the button is selected. |

### Remarks

At run time, the Customize Toolbar dialog box can be invoked either by a user double-clicking the **Toolbar** control or programmatically using the **Customize** method. In either case, when the user selects a button in the dialog box, a description of the button is displayed in the lower-left corner of the dialog box. The text for that description is set with the **Description** property.

You can set the **Description** text when you add a **Button** object, as follows:

```
Dim btnX As Button
' Set Image property to a button with the Key "save."
Set btnX = Toolbar1.Buttons.Add(,"save")
btnX.Description = "Save a file."
```

**See Also**

**AllowCustomize** Property
**Button** Object, **Buttons** Collection
**Caption** Property
**Customize** Method

## MixedState Property

See Also

Returns or sets a value that determines if a **Button** object in a **Toolbar** control appears in an indeterminate state.

**Important** This property requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object*.**MixedState** [= *boolean*]

The **MixedState** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Button** object. |
| *boolean* | A Boolean expression that determines if a **Button** shows the indeterminate state, as specified in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | The **Button** object is in the indeterminate state and becomes dimmed. |
| **False** | The **Button** object is not in the indeterminate state and looks normal. |

### Remarks

The **MixedState** property is typically used when a selection contains a variety of attributes. For example, if you select text that contains both plain (normal) characters and bold characters, the **MixedState** property is used. The image displayed by the **Button** object could then be changed to indicate its state, which would differ from the Checked and Unchecked value returned by the **Value** property.

**See Also**

**Button** Object, **Buttons** Collection
**Toolbar** Control
**Value** Property

## RestoreToolbar Method

See Also

Restores a toolbar, created with a **Toolbar** control, to its original state after being customized. Doesn't support named arguments.

---

**Important** This method requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

---

### Syntax

*object*.**RestoreToolbar(***key* **As Integer***, subkey* **As String**, *value* **As String)**

The **RestoreToolbar** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **Toolbar** control. |
| *key* | Required. An integer that specifies the key in the Windows registry where the method retrieves the **Toolbar** information. |
| *subkey* | Required. A string that specifies a location under the key specified in *key*. |
| *value* | Required. The **Toolbar** information stored in the subkey. |

### Remarks

Toolbars created with a **Toolbar** control can be customized at run time using the **Customize** method. The state of the toolbar can be saved in the registry using the **SaveToolbar** method; the **RestoreToolbar** method restores the state of a toolbar by reading the registry.

Both the *key* and *subkey* arguments must exist in the registry of the user's computer, or an error will occur.

**See Also**

**AllowCustomize** Property
Change Event (**Toolbar** Control)
**Customize** Method
**SaveToolbar** Method
**Toolbar** Control

## SaveToolbar Method

See Also

At run time, saves the state of a toolbar, created with the **Toolbar** control, in the registry.   Doesn't support named arguments.

---

**Important**   This method requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

---

### Syntax

*object*.**SaveToolbar(***key* **As Integer,** *subkey* **As String**, *value* **As String)**

The **SaveToolbar** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required.   An object expression that evaluates to a **Toolbar** control. |
| *key* | Required.   An integer specifying the key in the registry where the method stores the **Toolbar** information. |
| *subkey* | Required.   A string expression that specifies a location under the *key* specified in the previous parameter. |
| *value* | Required.   The **Toolbar** information to be stored in the subkey. |

### Remarks

If the *key* or *subkey* you specify doesn't exist in the registry, a new *key* or *subkey* is created.

You must set the **AllowCustomize** property to **True** to enable users to customize the toolbar.

Unless you create a new subkey, the **SaveToolbar** method allows the user to save only one version of the toolbar by overwriting the registry information each time the method is invoked.

**See Also**

**AllowCustomize** Property
Change Event (**Toolbar** Control)
**Customize** Method
**RestoreToolbar** Method
**Toolbar** Control

## Style Property (Button Object)

See Also

Returns or sets a constant or value that determines the appearance and behavior of a **Button** object in a **Toolbar** control.

**Important** This property requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object.***Style** [=*value*]

The **Style** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Button** object. |
| *value* | A constant or integer that determines the appearance and behavior of a **Button** object, as specified in Settings. |

### Settings

The settings for *value* are:

| Constant | Value | Description |
|---|---|---|
| **tbrDefault** | 0 | (Default) Button. The button is a regular push button. |
| **tbrCheck** | 1 | Check. The button is a check button, which can be checked or unchecked. |
| **tbrButtonGroup** | 2 | ButtonGroup. The button remains pressed until another button in the group is pressed. Exactly one button in the group can be pressed at any one moment. |
| **tbrSeparator** | 3 | Separator. The button functions as a separator with a fixed width of 8 pixels. |
| **tbrPlaceholder** | 4 | Placeholder. The button is like a separator in appearance and functionality, but has a settable width. |

### Remarks

Buttons that have the ButtonGroup style must be grouped. To distinguish a group, place all **Button** objects with the same style (ButtonGroup) between two **Button** objects with the Separator style.

You can also place another control on a toolbar by assigning a **Button** object the PlaceHolder style. For example, to place a drop-down text box on a toolbar at design time, add a **Button** object with the PlaceHolder style and size it to the size of a **ComboBox** control. Then place a **ComboBox** on the placeholder.

When a **Button** object is assigned the PlaceHolder style, you can set the value of the **Width** property to accommodate another control placed on the **Button**. If a **Button** object has the Button, Check, or ButtonGroup style, the height and width are determined by the **ButtonHeight** and **ButtonWidth** properties.

If you place a control on a button with the PlaceHolder style, you must programmatically align and size the control if the form is resized, as shown below:

```
Private Sub Form_Resize()
   ' Track a ComboBox by setting its Top, Left, and Width properties
   ' to the Top, Left, and Width properties of a Button object.
   Combo1.Top = Toolbar1.Buttons("combo1").Top
   Combo1.Left = Toolbar1.Buttons("combo1").Left
   Combo1.Width = Toolbar1.Buttons("combo1").Width
End Sub
```

**See Also**

**ButtonHeight**, **ButtonWidth** Properties
**Button** Object, **Buttons** Collection
**Toolbar** Control

## Wrappable Property

See Also

Returns or sets a value that determines if **Toolbar** control buttons will automatically wrap when the window is resized.

**Important** This property requires either the Microsoft Windows 95 operating system or Windows NT 3.51 and higher.

### Syntax

*object*.**Wrappable** [= *boolean*]

The **Wrappable** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Toolbar** control. |
| *boolean* | A Boolean expression that determines if the **Button** objects on a **Toolbar** control will wrap, as described in Settings. |

### Settings

The settings for *boolean* are:

| Value | Description |
|---|---|
| **True** | The buttons on the **Toolbar** control wrap if the form is resized. |
| **False** | The buttons on the **Toolbar** control won't wrap if the form is resized. |

**See Also**

**Button** Object, **Buttons** Collection
**Toolbar** Control

## Buttons

The **Buttons** keyword is used in these contexts:
**Buttons** Collection
**Buttons** Property

## Toolbar Control Constants

See Also

### Style Constants

| Constant | Value | Description |
|---|---|---|
| **tbrDefault** | 0 | The button is a regular push button. |
| **tbrCheck** | 1 | The button is a check button. |
| **tbrButtonGroup** | 2 | The button remains pressed until another button in the group is pressed. Exactly one button in the group is pressed at any time. |
| **tbrSeparator** | 3 | The button functions as a separator with a fixed width of 8 pixels. |
| **tbrPlaceholder** | 4 | The button is like a separator in appearance and functionality but has a settable width. |

### Value Constants

| Constant | Value | Description |
|---|---|---|
| **tbrUnpressed** | 0 | The button is not currently pressed or checked. |
| **tbrPressed** | 1 | The button is currently pressed or checked. |

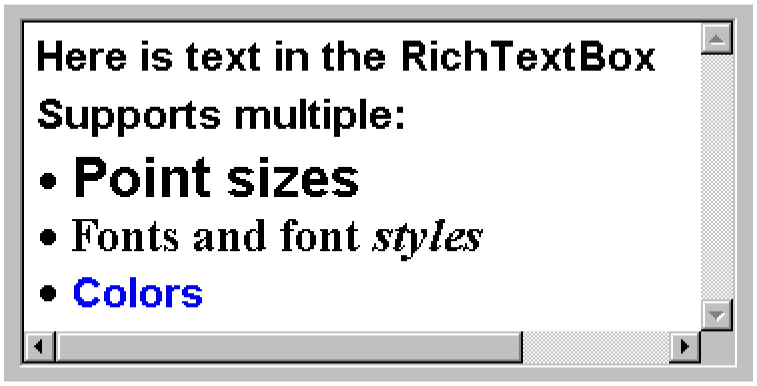
**See Also**

**Style** Property (**Button** Object)
**Toolbar** Control
**Value** Property
Visual Basic Custom Control Constants
Windows 95 Controls Constants

## RichTextBox Control

See Also Properties Methods Events Constants

The **RichTextBox** control allows the user to enter and edit text while also providing more advanced formatting features than the conventional **TextBox** control.



### Syntax

**RichTextBox**

### Remarks

The **RichTextBox** control provides a number of properties you can use to apply formatting to any portion of text within the control. To change the formatting of text, it must first be selected. Only selected text can be assigned character and paragraph formatting. Using these properties, you can make text bold or italic, change the color, and create superscripts and subscripts. You can also adjust paragraph formatting by setting both left and right indents, as well as hanging indents.

The **RichTextBox** also opens and saves files in both the RTF format and regular ASCII text format. You can use methods of the control (**LoadFile** and **SaveFile**) to directly read and write files, or use properties of the control such as **SelRTF** and **TextRTF** in conjunction with Visual Basic's file input/output statements. You can also load the contents of an .RTF file into the **RichTextBox** control simply by dragging the file (from the Windows 95 Explorer for example), or a highlighted portion of a file used in another application (such as Microsoft Word), and dropping the contents directly onto the control. You can also set the **FileName** property to load the contents of an .RTF or text file to the control.

You can also print all or part of the text in a **RichTextBox** control using the **SelPrint** method.

Because the **RichTextBox** is a data-bound control, you can bind it with a **Data** control to a Memo field in a Microsoft Access database or a similar large capacity text field in other databases (such as a TEXT data type field in SQL Server).

The **RichTextBox** control supports almost all of the properties, events and methods used with the standard **TextBox** control, such as **MaxLength**, **MultiLine**, **ScrollBars**, **SelLength**, **SelStart**, and **SelText**. Applications that already use **TextBox** controls can easily be adapted to make use of **RichTextBox** controls. However, the **RichTextBox** control doesn't have the same 64K character capacity limit of the conventional **TextBox** control.

---

**Distribution Note** The **RichTextBox** control is a 32-bit custom control that can only run on 32-bit systems, such as Windows 95 and Windows NT 3.51 or higher. To use the **RichTextBox** control in your application, you must add the RICHTX32.OCX file to the project. When distributing your application, install the RICHTX32.OCX file in the user's Microsoft Windows SYSTEM directory. For more information on how to add a custom control to a project, see the *Programmer's Guide*.

---

**See Also**

Supported RTF Codes

▪

## RichTextBox Control Properties

**Appearance** Property
**BackColor** Property
**BorderStyle** Property
**BulletIndent** Property
**Container** Property
**DataChanged** Property
**DataField** Property
**DataSource** Property
**DisableNoScroll** Property
**DragIcon** Property
**DragMode** Property
**Enabled** Property
**FileName** Property
**Font** Property
**Height** Property
**HelpContextID** Property
**HideSelection** Property
**hWnd** Property
**Index** Property
**Left** Property
**Locked** Property
**MaxLength** Property
**MouseIcon** Property
**MousePointer** Property
**MultiLine** Property
**Name** Property
**Object** Property
**Parent** Property
**ScrollBars** Property
**SelAlignment** Property
**SelBold** Property
**SelBullet** Property
**SelCharOffset** Property
**SelColor** Property
**SelFontName** Property
**SelFontSize** Property
**SelHangingIndent** Property
**SelIndent** Property
**SelItalic** Property
**SelLength** Property
**SelRightIndent** Property
**SelRTF** Property
**SelStart** Property
**SelStrikethru** Property
**SelTabCount** Property

**SelTabs** Property
**SelText** Property
**SelUnderline** Property
**TabIndex** Property
**TabStop** Property
**Tag** Property
**Text** Property
**TextRTF** Property
**Top** Property
**Visible** Property
**WhatsThisHelpID** Property
**Width** Property

■

## RichTextBox Control Methods

**Drag** Method
**Find** Method
**GetLineFromChar** Method
**LoadFile** Method
**Move** Method
**Refresh** Method
**SaveFile** Method
**SelPrint** Method
**SetFocus** Method
**ShowWhatsThis** Method
**Span** Method
**UpTo** Method
**ZOrder** Method

■

## RichTextBox Control Events

Change Event
Click Event
DblClick Event
DragDrop Event
DragOver Event
GotFocus Event
KeyDown Event
KeyPress Event
KeyUp Event
LostFocus Event
MouseDown Event
MouseMove Event
MouseUp Event
SelChange Event

## DisableNoScroll Property

See Also

Returns or sets a value that determines whether scroll bars in the **RichTextBox** control are disabled.

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**DisableNoScroll** [= *boolean*]

The **DisableNoScroll** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *boolean* | A Boolean expression specifying whether or not the scroll bars are enabled, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **False** | (Default) Scroll bars appear normally when displayed. |
| **True** | Scroll bars appear dimmed when displayed. |

### Remarks

The **DisableNoScroll** property is ignored when the **ScrollBars** property is set to 0 (None). However, when **ScrollBars** is set to 1 (Horizontal), 2 (Vertical), or 3 (Both), individual scroll bars are disabled when there are too few lines of text to scroll vertically or too few characters of text to scroll horizontally in the **RichTextBox** control.

**See Also**

**RichTextBox** Control

**ScrollBars** Property

## FileName Property (RichTextBox Control)

See Also

Returns or sets the filename of the file loaded into the **RichTextBox** control at design time.

---

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

---

### Syntax

*object*.**FileName**

The *object* placeholder represents an object expression that evaluates to a **RichTextBox** control.

### Settings

You can only specify the names of text files or valid .RTF files for this property.

**See Also**

**LoadFile** Method
**RichTextBox** Control
**SaveFile** Method
Supported RTF Codes

## Find Method

See Also	Example

Searches the text in a **RichTextBox** control for a given string. Doesn't support named arguments.

**Important** This method requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**Find**(*string, start, end, options*)

The **Find** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **RichTextBox** control. |
| *string* | Required. A string expression you want to find in the control. |
| *start* | Optional. An integer character index that determines where to begin the search. Each character in the control has an integer index that uniquely identifies it. The first character of text in the control has an index of 0. |
| *end* | Optional. An integer character index that determines where to end the search. |
| *options* | Optional. The sum of one or more constants used to specify optional features, as described in Settings. |

### Settings

The setting for *options* can include:

| Constant | Value | Description |
|---|---|---|
| **rtfWholeWord** | 2 | Determines if a match is based on a whole word or a fragment of a word. |
| **rtfMatchCase** | 4 | Determines if a match is based on the case of the specified string as well as the text of the string. |
| **rtfNoHighlight** | 8 | Determines if a match appears highlighted in the **RichTextBox** control. |

You can combine multiple options by either adding their values or constants together or combine the values with the **Or** operator.

### Remarks

If the text searched for is found, the **Find** method highlights the specified text and returns the index of the first character highlighted. If the specified text is not found, the **Find** method returns -1.

If you use the **Find** method without the **rtfNoHighlight** option while the **HideSelection** property is **True** and the **RichTextBox** control does not have the focus, the control still highlights the found text. Subsequent uses of the **Find** method will search only for the highlighted text until the insertion point moves.

The search behavior of the **Find** method varies based on the combination of values specified for the *start* and *end* arguments. This table describes the possible behaviors:

| Start | End | Search Behavior |
|---|---|---|
| Specified | Specified | Searches from the specified start location to the specified end location. |
| Specified | Omitted | Searches from the specified start location to the end of the text in the control. |
| Omitted | Specified | Searches from the current insertion point to the specified end location. |
| Omitted | Omitted | Searches the current selection if text is selected or the entire contents of the control if no text is selected. |

**See Also**

**RichTextBox** Control

■

## Find Method Example

This example finds a string in a **RichTextBox** control based on a word entered in a **TextBox** control. After it finds the specified string, it displays a message box that shows the number of the line containing the specified word. To try this example, put a **RichTextBox** control, a **CommandButton** control and a **TextBox** control on a form. Load a file into the **RichTextBox,** and paste this code into the Click event of the **CommandButton** control. Then run the example, enter a word in the **TextBox**, and click the **CommandButton**.

```
Private Sub Command1_Click()
   Dim FoundPos As Integer
   Dim FoundLine As Integer
   ' Find the text specified in the TextBox control.
   FoundPos = RichTextBox1.Find(Text1.Text, , , rtfWholeWord)
   ' Show message based on whether the text was found or not.
   If FoundPos <> -1 Then
      ' Returns number of line containing found text.
      FoundLine = RichTextBox1.GetLineFromChar(FoundPos)
      MsgBox "Word found on line " & CStr(FoundLine)
   Else
      MsgBox "Word not found."
   End If
End Sub
```

## GetLineFromChar Method

See Also Example

Returns the number of the line containing a specified character position in a **RichTextBox** control. Doesn't support named arguments.

**Important** This method requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**GetLineFromChar**(*charpos*)

The **GetLineFromChar** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **RichTextBox** control. |
| *charpos* | Required. A long integer that specifies the index of the character whose line you want to identify. The index of the first character in the **RichTextBox** control is 0. |

### Remarks

You use the **GetLineFromChar** method to find out which line in the text of a **RichTextBox** control contains a certain character position in the text. You might need to do this because the number of characters in each line of text can vary, making it very difficult to find out which line in the text contains a particular character, identified by its position in the text.

**See Also**

**RichTextBox** Control

■

## GetLineFromChar Method Example

This example finds a string in a **RichTextBox** control based on a word entered in a **TextBox** control. After it finds the specified string, it displays a message box that shows the number of the line containing the specified word. To try this example, put a **RichTextBox** control, a **CommandButton** control and a **TextBox** control on a form. Load a file into the **RichTextBox,** and paste this code into the Click event of the **CommandButton** control. Then run the example, enter a word in the **TextBox**, and click the **CommandButton**.

```
Private Sub Command1_Click()
   Dim FoundPos As Integer
   Dim FoundLine As Integer
   ' Find the text specified in the TextBox control.
   FoundPos = RichTextBox1.Find(Text1.Text, , , rtfWholeWord)
   ' Show message based on whether the text was found or not.
   If FoundPos <> -1 Then
      ' Returns number of line containing found text.
      FoundLine = RichTextBox1.GetLineFromChar(FoundPos)
      MsgBox "Word found on line " & CStr(FoundLine)
   Else
      MsgBox "Word not found."
   End If
End Sub
```

## SelHangingIndent, SelIndent, SelRightIndent Properties

See Also Example

Returns or sets the margin settings for the paragraph(s) in a **RichTextBox** control that either contain the current selection or are added at the current insertion point. Not available at design time.

**Important** These properties require the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**SelHangingIndent** [= *integer*]

*object*.**SelIndent** [= *integer*]

*object*.**SelRightIndent** [= *integer*]

The **SelHangingIndent**, **SelIndent**, and **SelRightIndent** properties syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *integer* | An integer that determines the amount of indent. These properties use the scale mode units of the **Form** object containing the **RichTextBox** control. |

### Remarks

For the affected paragraph(s), the **SelIndent** property specifies the distance between the left edge of the **RichTextBox** control and the left edge of the text that is selected or added. Similarly, the **SelRightIndent** property specifies the distance between the right edge of the **RichTextBox** control and the right edge of the text that is selected or added.

The **SelHangingIndent** property specifies the distance between the left edge of the first line of text in the selected paragraph(s) (as specified by the **SelIndent** property) and the left edge of subsequent lines of text in the same paragraph(s).

These properties return zero (0) if the selection spans multiple paragraphs with different margin settings.

**See Also**

**RichTextBox** Control
**SelAlignment** Property
**SelBullet** Property
**SelTabCount**, **SelTabs** Properties

■

## SelHangingIndent, SelIndent, SelRIghtIndent Properties Example

This example selects all the text in a **RichTextBox** control, then sets both the left and right indents to create margins.   To try this example, put a **RichTextBox** control, a **CommandButton** control, and a **TextBox** control on a form.   Load a file into the **RichTextBox,** and paste this code into the Click event of the **CommandButton** control.   Then run the example.

```
Private Sub Command1_Click()
   Dim Margins As Integer
   Margins = CInt(Text1.Text)
   With RichTextBox1
      .SelStart = 1
      .SelLength = Len(RichTextBox1.Text)
      .SelIndent = Margins
      .SelRightIndent = Margins
   End With
End Sub
```

## LoadFile Method

See Also Example

Loads an .RTF file or text file into a **RichTextBox** control. Doesn't support named arguments.

**Important** This method requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**LoadFile** *pathname*, *filetype*

The **LoadFile** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **RichTextBox** control. |
| *pathname* | Required. A string expression defining the path and filename of the file to load into the control. |
| *filetype* | Optional. An integer or constant that specifies the type of file loaded, as described in Settings. |

### Settings

The settings for *filetype* are:

| Constant | Value | Description |
|---|---|---|
| **rtfRTF** | 0 | (Default) RTF. The file loaded must be a valid .RTF file. |
| **rtfText** | 1 | Text. The **RichTextBox** control loads any text file. |

### Remarks

When loading a file with the **LoadFile** method, the contents of the loaded file replaces the entire contents of the **RichTextBox** control. This will cause the values of the **Text** and **RTFText** properties to change.

You can also use the **Input** function in Visual Basic and the **TextRTF** and **SelRTF** properties of the **RichTextBox** control to read .RTF files. For example, you can load the contents of an .RTF file to the **RichTextBox** control as follows:

```
Open "mytext.rtf" For Input As 1
RichTextBox1.TextRTF = Input$(LOF(1), 1)
```

**See Also**

**FileName** Property
**RichTextBox** Control
**SaveFile** Method
**SelRTF** Property
Supported RTF Codes
**TextRTF** Property

■

## LoadFile Method Example

This example displays a dialog box to choose an .RTF file, then loads that file into a **RichTextBox** control. To try this example, put a **RichTextBox** control, a **CommandButton** control, and a **CommonDialog** control on a form. Paste this code into the Click event of the **CommandButton** control. Then run the example.

```
Private Sub Command1_Click()
   CommonDialog1.Filter = "Rich Text Format files|*.rtf"
   CommonDialog1.ShowOpen
   RichTextBox1.LoadFile CommonDialog1.Filename, rtfRTF
End Sub
```

## BulletIndent Property

See Also

Returns or sets the amount of indent used in a **RichTextBox** control when **SelBullet** is set to **True**.

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**BulletIndent** [= *integer*]

The **BulletIndent** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *integer* | An integer that determines the amount of indent. These properties use the scale mode units of the **Form** object containing the **RichTextBox** control. |

### Remarks

The **BulletIndent** property returns **Null** if the selection spans multiple paragraphs with different margin settings.

**See Also**

**RichTextBox** Control

**SelBullet** Property

## SaveFile Method

See Also Example

Saves the contents of a **RichTextBox** control to a file. Doesn't support named arguments.

**Important** This method requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**SaveFile**(*pathname*, *filetype*)

The **SaveFile** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **RichTextBox** control. |
| *pathname* | Required. A string expression defining the path and filename of the file to receive the contents of the control. |
| *filetype* | Optional. An integer or constant that specifies the type of file loaded, as described in Settings. |

### Settings

The settings for *filetype* are:

| Constant | Value | Description |
|---|---|---|
| **rtfRTF** | 0 | (Default) RTF. The **RichTextBox** control saves its contents as an .RTF file. |
| **rtfText** | 1 | Text. The **RichTextBox** control saves its contents as a text file. |

### Remarks

You can also use the **Write** function in Visual Basic and the **TextRTF** and **SelRTF** properties of the **RichTextBox** control to write .RTF files. For example, you can save the highlighted contents of a **RichTextBox** control to an .RTF file as follows:

```
Open "mytext.rtf" For Output As 1
Print #1, RichTextBox1.SelRTF
```

**See Also**

**LoadFile** Method
**RichTextBox** Control
**SelRTF** Property
Supported RTF Codes
**TextRTF** Property

■

## SaveFile Method Example

This example displays a dialog box to choose an .RTF file to which you will save the contents of a **RichTextBox** control.   To try this example, put a **RichTextBox** control, a **CommandButton** control, and a **CommonDialog** control on a form.   Paste this code into the Click event of the **CommandButton** control.   Then run the example.

```
Private Sub Command1_Click()
   CommonDialog1.ShowSave
   RichTextBox1.SaveFile(CommonDialog1.Filename, rtfRTF)
End Sub
```

## SelAlignment Property

See Also Example

Returns or sets a value that controls the alignment of the paragraphs in a **RichTextBox** control. Not available at design time.

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**SelAlignment** [= *value*]

The **SelAlignment** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *value* | An integer or constant that determines paragraph alignment, as described in Settings. |

### Settings

The settings for *value* are:

| Constant | Value | Description |
|---|---|---|
| | **Null** | Neither. The current selection spans more than one paragraph with different alignments. |
| **rtfLeft** | 0 | (Default) Left. The paragraph is aligned along the left margin. |
| **rtfRight** | 1 | Right. The paragraph is aligned along the right margin. |
| **rtfCenter** | 2 | Center. The paragraph is centered between the left and right margins. |

### Remarks

The **SelAlignment** property determines paragraph alignment for all paragraphs that have text in the current selection or for the paragraph containing the insertion point if no text is selected.

To distinguish between the values of **Null** and 0 when reading this property at run time, use the **IsNull** function with the **If...Then...Else** statement. For example:

```
If IsNull(RichTextBox1.SelAlignment) = True Then
   ' Code to run when selection is mixed.
ElseIf RichTextBox1.SelAlignment = 0 Then
   ' Code to run when selection is left aligned.
...
End If
```

**See Also**

**RichTextBox** Control
**SelHangingIndent**, **SelIndent**, **SelRightIndent** Properties
**SelBullet** Property
**SelTabCount**, **SelTabs** Properties

■

## SelAlignment Property Example

This example uses an array of **OptionButton** controls to change the paragraph alignment of selected text in a **RichTextBox** control, but only if text is selected. The indices of the controls in the array correspond to settings for the **SelAlignment** property. To try this example, put a **RichTextBox** control and three **OptionButton** controls on a form. Give all three of the **OptionButton** controls the same name and set their **Index** property to 0, 1, and 2. Paste this code into the Click event of the **OptionButton** control. Then run the example.

```
Private Sub Option1_Click(Index As Integer)
   If RichTextBox1.SelLength > 0 Then
      RichTextBox1.SelAlignment = Index
   End If
End Sub
```

## SelBold, SelItalic, SelStrikethru, SelUnderline Properties

See Also

Return or set font styles of the currently selected text in a **RichTextBox** control. The font styles include the following formats: **Bold**, *Italic*, ~~Strikethru~~, and Underline. Not available at design time.

**Important** These properties require the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**SelBold** [= *value*]

*object*.**SelItalic** [= *value*]

*object*.**SelStrikethru** [= *value*]

*object*.**SelUnderline** [= *value*]

The **SelBold**, **SelItalic**, **SelStrikethru**, and **SelUnderline** properties syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *value* | A Boolean expression or constant that determines the font style, as described in Settings. |

### Settings

The settings for *value* are:

| Setting | Description |
|---|---|
| **Null** | Neither. The selection or character following the insertion point contains characters that have a mix of the appropriate font styles. |
| **True** | All the characters in the selection or character following the insertion point have the appropriate font style. |
| **False** | (Default) None of the characters in the selection or character following the insertion point have the appropriate font style. |

### Remarks

These properties behave like the **Bold**, **Italic**, **Strikethru**, and **Underline** properties of a **Font** object. The **RichTextBox** control has a **Font** property and therefore the ability to apply font styles to all the text in the control through the properties of the control's **Font** object. Use these properties to apply font styles to selected text or to characters entered at the insertion point.

Typically, you access these properties by creating a toolbar in your application with buttons to toggle these properties individually.

To distinguish between the values of **Null** and **False** when reading these properties at run time, use the **IsNull** function with the **If...Then...Else** statement. For example:

```
If IsNull(RichTextBox1.SelBold) = True Then
   ' Code to run when selection is mixed.
ElseIf RichTextBox1.SelBold = False Then
   ' Code to run when selection is not bold.
...
End If
```

**See Also**

- **RichTextBox** Control
- **SelCharOffset** Property
- **SelColor** Property
- **SelFontName** Property
- **SelFontSize** Property

## SelChange Event

See Also      Example

Occurs when the current selection of text in the **RichTextBox** control has changed or the insertion point has moved.

### Syntax

**Private Sub** *object*_**SelChange(**[*index* **As Integer**]**)**

The SelChange event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *index* | An integer that uniquely identifies a control if it's in a control array. |

### Remarks

You can use the SelChange event to check the various properties that give information about the current selection (such as **SelBold**) so you can update buttons in a toolbar, for example.

**See Also**

**RichTextBox** Control

■

## SelChange Event Example

This example checks the size of the current selection to see if the menu commands for cutting or copying text to the Clipboard should be enabled.   To try this example, put a **RichTextBox** control and three **Menu** controls on a form to create a menu with commands to cut and copy.   Paste this code into the SelChange event of the **RichTextBox** control.   Then run the example.

```
Private Sub RichTextBox1_SelChange()
   If RichTextBox1.SelLength > 0 Then
      EditCutMenu.Enabled = True
      EditCopyMenu.Enabled = True
   Else
      EditCutMenu.Enabled = False
      EditCopyMenu.Enabled = False
   End If
End Sub
```

## SelCharOffset Property

See Also Example

Returns or sets a value that determines whether text in the **RichTextBox** control appears on the baseline (normal), as a superscript above the baseline, or as a subscript below the baseline. Not available at design time.

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**SelCharOffset** [= *offset*]

The **SelCharOffset** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *offset* | An integer that determines how far the characters in the current selection or that following the insertion point are offset from the baseline of the text, as described in Settings. |

### Settings

The settings for *offset* are:

| Setting | Description |
|---|---|
| **Null** | Neither. The selection has a mix of characters with different offsets. |
| 0 | (Default) Normal. The characters all appear on the normal text baseline. |
| Positive integer | Superscript. The characters appear above the baseline by the number of twips specified. |
| Negative integer | Subscript. The characters appear below the baseline by the number of twips specified. |

### Remarks

To distinguish between the values of **Null** and 0 when reading this property at run time, use the **IsNull** function with the **If...Then...Else** statement. For example:

```
If IsNull(RichTextBox1.SelCharOffset) = True Then
   ' Code to run when selection is mixed.
ElseIf RichTextBox1.SelCharOffset = 0 Then
   ' Code to run when selection is all on the baseline.
...
End If
```

**See Also**

**RichTextBox** Control
**SelBold**, **SelItalic**, **SelStrikethru**, **SelUnderline** Properties
**SelColor** Property
**SelFontName** Property
**SelFontSize** Property

■

## SelCharOffset Property Example

This example uses a scroll bar to move selected text above or below the baseline. The minimum and maximum amount of offset is established by the font size of the text within the **RichTextBox** control. To try this example, put a **RichTextBox** control and a **VScrollBar** control on a form. Paste this code into the Change event of the **VScrollBar** control. Then run the example.

```
Private Sub VScroll1_Change ()
   VScroll1.Max = RichTextBox1.SelFontSize
   VScroll1.Min = -(VScroll1.Max)
   RichTextBox1.SelCharOffset = VScroll1.Value
End Sub
```

## SelColor Property

See Also    Example

Returns or sets a value that determines the color of text in the **RichTextBox** control. Not available at design time.

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**SelColor** [= *color*]

The **SelColor** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *color* | A value that specifies a color, as described in Settings. |

### Settings

The settings for *color* are:

| Setting | Description |
|---|---|
| **Null** | The text contains a mixture of different color settings. |
| RGB colors | Colors specified in code with the **RGB** or **QBColor** functions. |
| System | Colors specified with the system color constants in the Visual Basic object library in the Object Browser. The color of the text then matches user selections for the specified constant in the Windows Control Panel. |

### Remarks

If there is no text selected in the **RichTextBox** control, setting this property determines the color of all new text entered at the current insertion point.

**See Also**

**RichTextBox** Control
**SelBold**, **SelItalic**, **SelStrikethru**, **SelUnderline** Properties
**SelCharOffset** Property
**SelFontName** Property
**SelFontSize** Property

■

## SelColor Property Example

This example displays a color dialog box from a **CommonDialog** control to specify the color of selected text in a **RichTextBox** control.   To try this example, put a **RichTextBox** control, a **CommandButton** control, and a **CommonDialog** control on a form.   Paste this code into the Click event of the **CommandButton** control.   Then run the example.

```
Private Sub Command1_Click()
   CommonDialog1.ShowColor
   RichTextBox1.SelColor = CommonDialog1.Color
End Sub
```

## SelFontName Property

See Also Example

Returns or sets the font used to display the currently selected text or the character(s) immediately following the insertion point in the **RichTextBox** control. Not available at design time.

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**SelFontName** [= *string*]

The **SelFontName** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *string* | A string expression that identifies a font installed on the system. |

### Remarks

The **SelFontName** property returns **Null** if the selected text contains different fonts.

**See Also**

**RichTextBox** Control

**SelBold**, **SelItalic**, **SelStrikethru**, **SelUnderline** Properties

**SelCharOffset** Property

**SelColor** Property

**SelFontSize** Property

■
## SelFontName Property Example

This example displays a font dialog box from a **CommonDialog** control to specify font attributes of selected text in a **RichTextBox** control. To try this example, put a **RichTextBox** control, a **CommandButton** control, and a **CommonDialog** control on a form. Paste this code into the Click event of the **CommandButton** control. Then run the example.

```
Private Sub Command1_Click ()
   CommonDialog1.Flags = cdlCFBoth
   CommonDialog1.ShowFont
   With RichTextBox1
      .SelFontName = CommonDialog1.FontName
      .SelFontSize = CommonDialog1.FontSize
      .SelBold = CommonDialog1.FontBold
      .SelItalic = CommonDialog1.FontItalic
      .SelStrikethru = CommonDialog1.FontStrikethru
      .SelUnderline = CommonDialog1.FontUnderline
   End With
End Sub
```

## SelFontSize Property

See Also Example

Returns or sets a value that specifies the size of the font used to display text in a **RichTextBox** control. Not available at design time.

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**SelFontSize** [= *points*]

The **SelFontSize** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *points* | An integer that specifies the size in points of the currently selected text or the characters immediately following the insertion point. |

### Remarks

The maximum value for **SelFontSize** is 2160 points.

In general, you should change the **SelFontName** property before you set the size and style attributes. However, when you set TrueType fonts to smaller than 8 points, you should set the point size to 3 with the **SelFontSize** property, then set the **SelFontName** property, and then set the size again with the **SelFontSize** property.

**Note** Available fonts depend on your system configuration, display devices, and printing devices, and therefore may vary from system to system.

The **SelFontSize** property returns **Null** if the selected text contains different font sizes.

**See Also**

**RichTextBox** Control
**SelBold**, **SelItalic**, **SelStrikethru**, **SelUnderline** Properties
**SelCharOffset** Property
**SelColor** Property
**SelFontName** Property

▪

## SelFontSize Property Example

This example displays a font dialog box from a **CommonDialog** control to specify font attributes of selected text in a **RichTextBox** control.   To try this example, put a **RichTextBox** control, a **CommandButton** control, and a **CommonDialog** control on a form.   Paste this code into the Click event of the **CommandButton** control.   Then run the example.

```
Private Sub Command1_Click ()
   CommonDialog1.Flags = Both
   CommonDialog1.ShowFont
   With RichTextBox1
      .SelFontName = CommonDialog1.FontName
      .SelFontSize = CommonDialog1.FontSize
      .SelBold = CommonDialog1.FontBold
      .SelItalic = CommonDialog1.FontItalic
      .SelStrikethru = CommonDialog1.FontStrikethru
      .SelUnderline = CommonDialog1.FontUnderline
   End With
End Sub
```

## SelBullet Property

See Also Example

Returns or sets a value that determines if a paragraph in the **RichTextBox** control containing the current selection or insertion point has the bullet style. Not available at design time.

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**SelBullet** [= *value*]

The **SelBullet** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *value* | An integer or constant that determines the bullet style of the paragraph(s), as described in Settings. |

### Settings

The settings for *value* are:

| Setting | Description |
|---|---|
| **Null** | Neither. The selection spans more than one paragraph and contains a mixture of bullet and nonbullet styles. |
| **True** | The paragraphs in the selection have the bullet style. |
| **False** | (Default) The paragraphs in the selection don't have the bullet style. |

### Remarks

Use the **SelBullet** property to build a list of bulleted items in a **RichTextBox** control.

To distinguish between the values of **Null** and **False** when reading this property at run time, use the **IsNull** function with the **If...Then...Else** statement. For example:

```
If IsNull(RichTextBox1.SelBullet) = True Then
   ' Code to run when selection is mixed.
ElseIf RichTextBox1.SelBullet = False Then
   ' Code to run when selection doesn't have bullet style.
...
End If
```

**See Also**

**RichTextBox** Control
**SelAlignment** Property
**SelHangingIndent**, **SelIndent**, **SelRightIndent** Properties
**SelTabCount**, **SelTabs** Properties

■

## SelBullet Property Example

This example changes the state of a **CheckBox** control on a form to show the bullet status of selected text in a **RichTextBox** control.   To try this example, put a **RichTextBox** control and a **CheckBox** control on a form.   Paste this code into the SelChange event of the **RichTextBox** control.   Then run the example.

```
Private Sub RichTextBox1_SelChange()
   If IsNull(RichTextBox1.SelBullet) = True Then
      Check1.Value = vbGrayed
   ElseIf RichTextBox1.SelBullet = True Then
      Check1.Value = vbChecked
   ElseIf RichTextBox1.SelBullet = False Then
      Check1.Value = vbUnchecked
   End If
End Sub
```

## SelPrint Method

See Also Example

Sends formatted text in a **RichTextBox** control to a device for printing.

**Important** This method requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**SelPrint(***hdc***)**

The **SelPrint** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *hdc* | The device context of the device you plan to use to print the contents of the control. |

### Remarks

If text is selected in the **RichTextBox** control, the **SelPrint** method sends only the selected text to the target device. If no text is selected, the entire contents of the **RichTextBox** are sent to the target device.

The **SelPrint** method does not print text from the **RichTextBox** control. Rather, it sends a copy of formatted text to a device which can print the text. For example, you can send the text to the **Printer** object using code as follows:

```
RichTextBox1.SelPrint(Printer.hDC)
```

Notice that the **hDC** property of the **Printer** object is used to specify the device context argument of the **SelPrint** method.

**Note** If you use the **Printer** object as the destination of the text from the **RichTextBox** control, you must first initialize the device context of the **Printer** object by printing something like a zero-length string.

**See Also**

**RichTextBox** Control

▪

## SelPrint Method Example

This example prints the formatted text in a **RichTextBox** control.   To try this example, put a **RichTextBox** control, a **CommonDialog** control, and a **CommandButton** control on a form.   Paste this code into the Click event of the **CommandButton** control.   Then run the example.

```
Private Sub Command1_Click()
   CommonDialog1.Flags = cdlPDReturnDC + cdlPDNoPageNums
   If RichTextBox1.SelLength = 0 Then
      CommonDialog1.Flags = CommonDialog1.Flags + cdlPDAllPages
   Else
      CommonDialog1.Flags = CommonDialog1.Flags + cdlPDSelection
   End If
   CommonDialog1.ShowPrinter
   RichTextBox1.SelPrint CommonDialog1.hDC
End Sub
```

## SelTabCount, SelTabs Properties

See Also Example

Returns or sets the number of tabs and the absolute tab positions of text in a **RichTextBox** control. Not available at design time.

**Important** These properties require the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**SelTabCount** [= *count*]

*object*.**SelTabs(***index***)** [= *location*]

The **SelTabCount** and **SelTabs** properties syntaxes have these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *count* | An integer that determines the number of tab positions in the selected paragraph(s) or in those paragraph(s) following the insertion point. |
| *index* | An integer that identifies a specific tab. The first tab location has an index of zero (0). The last tab location has an index equal to **SelTabCount** minus 1. |
| *location* | An integer that specifies the location of the designated tab. The units used to express tab positions are determined by the scale mode of the **Form** object or other object containing the **RichTextBox** control. |

### Remarks

By default, pressing TAB when typing in a **RichTextBox** control causes focus to move to the next control in the tab order, as specified by the **TabIndex** property. One way to insert a tab in the text is by pressing CTRL+TAB. However, users who are accustomed to working with word processors may find the CTRL+TAB key combination contrary to their experience. You can enable use of the TAB key to insert a tab in a **RichTextBox** control by temporarily switching the **TabStop** property of all the controls on the **Form** object to **False** while the **RichTextBox** control has focus. For example:

```
Private Sub RichTextBox1_GotFocus()
   ' Ignore errors for controls without the TabStop property.
   On Error Resume Next
   ' Switch off the change of focus when pressing TAB.
   For Each Control In Controls
      Control.TabStop = False
   Next Control
End Sub
```

Make sure to reset the **TabStop** property of the other controls when the **RichTextBox** control loses focus.

**See Also**

**RichTextBox** Control
**SelAlignment** Property
**SelHangingIndent**, **SelIndent**, **SelRightIndent** Properties
**SelBullet** Property

▪

## SelTabCount, SelTabs Properties Example

This example sets the number of tabs in a **RichTextBox** control to a total of five and then sets the positions of the tabs to multiples of five.   To try this example, put a **RichTextBox** control and a **CommandButton** control on a form.   Paste this code into the Click event of the **CommandButton** control.   Then run the example.

```
Private Sub Command1_Click()
   With RichTextBox1
      .SelTabCount = 5
      For X = 0 To .SelTabCount - 1
         .SelTabs(X) = 5 * X
      Next X
   End With
End Sub
```

## SelRTF Property

See Also Example

Returns or sets the text (in .RTF format) in the current selection of a **RichTextBox** control. Not available at design time.

---

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

---

### Syntax

*object*.**SelRTF** [= *string*]

The **SelRTF** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *string* | A string expression in .RTF format. |

### Remarks

Setting the **SelRTF** property replaces any selected text in the **RichTextBox** control with the new string. This property returns a zero-length string ("") if no text is selected in the control.

You can use the **SelRTF** property along with the **Print** function to write .RTF files.

**See Also**

**RichTextBox** Control
Supported RTF Codes
**TextRTF** Property

■

## SelRTF Property Example

This example saves the highlighted contents of a **RichTextBox** control to an .RTF file.   To try this example, put a **RichTextBox** control and a **CommandButton** control on a form.   Paste this code into the Click event of the **CommandButton** control.   Then run the example.

```
Private Sub Command1_Click ()
   Open "mytext.rtf" For Output As 1
   Print #1, RichTextBox1.SelRTF
   Close 1
End Sub
```

## Span Method

See Also Example

Selects text in a **RichTextBox** control based on a set of specified characters. Doesn't support named arguments.

**Important** This method requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**Span** *characterset*, *forward*, *negate*

The **Span** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to a **RichTextBox** control. |
| *characterset* | Required. A string expression that specifies the set of characters to look for when extending the selection, based on the value of *negate*. |
| *forward* | Optional. A Boolean expression that determines which direction the insertion point moves, as described in Settings. |
| *negate* | Optional. A Boolean expression that determines whether the characters in *characterset* define the set of target characters or are excluded from the set of target characters, as described in Settings. |

### Settings

The settings for *forward* are:

| Setting | Description |
|---|---|
| **True** | (Default) Selects text from the current insertion point or the beginning of the current selection forward, toward the end of the text. |
| **False** | Selects text from the current insertion point or the beginning of the current selection backward, toward the start of the text. |

The settings for *negate* are:

| Setting | Description |
|---|---|
| **True** | The characters included in the selection are those which do not appear in the *characterset* argument. The selection stops at the first character found that appears in the *characterset* argument. |
| **False** | (Default) The characters included in the selection are those which appear in the *characterset* argument. The selection stops at the first character found that does not appear in the *characterset* argument. |

### Remarks

The **Span** method is primarily used to easily select a word or sentence in the **RichTextBox** control.

If the **Span** method cannot find the specified characters based on the values of the arguments, then the current insertion point or selection remains unchanged.

The **Span** method does not return any data.

**See Also**

**RichTextBox** Control
**UpTo** Method

■

## Span Method Example

This example defines a pair of keyboard shortcuts that selects text in a **RichTextBox** control to the end of a sentence (CTRL+S) or the end of a word (CTRL+W). To try this example, put a **RichTextBox** control on a form. Paste this code into the KeyUp event of the **RichTextBox** control. Then run the example.

```
Private Sub RichTextBox1_KeyUp (KeyCode As Integer, Shift As Integer)
   If Shift = vbCtrlMask Then
      Select Case KeyCode
         ' If Ctrl+S:
         Case vbKeyS
            ' Select to the end of the sentence.
            RichTextBox1.Span ".?!:", True, True
            ' Extend selection to include punctuation.
            RichTextBox1.SelLength = RichTextBox1.SelLength + 1
         ' If Ctrl+W:
         Case vbKeyW
            ' Select to the end of the word.
          RichTextBox1.Span " ,;:.?!", True, True
      End Select
   End If
End Sub
```

## TextRTF Property

See Also Example

Returns or sets the text of a **RichTextBox** control, including all .RTF code.

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**TextRTF** [= *string*]

The **TextRTF** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *string* | A string expression in .RTF format. |

### Remarks

Setting the **TextRTF** property replaces the entire contents of a **RichTextBox** control with the new string.

You can use the **TextRTF** property along with the **Print** function to write .RTF files. The resulting file can be read by any other word processor capable of reading RTF-encoded text.

**See Also**

- **RichTextBox** Control
- **SaveFile** Method
- **SelRTF** Property
- Supported RTF Codes

■

## TextRTF Property Example

This example saves the entire contents of a **RichTextBox** control to an .RTF file.   To try this example, put a **RichTextBox** control and a **CommandButton** control on a form.   Paste this code into the Click event of the **CommandButton** control.   Then run the example.

```
Private Sub Command1_Click ()
   Open "mytext.rtf" For Output As 1
   Print #1, RichTextBox1.TextRTF
   Close 1
End Sub
```

## Upto Method

See Also Example

Moves the insertion point up to, but not including, the first character that is a member of the specified character set in a **RichTextBox** control.  Doesn't support named arguments.

**Important**  This method requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**Upto**(*characterset*, *forward, negate*)

The **Upto** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Required.  An object expression that evaluates to a **RichTextBox** control. |
| *characterset* | Required.  A string expression that specifies the set of characters to look for when moving the insertion point, based on the value of *negate*. |
| *forward* | Optional.  A Boolean expression that determines which direction the insertion point moves, as described in Settings. |
| *negate* | Optional.  A Boolean expression that determines whether the characters in *characterset* define the set of target characters or are excluded from the set of target characters, as described in Settings. |

### Settings

The settings for *forward* are:

| Setting | Description |
|---|---|
| **True** | (Default)  Moves the insertion point forward, toward the end of the text. |
| **False** | Moves the insertion point backward, toward the start of the text. |

The settings for *negate* are:

| Setting | Description |
|---|---|
| **True** | The characters not specified in the *characterset* argument are used to move the insertion point. |
| **False** | (Default)  The characters specified in the *characterset* argument are used to move the insertion point. |

**See Also**

**RichTextBox** Control

**Span** Method

▪

## Upto Method Example

This example defines a pair of keyboard shortcuts that moves the insertion point in a **RichTextBox** control to the end of a sentence (ALT+S) or the end of a word (ALT+W). To try this example, put a **RichTextBox** control on a form. Paste this code into the KeyUp event of the **RichTextBox** control. Then run the example.

```
Private Sub RichTextBox1_KeyUp (KeyCode As Integer, Shift As Integer)
   If Shift = vbAltMask Then
      Select Case KeyCode
         ' If Alt+S:
         Case vbKeyS
            ' Move insertion point to the end of the sentence.
            RichTextBox1.Upto ".?!:", True, False
         ' If Alt+W:
         Case vbkeyW
            ' Move insertion point to the end of the word.
            RichTextBox1.Upto " .?!:", True, False
      End Select
   End If
End Sub
```

## RichTextBox Control Constants

See Also

**Appearance** Property

| Constant | Value | Description |
|---|---|---|
| **rtfFlat** | 0 | Flat. Paints without visual effects. |
| **rtfThreeD** | 1 | (Default). 3D. Paints with three-dimensional effects. |

**Find** Method

| Constant | Value | Description |
|---|---|---|
| **rtfWholeWord** | 2 | Determines if a match is based on a whole word or a fragment of a word. |
| **rtfMatchCase** | 4 | Determines if a match is based on the case of the specified string as well as the text of the string. |
| **rtfNoHighlight** | 8 | Determines if a match appears highlighted in the **RichTextBox** control. |

**LoadFile**, **SaveFile** Methods

| Constant | Value | Description |
|---|---|---|
| **rtfRTF** | 0 | (Default) RTF. The file loaded must be a valid .RTF file (**LoadFile** method) or the contents in the control are saved to an .RTF file (**SaveFile** method). |
| **rtfText** | 1 | Text. The **RichTextBox** control loads any text file (**LoadFile** method) or the contents in the control are saved to a text file (**SaveFile** method). |

**MousePointer** Property

| Constant | Value | Description |
|---|---|---|
| **rtfDefault** | 0 | (Default) Shape determined by the object. |
| **rtfArrow** | 1 | Arrow. |
| **rtfCross** | 2 | Cross (cross-hair pointer). |
| **rtfIbeam** | 3 | I Beam. |
| **rtfIcon** | 4 | Icon (small square within a square). |
| **rtfSize** | 5 | Size (four-pointed arrow pointing north, south, east, and west). |
| **rtfSizeNESW** | 6 | Size NE SW (double arrow pointing northeast and southwest). |
| **rtfSizeNS** | 7 | Size N S (double arrow pointing north and south). |
| **rtfSizeNWSE** | 8 | Size NW, SE. |
| **rtfSizeEW** | 9 | Size E W (double arrow pointing east and west). |
| **rtfUpArrow** | 10 | Up Arrow. |
| **rtfHourglass** | 11 | Hourglass (wait). |
| **rtfNoDrop** | 12 | No Drop. |
| **rtfArrowHourglass** | 13 | Arrow and hourglass. (Only available in 32-bit Visual Basic 4.0.) |
| **rtfArrowQuestion** | 14 | Arrow and question mark. (Only available in 32-bit Visual Basic 4.0.) |
| **rtfSizeAll** | 15 | Size all. (Only available in 32-bit Visual Basic 4.0.) |
| **rtfCustom** | 99 | Custom icon specified by the MouseIcon property. |

**SelAlignment** Property

| Constant | Value | Description |
|---|---|---|
| **rtfLeft** | 0 | (Default) Left. The paragraph is aligned along the left margin. |
| **rtfRight** | 1 | Right. The paragraph is aligned along the right margin. |

| | | |
|---|---|---|
| **rtfCenter** | 2 | Center. The paragraph is centered between the left and right margins. |

**ScrollBars** Property

| Constant | Value | Description |
|---|---|---|
| **rtfNone** | 0 | (Default) None. |
| **rtfHorizontal** | 1 | Horizontal scroll bar only. |
| **rtfVertical** | 2 | Vertical scroll bar only. |
| **rtfBoth** | 3 | Both horizontal and vertical scroll bars. |

**See Also**

**Appearance** Property
**Find** Method
**LoadFile** Method
**MousePointer** Property
**RichTextBox** Control
**SaveFile** Method
**SelAlignment** Property
**ScrollBars** Property
Visual Basic Custom Control Constants
Windows 95 Controls Constants

## Supported RTF Codes

See Also

The **RichTextBox** control recognizes the following RTF (Rich Text Format) codes. All other RTF codes are ignored by the control when loading text.

| RTF Code | Description | RTF Code | Description |
|---|---|---|---|
| - | OptionalHyphen | objcropl | CropLeft |
| \n | EndParagraph | objcropr | CropRight |
| \r | EndParagraph | objcropt | CropTop |
| _ | NonBreakingHyphen | objdata | ObjectData |
| \| | FormulaCharacter | object | Object |
| ~ | NonBreakingSpace | objemb | ObjectEmbedded |
| ansi | CharSetAnsi | objh | Height |
| b | Bold | objicemb | ObjectMacICEmbedder |
| bin | BinaryData | objlink | ObjectLink |
| blue | ColorBlue | objname | ObjectName |
| bullet | ANSI Character 149 | objpub | ObjectMacPublisher |
| cb | ColorBackground | objscalex | ScaleX |
| cell | Cell | objscaley | ScaleY |
| cf | ColorForeground | objsetsize | ObjectSetSize |
| colortbl | ColorTable | objsub | ObjectMacSubscriber |
| cpg | CodePage | objw | Width |
| deff | DefaultFont | par | EndParagraph |
| deflang | DefaultLanguage | pard | ParagraphDefault |
| deftab | DefaultTabWidth | pc | CharSetPc |
| deleted | Deleted | pca | CharSetPs2 |
| dibitmap | PictureWindowsDIB | piccropb | CropBottom |
| dn | Down | piccropl | CropLeft |
| dy | TimeDay | piccropr | CropRight |
| emdash | ANSI Character 151 | piccropt | CropTop |
| endash | ANSI Character 150 | pich | Height |
| f | FontSelect | pichgoal | DesiredHeight |
| fbidi | FontFamilyBidi | picscalex | ScaleX |
| fchars | FollowingPunct | picscaley | ScaleY |
| fcharset | CharSet | pict | Picture |
| fdecor | FontFamilyDecorative | picw | Width |
| fi | IndentFirst | picwgoal | DesiredWidth |
| field | Field | plain | CharacterDefault |
| fldinst | FieldInstruction | pmmetafile | PictureOS2Metafile |
| fldrslt | FieldResult | pn | ParaNum |
| fmodern | FontFamilyModern | pnindent | ParaNumIndent |
| fname | RealFontName | pnlvlblt | ParaNumBullet |
| fnil | FontFamilyDefault | pntext | ParaNumText |
| fontemb | FontEmbedded | pntxta | ParaNumAfter |
| fontfile | FontFile | pntxtb | ParaNumBefore |
| fonttbl | FontTable | protect | Protect |
| footer | NullDestination (Footer) | qc | AlignCenter |
| footerf | NullDestination (Footer, first) | ql | AlignLeft |

| | | | |
|---|---|---|---|
| footerl | NullDestination (Footer, left) | qr | AlignRight |
| footerr | NullDestination (Footer, right) | rdblquote | ANSI Character 34 |
| footnote | NullDestination (footnote) | red | ColorRed |
| fprq | Pitch | result | ObjectResult |
| froman | FontFamilyRoman | revauth | RevAuthor |
| fs | FontSize | revised | Revision |
| fscript | FontFamilyScript | ri | IndentRight |
| fswiss | FontFamilySwiss | row | Row |
| ftech | FontFamilyTechnical | rquote | ANSI Character 39 |
| ftncn | NullDestination (Footnote cont.) | rtf | Rtf |
| ftnsep | NullDestination (Footnote separ) | rtlch | RightToLeftChars |
| ftnsepc | NullDestination (Footnote cont. separ) | rtldoc | RightToLeftDocument |
| green | ColorGreen | rtlmark | DisplayRightToLeft |
| header | NullDestination (Header) | rtlpar | RightToLeftParagraph |
| headerf | NullDestination (Header, first) | sec | TimeSecond |
| headerl | NullDestination (Header, left) | sect | EndSection |
| headerr | NullDestination (Header, right) | sectd | SectionDefault |
| horzdoc | HorizontalRender | strike | StrikeOut |
| hr | TimeHour | stylesheet | StyleSheet |
| i | Italic | sub | Subscript |
| info | DocumentArea (Info fields) | super | Superscript |
| intbl | InTable | tb | TabPosition |
| lang | Language | tc | NullDestination (Table of contents) |
| lchars | LeadingPunct | tx | TabPosition |
| ldblquote | ANSI Character 34 | ul | Underline |
| li | IndentLeft | uld | UnderlineDotted |
| line | SoftBreak | uldash | UnderlineDash |
| lquote | ANSI Character 39 | uldashd | UnderlineDashDotted |
| ltrch | LeftToRightChars | uldashdd | UnderlineDashDotDotted |
| ltrdoc | LeftToRightDocument | uldb | UnderlineDouble |
| ltrmark | DisplayLeftToRight | ulhair | UnderlineHairline |
| ltrpar | LeftToRightParagraph | ulnone | StopUnderline |
| mac | CharSetMacintosh | ulth | UnderlineThick |
| macpict | PictureQuickDraw | ulw | UnderlineWord |
| margl | MarginLeft | ulwave | UnderlineWave |
| marglsxn | SectionMarginLeft | up | Up |
| margr | MarginRight | v | HiddenText |
| margrsxn | SectionMarginRight | vertdoc | VerticalRender |
| min | TimeMinute | wbitmap | PictureWindowsBitmap |
| mo | TimeMonth | wbmbitspixel | BitmapBitsPerPixel |
| nocwrap | NoWordBreak | wbmplanes | BitmapNumPlanes |
| nooverflow | NoOverflow | wbmwidthbytes | BitmapWidthBytes |
| nosupersub | NoSuperSub | wmetafile | PictureWindowsMetafile |
| nowwrap | NoWordWrap | xe | NullDestination (index |

| | | | |
|---|---|---|---|
| | | | entry) |
| objautlink | ObjectAutoLink | yr | TimeYear |
| objclass | ObjectClass | zwj | ZeroWidthJoiner |
| objcropb | CropBottom | zwnj | ZeroWidthNonJoiner |

**See Also**

**FileName** Property
**LoadFile** Method
**RichTextBox** Control
**SaveFile** Method
**SelRTF** Property
**TextRTF** Property

## CellHeight, CellWidth Properties, Picture Clip Control

See Also

Returns the height and width, in pixels, of a **Picture Clip** control's **GraphicCell** property.

### Syntax

*object*.**CellHeight**

*object*.**CellWidth**

The *object* placeholder represents an object expression that evaluates to a **PictureClip** control.

### Remarks

The **CellHeight** and **CellWidth** properties are dependent upon the **Cols** and **Rows** properties of a **PictureClip** control. For example, dividing a picture into four columns and four rows would result in a **GraphicCell** that is twice the size of the same picture divided into eight columns and eight rows.

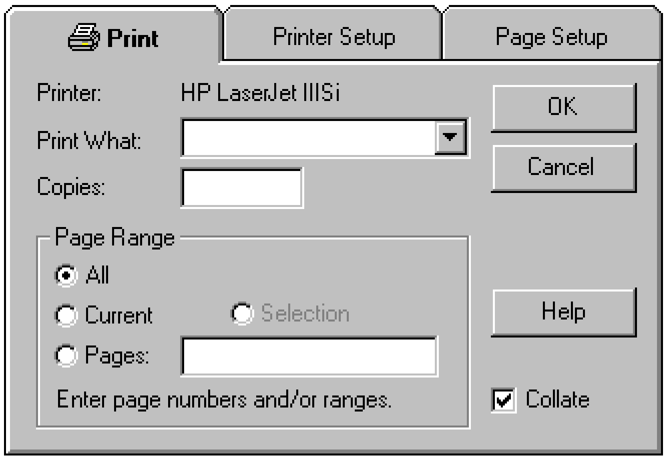
**See Also**

**Cols, Rows** Properties, Picture Clip Control
**GraphicCell** Propery, Picture Clip Control

## SSTab Control

See Also Properties Methods Events Constants

The **SSTab** control provides an easy way of presenting several dialogs or screens of information on a single form using the same interface seen in many commercial Microsoft Windows applications.



Printer: HP LaserJet IIISi

[✓ All]

[✓ Collate]

### Syntax

**SSTab**

### Remarks

The **SSTab** control provides a group of tabs, each of which acts as a container for other controls. Only one tab is active in the control at a time, displaying the controls it contains to the user while hiding the controls in the other tabs. Using the properties of this control, you can:

- Determine the number of tabs.
- Organize the tabs into more than one row.
- Set the text for each tab.
- Display a graphic on each tab.
- Determine the style of tabs used.
- Set the size of each tab.

To use this control, you must first decide how you want to organize the controls you will place into various tabs. Set the **Tabs** and **TabsPerRow** properties to create the tabs and organize them into rows. Then select each tab at design time by clicking the tab. For each tab, draw the controls you want displayed when the user selects that tab. Set the **Caption, Picture, TabHeight,** and **TabMaxWidth** properties as needed to customize the top part of the tab.

At run time, users can navigate between tabs by either pressing CTRL+TAB or by using mnemonics defined in the caption of each tab.

You can also customize the entire **SSTab** control using the **Style, ShowFocusRect, TabOrientation,** and **WordWrap** properties.

---

**Distribution Note** The **SSTab** control is found in the TABCTL32.OCX file (32-bit version) or in TABCTL16.OCX (16-bit version). To use the **SSTab** control in your application, you must add the control's OCX file to the project. When distributing your application, install the appropriate OCX file in the user's Microsoft Windows SYSTEM directory. For more information on how to add a custom control to a project, see the *Programmer's Guide*.

---

**See Also**

**TabStrip** Control

■

## SSTab Control Properties

**BackColor** Property
**Caption** Property
**Container** Property
**DragIcon** Property
**DragMode** Property
**Enabled** Property
**Font** Property
**ForeColor** Property
**Height** Property
**HelpContextID** Property
**hWnd** Property
**Index** Property
**Left** Property
**MouseIcon** Property
**MousePointer** Property
**Name** Property
**Object** Property
**Parent** Property
**Picture** Property
**Rows** Property
**ShowFocusRect** Property
**Style** Property
**Tab** Property
**TabCaption** Property
**TabEnabled** Property
**TabHeight** Property
**TabIndex** Property
**TabMaxWidth** Property
**TabOrientation** Property
**TabPicture** Property
**Tabs** Property (**SSTab** Control)
**TabsPerRow** Property
**TabStop** Property
**TabVisible** Property
**Tag** Property
**Top** Property
**Visible** Property
**WhatsThisHelpID** Property
**Width** Property
**WordWrap** Property

■
## SSTab Control Methods

**Drag** Method
**Move** Method
**SetFocus** Method
**ShowWhatsThis** Method
**ZOrder** Method

■

## SSTab Control Events

Click Event
DblClick Event
DragDrop Event
DragOver Event
GotFocus Event
KeyDown Event
KeyPress Event
KeyUp Event
LostFocus Event
MouseDown Event
MouseMove Event
MouseUp Event

## Click Event (SSTab Control)

See Also Example

Occurs when the user selects a tab on an **SSTab** control.

### Syntax

**Sub** *object*_**Click** ([*index* **As Integer**], p*revioustab* **As Integer**)

The Click event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *Index* | An integer that uniquely identifies a control if it is in a control array. |
| *previoustab* | A numeric expression that identifies the tab that was previously active. |

### Remarks

Use the Click event to determine when a user clicks a tab to make it the active tab. When a tab receives a Click event, that tab becomes the active tab and the controls placed on it at design time appear.

With the p*revioustab* argument, you can check for changes made when the user clicks another tab.

Use the **Tab** property to determine the current tab.

**See Also**

**SSTab** Control

■

## Click Event (SSTab Control) Example

This example saves preferences information from two tabs of an **SSTab** control as soon as the user selects a different tab.

```
Private Sub sstbPrefs_Click(PreviousTab As Integer)
   Dim ThisSetting As String
   Select Case PreviousTab
      Case 0
         If optLoanLen(0) = True Then
            ThisSetting = "Months"
         Else
            ThisSetting = "Years"
         End If
         SaveSetting("LoanSheet", "LoanLength", _
               "Period", ThisSetting)
      Case 1
         Dim X As Integer
         For X = 0 To 3
            If OptPctsShown(X) = True Then
               SaveSetting("LoanSheet", "InterestRate", _
                "Precision", OptPctsShown(X).Tag)
               Exit For
            End If
         Next X
   End Select
End Sub
```

## Picture Property (SSTab Control)

See Also Example

Returns or sets a graphic to be displayed in the current tab of an **SSTab** control.

### Syntax

*object*.**Picture** [ = *picture*]

The **Picture** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *picture* | A string expression that designates a bitmap or icon to display on the current tab, as described in Settings. |

### Settings

The settings for *picture* are:

| Setting | Description |
|---|---|
| (None) | An object expression that evaluates to an **SSTab** control. |
| (Bitmap, icon, metafile) | A string expression that designates a bitmap or icon to display on the current tab. |

### Remarks

At design time, you set the **Picture** property for a tab by clicking that tab and then setting the property in the Properties window. At run time, you can set the **Picture** property using the **LoadPicture** function or the **Picture** property of another control or of a **Form** object. You can make any tab the current tab by setting the **Tab** property.

When setting the **Picture** property at design time, the graphic is saved and loaded with the **Form** object containing the **SSTab** control. If you create an executable file, the file contains the image. When you load a graphic at run time, the graphic isn't saved with the application.

Setting the **Picture** property affects the value of the **TabPicture** property for the current tab as well as displays the picture in the active tab.

**See Also**

**SSTab** Control

**TabPicture** Property

▪

## Picture Property (SSTab Control) Example

This example loads a bitmap from a file and places that bitmap on the active tab. To try this example, put the **SSTab** and **CommandButton** controls on the **Form**. Then run the example.

```
Private Sub Command1_Click()
   SSTab1.Picture = LoadPicture("c:\windows\cars.bmp")
End Sub
```

## Rows Property (SSTab Control)

See Also

Returns the total number of rows of tabs in an **SSTab** control.

### Syntax

*object*.**Rows**

The *object* placeholder represents an object expression that evaluates to an **SSTab** control.

### Remarks

You specify the number of rows in the **SSTab** control at design time by setting the **Tabs** and **TabsPerRow** properties.

**See Also**

**SSTab** Control

**Tabs** Property (**SSTab** Control)

**TabsPerRow** Property

## ShowFocusRect Property

See Also

Returns or sets a value that determines if the focus rectangle is visible on a tab on an **SSTab** control when the tab gets the focus.

### Syntax

*object*.**ShowFocusRect** [ = *boolean* ]

The **ShowFocusRect** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *boolean* | A Boolean expression that specifies how the focus rectangle behaves, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) The control shows the focus rectangle on the tab that has the focus. |
| **False** | The control does not show the focus rectangle on the tab that has the focus. |

**See Also**

**SSTab** Control

## Style Property (SSTab Control)

See Also

Returns or sets the style of the tabs on an **SSTab** control.

### Syntax

*object*.**Style** [ = *value* ]

The **Style** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *value* | A constant or integer that specifies the style of the tabs, as described in Settings. |

### Settings

The settings for *value* are:

| Constant | Value | Description |
|---|---|---|
| **ssStyleTabbedDialog** | 0 | (Default) The tabs that appear in the tabbed dialogs look like those in Microsoft Office for Microsoft Windows 3.1 applications. If you select this style, the active tab's font is bold. |
| **ssStylePropertyPage** | 1 | The tabs that appear in the tabbed dialogs look like those in Microsoft Windows 95. When you select this setting, the **TabMaxWidth** property is ignored and the width of each tab adjusts to the length of the text in its caption. The font used to display text in the tab is not bold. |

**See Also**

**SSTab** Control

**TabMaxWidth** Property

## Tab Property (SSTab Control)

See Also Example

Returns or sets the current tab for an **SSTab** control.

### Syntax

*object*.**Tab** [ = *tabnumber* ]

The **Tab** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tabnumber* | A numeric expression that indicates a specific tab. The first tab is always 0. |

### Remarks

The current tab moves to the front and becomes the active tab.

Typically, the user of your application clicks a tab to make it the current tab. However, you may need to select the current tab in code. For example, you may want the same tab to be the current tab each time you display a certain dialog box in your application. If you dismiss the dialog box by using the **Hide** method of the **Form**, the last tab to be the active tab when the **Form** was hidden will be the active tab the next time the dialog box appears. You can set the **Tab** property of the **SSTab** control so the same tab is active every time the dialog box appears.

**See Also**

**SSTab** Control

■

## Tab Property Example

This example always makes the first tab in the **SSTab** control the active tab just before showing the form which contains the control.   To try this example, create two **Form** objects.   Place a **CommandButton** control on Form1 and an **SSTab** control on Form2.   Paste the code into the Click event of the **CommandButton** on Form1, and then run the example.

```
Private Sub Command1_Click()
   Form2.SSTab1.Tab = 1
   Form2.Show
End Sub
```

## TabCaption Property

See Also Example

Returns or sets the caption for each tab for an **SSTab** control.

### Syntax

*object*.**TabCaption**(*tab*) [ = *text* ]

The **TabCaption** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tab* | A numeric expression that specifies the tab you want the caption to appear on. |
| *text* | A string expression that evaluates to the text displayed as the caption for the specified tab. |

### Remarks

At design time, you can set the **TabCaption** property by clicking a tab and then setting the **Caption** property in the Properties window. Or you can select (Custom) in the Properties window and set the **TabCaption** property in the General tab of the Properties dialog box.

At run time, you can read or change the caption of any tab using the **TabCaption** property. You can also use the **Caption** property to change the **TabCaption** property for just the active tab.

You can use the **TabCaption** property to assign an access key to a tab. In the **TabCaption** setting, include an ampersand (&) immediately preceding the character you want to designate as an access key. The character is underlined. Press the ALT key plus the underlined character to make that tab the active tab. To include an ampersand in a caption without creating an access key, include two ampersands (&&). A single ampersand is displayed in the caption and no characters are underlined.

**See Also**

**Caption** Property
**SSTab** Control
**Tab** Property (**SSTab** Control)

▪

## TabCaption Property Example

This example adds or removes an extra word from the tabs of an **SSTab** control that lists the defensive players of a sport on one tab and the offensive players on another tab.   By clicking the **CheckBox** control on the **Form**, the user can toggle between longer captions or shorter ones.

```
Private Sub Check1_Click()
   Dim X As Integer
   For X = 0 To SSTab1.Tabs - 1
      Select Case Check1.Value
         Case 0         ' Toggle to short captions.
            SSTab1.TabCaption(X) = Left(SSTab.TabCaption(X), 7)
         Case 1         ' Toggle to long captions.
            SSTab1.TabCaption(X) = SSTab1.TabCaption(X) & " Players"
      End Select
   Next X
End Sub
```

## TabEnabled Property

See Also

Returns or sets a value that determines whether a tab in an **SSTab** control will respond to being clicked.

### Syntax

*object*.**TabEnabled**(*tab*)[ = *boolean* ]

The **TabEnabled** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tab* | A numeric expression that specifies the tab. |
| *boolean* | A Boolean expression that specifies if the tab will respond to being clicked, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) The tab responds when clicked. |
| **False** | The tab doesn't respond when clicked. |

### Remarks

When a tab is disabled, the text on the tab is grayed out and the user cannot select that tab.

The **TabEnabled** property enables or disables a single tab.   Use the **Enabled** property to enable or disable the entire **SSTab** control.

**See Also**

**SSTab** Control

**Enabled** Property

## TabHeight Property

See Also

Returns or sets the height of all tabs on an **SSTab** control.

### Syntax

*object*.**TabHeight** [ = *height* ]

The **TabHeight** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *height* | A numeric expression that specifies the height of the tab, based on the scale mode of its container**.** |

**See Also**

**SSTab** Control

## TabMaxWidth Property

See Also

Returns or sets the maximum width of each tab on an **SSTab** control.

### Syntax

*object*.**TabMaxWidth** [ = *width* ]

The **TabMaxWidth** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *width* | A numeric expression that determines the maximum width of each tab in the scale mode of its container. |

### Remarks

When the **Style** property setting is **ssStyleTabbedDialog** and the **TabMaxWidth** property is set to zero (0), the **SSTab** control automatically sizes the tabs, based on the **TabsPerRow** property, to fit evenly across the control.

If you select the **ssStylePropertyPage** setting in the **Style** property, the **TabMaxWidth** property is ignored.   The width of each tab adjusts automatically to the length of the text in the **TabCaption** property.

**See Also**

**SSTab** Control

## TabOrientation Property

See Also

Returns or sets the location of the tabs on the **SSTab** control.

### Syntax

*object*.**TabOrientation** [ = *number* ]

The **TabOrientation** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *number* | A numeric expression that specifies the location of the tabs, as described in Settings. |

### Settings

The settings for *number* are:

| Constant | Value | Description |
|---|---|---|
| **ssTabOrientationTop** | **0** | The tabs appear at the top of the control. |
| **ssTabOrientationBottom** | **1** | The tabs appear at the bottom of the control. |
| **ssTabOrientationLeft** | **2** | The tabs appear on the left side of the control. |
| **ssTabOrientationRight** | **3** | The tabs appear on the right side of the control. |

### Remarks

If you are using TrueType fonts, the text is rotated when the **TabOrientation** property is set to **ssTabOrientationLeft** or **ssTabOrientationRight**.

**See Also**

**SSTab** Control

## TabPicture Property

See Also

Returns or sets the bitmap or icon to display on the specified tab of an **SSTab** control.

### Syntax

*object*.**TabPicture**(*tab*) [ = *picture* ]

The **TabPicture** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tab* | A numeric expression that specifies the tab on which to display the picture. |
| *picture* | A string expression that specifies a graphic, as described in Settings. |

### Settings

The settings for *picture* are:

| Setting | Description |
|---|---|
| (None) | (Default) No picture. |
| (Bitmap, icon, metafile) | Specifies a graphic. At run time, you can set this property using the **LoadPicture** function or the **Picture** property of another control or **Form** object. |

### Remarks

At design time, you can set the **TabPicture** property by clicking a tab then setting the **Picture** property in the Properties window. Or you can select (Custom) in the Properties window and set the **Picture** property in the Pictures tab of the Properties dialog box.

At run time, you can refer to or change the graphic on any tab using the **TabPicture** property or use the **Picture** property to work with the active tab.

**See Also**

**Picture** Property (**SSTab** Control)
**SSTab** Control
**Tab** Property (**SSTab** Control)

## Tabs Property (SSTab Control)

See Also

Returns or sets the total number of tabs on an **SSTab** control.

### Syntax

*object*.**Tabs** [ = *tabnumber* ]

The **Tabs** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tabnumber* | A numeric expression that specifies the number of tabs you want on the control. The tabs are automatically given the captions Tab x where x is 0, 1, 2, 3, and so on. |

### Remarks

You can change the **Tabs** property at run time to add new tabs or remove tabs.

At design time, use the **Tabs** property in conjunction with the **TabsPerRow** property to determine the number of rows of tabs displayed by the control. At run time, use the **Rows** property.

**See Also**

**SSTab** Control

**TabsPerRow** Property

## TabsPerRow Property

See Also

Returns or sets the number of tabs for each row of an **SSTab** control.

### Syntax

*object*.**TabsPerRow** [ = *tabnumber* ]

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tabnumber* | A numeric expression that specifies the number of tabs you want on each row. |

### Remarks

Use this property at design time in conjunction with the **Tabs** property to determine the number of rows displayed by the control.   At run time, use the **Rows** property.

**See Also**

**SSTab** Control

**Tabs** Property (**SSTab** Control)

## TabVisible Property

See Also

Returns or sets a value indicating if a tab in an **SSTab** control is visible or hidden.   Not available at design time.

### Syntax

*object*.**TabVisible**(*tab*) [ = *boolean* ]

The **TabVisible** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tab* | A numeric expression that specifies the tab you want to be visible or hidden. |
| *boolean* | A Boolean expression that specifies if the tab is visible or hidden, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) Tab is visible. |
| **False** | Tab is hidden.   Other tabs adjust their position so there are no gaps between tabs. |

### Remarks

The **TabVisible** property hides or displays a single tab.   Use the **Visible** property to hide or display the entire **SSTab** control.

**See Also**

**SSTab** Control

**Visible** Property

## SSTab Control Constants

See Also

### StyleConstants

| Constant | Value | Description |
|---|---|---|
| **ssStyleTabbedDialog** | 0 | The tabs look like those in the tabbed dialogs in Microsoft Office for Microsoft Windows 3.1 applications. |
| **ssStylePropertyPage** | 1 | The tabs look like the tabs in Microsoft Windows 95. |

### Tab OrientationConstants

| Constant | Value | Description |
|---|---|---|
| **ssTabOrientationTop** | 0 | The tabs appear at the top of the control. |
| **ssTabOrientationBottom** | 1 | The tabs appear at the bottom of the control. |
| **ssTabOrientationLeft** | 2 | The tabs appear on the left side of the control. |
| **ssTabOrientationRight** | 3 | The tabs appear on the right side of the control. |

**See Also**

**SSTab** Control
**Style** Property (**SSTab** Control)
**TabOrientation** Property

## WordWrap Property (SSTab Control)

See Also

Returns or sets a value indicating whether the text on each tab is wrapped to the next line if it is too long to fit horizontally on the tab on an **SSTab** control.

### Syntax

*object*.**WordWrap** [ **=** *boolean* ]

The **WordWrap** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *boolean* | A Boolean expression that specifies whether the text on each tab will wrap to the next line if it does not fit horizontally, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | The text wraps if it is too long to fit within the width of each tab. |
| **False** | (Default) The text doesn't wrap and will be truncated if it is too long. |

### Remarks

Use the **WordWrap** property to determine how an **SSTab** control displays the text on each tab.   For example, a tabbed dialog that changes dynamically might have text that also changes.   To assure that text will not be truncated if it is too long, set the **WordWrap** property to **True,** the **TabMaxWidth** property to 0, and the **TabHeight** property to a height that allows you to view the longest piece of text.

**See Also**

**Caption** Property
**SSTab** Control
**TabHeight** Property
**TabMaxWidth** Property
**Tabs** (**SSTab** Control) Property

**access key**

A key pressed while holding down the ALT key that allows the user to open a menu, carry out a command, select an object, or move to an object. For example, ALT+F opens the File menu.

**ANSI Character Set**

American National Standards Institute (I) 8-bit character set used by Microsoft Windows that allows you to represent up to 256 characters (5) using your keyboard. The first 128 characters (7) correspond to the letters and symbols on a standard U.S. keyboard. The second 128 characters (5) represent special characters, such as letters in international alphabets, accents, currency symbols, and fractions.

**application**

A collection of code and visual elements that work together as a single program. Developers can build and run applications within the development environment, while users usually run applications as executable files outside the development environment.

**array**

A set of sequentially indexed elements having the same intrinsic data type. Each element of an array has a unique identifying index number. Changes made to one element of an array do not affect the other elements.

**bitmap**

An image represented by pixels and stored as a collection of bits in which each bit corresponds to one pixel. On color systems, more than one bit corresponds to each pixel. A bitmap usually has a .BMP filename extension.

**Boolean expression**

An expression that evaluates to either **True** or **False**.

**bound control**

A data-aware control that can provide access to a specific field in a database through a **Data** control. A data-aware control can be bound to a **Data** control through its **DataSource** and **DataField** properties. When a **Data** control moves from one record to the next, all bound controls connected to the **Data** control change to display data from fields in the current record. When users change data in a bound control and then move to a different record, the changes are automatically saved in the database.

**Cancel button**

A button you choose to cancel changes. This is also a generic term for any command button on a form that has the **Cancel** property set to **True**, allowing the user to press ESC or to click the button to cancel changes on that form.

**cascading event**

A sequence of events caused by an event procedure directly or indirectly calling itself; also referred to as an event cascade or recursion.   Cascading event procedures often result in run-time errors, such as stack overflow.

**Clipboard**

A temporary storage location used to transfer text, graphics, and code.

**collection**

An object that contains a set of related objects. An object's position in the collection can change whenever a change occurs in the collection; therefore, the position of any specific object in the collection may vary.

**collection syntax**

For a **Things** collection that contains **Thing** objects, the collection itself would be referred to as

*object*.**Things**

while an individual **Thing** object would be referred to as

*object*.**Things**.**Item(**i*ndex***)**

or

*object*.**Things(**i*ndex***)**

where *index* is an integer denoting a specific element in the collection.

For a complete discussion of collection syntax see Using Syntax for Collections.

**container**

An object that contains child forms or controls.

**constant**

A named item that retains a constant value throughout the execution of a program, as opposed to a variable, whose value can change during execution. Each host application can define its own set of constants. Additional constants may be defined by the user with the **Const** statement. Constants can be used anywhere in your code in place of actual values. A constant may be a string or numeric literal, another constant, or any combination that includes arithmetic or logical operators except **Is** and exponentiation. For example:

```
Const A = "MyString"
```

**control array**

A group of controls that share a common name, type, and event procedures.  Each control in the array has a unique index number that can be used to determine which control recognizes an event.

**Control Panel**

A set of programs that control your system configuration. You use Control Panel utilities to adjust hardware and software options, such as desktop colors, printer selections, date and number formats, fonts, and locale settings, such as language and system of measurement.

**current record**

The record in a **Recordset** object that you can use to modify or examine data. Use the Move methods to reposition the current record in a recordset. Use the Find methods (t) or the **Seek** method (t) to change the current record position according to specific criteria.

Only one record in a **Recordset** can be the current record; however, a **Recordset** may have no current record. For example, after a dynaset-type **Recordset** record has been deleted, or when a **Recordset** has no records, the current record is undefined. In this case, operations that refer to the current record result in a trappable error.

**custom control**

A file with an .OCX filename extension or an insertable object that, when added to a project using the Custom Controls dialog box, extends the Toolbox. The **ProgressBar** and **StatusBar** controls are examples of custom controls.

**data type**

The characteristic of a variable that determines what kind of data it can hold. Data types include **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Date**, **String**, **Object**, **Variant** (default) and user-defined types, as well as specific types of objects.

**design time**

The time during which you build an application in the development environment by adding controls, setting control or form properties, and so on. In contrast, during run time, you interact with the application as a user would.

**device context**

A link between a Windows-based application, a device driver, and an output device such as a display, printer, or plotter.

**executable file**

A Windows-based application that can run outside the development environment. An executable file has an .EXE filename extension.

**expression**

A combination of keywords, operators, variables, and constants that yield a string, number, or object. An expression can perform a calculation, manipulate characters, or test data.

**flag**

A variable you use to keep track of a condition in your application. You can set a flag using a constant or combination of constants.

**focus**

In the Microsoft Windows environment, only one window, form, or control can receive mouse clicks or keyboard input at any one time. The object that "has the focus" is normally indicated by a highlighted caption or title bar. The focus can be set by the user or by the application.

**form**

A window or dialog box. Forms are containers for controls. A multiple-document interface (MDI) form can also act as a container for child forms and some controls.

**handle**

A unique integer value defined by the operating environment and used by a program to identify and access an object, such as a form or control.

**icon**

A graphical representation of an object or concept; commonly used to represent minimized applications in Microsoft Windows. Essentially, an icon is a bitmap with a maximum size of 32 x 32 pixels. Icons have an .ICO filename extension.

**internal area**

The area in a multiple-document interface (MDI) form used to display MDI child forms.   The internal area excludes the MDI form's title bar, border, menu bar, and aligned controls on the MDI form.   Also called the client area.

**legend**

■ Applies only to object.
■ Applies only to collection.
▣ Applies to both object and collection.

**named argument**

An argument that has a name that is predefined in the object library. Instead of providing values for arguments in the order expected by the syntax, you can use named arguments to assign values in any order. For example, suppose a method accepts three arguments:

**DoSomeThing *namedarg1*, *namedarg2*, *namedarg3***

By assigning values to named arguments, you can use the following statement:

```
DoSomeThing namedarg3 := 4, namedarg2 := 5, namedarg1 := 20
```

Note that the arguments need not be in their normal positional order.

**numeric expression**

Any expression that can be evaluated as a number. Elements of the expression can include any combination of keywords, variables, constants, and operators that result in a number.

**Object Browser**

A dialog box that lets you examine the contents of an object library to get information about the objects provided.

**object expression**

An expression that specifies a particular object.   This expression can include any of the object's containers.   For example, if your application has an **Application** object that contains a **Document** object that contains a **Text** object, the following are valid object expressions:

```
Application.Document.Text
Application.Text
Document.Text
Text
```

**object library**

A file with the .OLB extension that provides information to OLE Automation controllers (like Visual Basic) about available OLE Automation objects.   You can use the Object Browser to examine the contents of an object library to get information about the objects provided.

**object variable**

A variable that contains a reference to an object.

**parent form**

A form containing controls.

**persistent graphic**

The output from a graphics method that is stored in memory. Persistent graphics are automatically retained when certain kinds of screen events occur, for example, when a form is redisplayed after being hidden behind another window. Graphics are persistent if they are drawn when the **AutoRedraw** property is set to **True**.

**Properties window**

A window used to display or change properties of a selected form or control at design time. Some custom controls have customized Properties windows.

**registry**

In Windows version 3.1, OLE registration information and file associations are stored in the registration database, and program settings are stored in Windows system initialization (.INI) files. In Windows 95, the Windows registry serves as a central configuration database for user, application, and computer-specific information, including the information previously contained in both the Windows 3.1 registration database and .INI files.

**run time**

The time when code is running. During run time, you interact with the code as a user would.

**source**

The application, form, or control that sends information and commands when two or more programs that support dynamic data exchange (DDE) are running simultaneously.

**string expression**

Any expression that evaluates to a sequence of contiguous characters. Elements of the expression can include a function that returns a string, a string literal, a string constant, a string variable, a string **Variant**, or a function that returns a string **Variant** (**VarType** 8).

**title bar**

An area at the top of a window that displays the window's caption or name.

**ToolTip**

The word or short phrase that describes the function of a toolbar button or other tool. The ToolTip appears when you pause the cursor over an object.

**Windows API**

The Windows API (Application Programming Interface) consists of the functions, messages, data structures, data types, and statements you can use in creating applications that run under Microsoft Windows. The parts of the API you use most are code elements included for calling API functions from Windows. These include procedure declarations (for the Windows functions), user-defined type definitions (for data structures passed to those functions), and constant declarations (for values passed to and returned from those functions).

**z-order**

The relative order that determines how controls overlap each other on a form.

## Windows 95 Control Constants

See Also

The following constants are recognized by the custom controls.   As a result, they can be used anywhere in your code in place of the actual values.

- **BorderStyle** Constants
- **MousePointer** Constants

Use the Object Browser to view the intrinsic constants you can use with methods and properties.   From the View menu, choose Object Browser, select the appropriate control library, and then the Constants object.   You can scroll through the constants that appear under Methods/Properties.

**Note**   Prefixes for the constants change with the specific control or group of controls.   However, the description remains the same unless indicated.

**See Also**

Visual Basic Custom Control Constants

## MousePointer Constants

See Also

| Constant | Value | Description |
|---|---|---|
| **ccDefault** | 0 | (Default) Shape determined by the object. |
| **ccArrow** | 1 | Arrow. |
| **ccCross** | 2 | Cross (cross-hair pointer). |
| **ccIbeam** | 3 | I Beam. |
| **ccIcon** | 4 | Icon (small square within a square). |
| **ccSize** | 5 | Size (four-pointed arrow pointing north, south, east, and west). |
| **ccSizeNESW** | 6 | Size NE SW (double arrow pointing northeast and southwest). |
| **ccSizeNS** | 7 | Size N S (double arrow pointing north and south). |
| **ccSizeNWSE** | 8 | Size NW, SE. |
| **ccSizeEW** | 9 | Size E W (double arrow pointing east and west). |
| **ccUpArrow** | 10 | Up Arrow. |
| **ccHourglass** | 11 | Hourglass (wait). |
| **ccNoDrop** | 12 | No Drop. |
| **ccArrowHourglass** | 13 | Arrow and hourglass. (Only available in 32-bit Visual Basic.) |
| **cc ArrowQuestion** | 14 | Arrow and question mark. (Only available in 32-bit Visual Basic.) |
| **ccSizeAll** | 15 | Size all. (Only available in 32-bit Visual Basic.) |
| **ccCustom** | 99 | Custom icon specified by the **MouseIcon** property. |

**Note** The cc prefix refers to the custom controls. Prefixes for the constants change with the specific control or group of controls. However, the description remains the same unless indicated.

**See Also**

**BorderStyle** Constants
**MouseIcon** Property
Visual Basic Custom Control Constants

## BorderStyle Constants

See Also

| Constant | Value | Description |
|---|---|---|
| **ccNone** | 0 | (Default) No border or border-related elements. |
| **ccFixedSingle** | 1 | (Default for **ListView** control) Fixed single. Can include Control-menu box, title bar, Maximize button, and Minimize button. Resizable only using Maximize and Minimize buttons. |

**Note** The cc prefix refers to the custom controls. The prefixes for the constants change with the specific control or group of controls. However, the description remains the same unless indicated.

**See Also**

**MousePointer** Constants

Visual Basic Custom Control Constants

## WhatsThisHelpID Property (Custom Controls)

See Also

Returns or sets an associated context number for an object. Used to provide context-sensitive Help for your application using the What's This popup in Windows 95 Help.

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**WhatsThisHelpID** [= *number*]

The **WhatsThisHelpID** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Spin button, StatusBar, SSTab, TabStrip, Toolbar, or TreeView control. |
| *number* | A numeric expression specifying a Help context number, as described in Settings. |

### Settings

The settings for *number* are:

| Setting | Description |
|---|---|
| 0 | (Default) No context number specified. |
| >0 | An integer specifying the valid context number for the What's This topic associated with the object. |

### Remarks

Visual Basic applications can support either of two different models for context-sensitive Help.

- Windows 3.x
- Windows 95

The Windows 3.x model uses the F1 key to start Windows Help and load the topic identified by the **HelpContextID** property. The Windows 95 model typically uses the What's This button in the upper-right corner of the window to start Windows Help and load a topic identified by the **WhatsThisHelpID** property. Use the **WhatsThisHelp** property to select between the two context-sensitive models.

**See Also**

**HelpContextID** Property
**ShowWhatsThis** Method
**WhatsThisButton** Property
**WhatsThisHelp** Property

## ShowWhatsThis Method (Custom Controls)

See Also Example

Displays a selected topic in a Help file using the What's This popup provided by Windows 95 Help.

**Important** This method requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**ShowWhatsThis**

The *object* placeholder represents an object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Spin button, StatusBar, SSTab, TabStrip, Toolbar, or TreeView control.

### Remarks

The **ShowWhatsThis** method is very useful for providing context-sensitive Help from a context menu in your application. The method displays the topic identified by the **WhatsThisHelpID** property of the object specified in the syntax.

**See Also**

**WhatsThisButton** Property
**WhatsThisHelp** Property
**WhatsThisHelpID** Property

■

## ShowWhatsThis Method (Custom Controls) Example

This example displays the What's This Help topic for a **CommandButton** control by selecting a menu command from a context menu created for the button.

```
Private ThisControl As Control

Private Sub Command1_MouseUp(Button As Integer, Shift As Integer, _
   X As Single, Y As Single)
      If Button = vbRightButton Then
         Set ThisControl = Command1
         PopupMenu mnuBtnContextMenu
      End If
   Set ThisControl = Nothing
End Sub

Private Sub mnuBtnWhatsThis_Click()
   ThisControl.ShowWhatsThis
End Sub
```

## WhatsThisButton Property (Custom Controls)

See Also

Returns or sets a value that determines whether the What's This button appears in the title bar of a **Form** object. Read only at run time.

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**WhatsThisButton**

The *object* placeholder represents an object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Spin button, StatusBar, SSTab, TabStrip, Toolbar, or TreeView control.

### Settings

The settings for the **WhatsThisButton** property are:

| Setting | Description |
|---|---|
| **True** | Turns display of the What's This Help button on. |
| **False** | (Default) Turns display of the What's This Help button off. |

### Remarks

The **WhatsThisHelp** property must be **True** for the **WhatsThisButton** property to be **True**. In addition, the **BorderStyle** property must be set to **ccFixedSingle**.

**See Also**

**BorderStyle** Property
**ShowWhatsThis** Method
**WhatsThisHelp** Property
**WhatsThisHelpID** Property

## WhatsThisHelp Property (Custom Controls)

See Also

Returns or sets a value that determines whether context-sensitive Help uses the What's This pop-up provided by Windows 95 Help or the main Help window. Read-only at run time.

**Important** This property requires the Microsoft Windows 95 or Microsoft Windows NT 3.51 operating system.

### Syntax

*object*.**WhatsThisHelp**

The *object* placeholder represents an object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Spin button, StatusBar, SSTab, TabStrip, Toolbar, or TreeView control.

### Settings

The settings for the **WhatsThisHelpID** property are:

| Setting | Description |
|---|---|
| **True** | The application uses one of the What's This access techniques to start Windows Help and load a topic identified by the **WhatsThisHelpID** property. |
| **False** | (Default) The application uses the F1 key to start Windows Help and load the topic identified by the **HelpContextID** property. |

### Remarks

There are three access techniques for providing What's This Help in an application. The **WhatsThisHelp** property must be set to **True** for any of these techniques to work.

- Providing a What's This button in the title bar of the form using the **WhatsThisButton** property. The mouse pointer changes into the What's This state (arrow with question mark). The topic displayed is identified by the **WhatsThisHelpID** property of the control clicked by the user.
- Invoking the **WhatsThisMode** method of a form. This produces the same behavior as clicking the What's This button without using a button. For example, you can invoke this method from a command on a menu in the menu bar of your application.
- Invoking the **ShowWhatsThis** method for a particular control. The topic displayed is identified by the **WhatsThisHelpID** property of the control.

**See Also**

**ShowWhatsThis** Method
**WhatsThisButton** Property
**WhatsThisHelpID** Property

## Add

The **Add** keyword is used in these contexts:

**Add** Method (**Buttons** Collection)
**Add** Method (**ColumnHeaders** Collection)
**Add** Method (**ListImages** Collection)
**Add** Method (**ListItems** Collection)
**Add** Method (**Nodes** Collection)
**Add** Method (**Panels** Collection)
**Add** Method (**Tabs** Collection)

## Index

The **Index** keyword is used in these contexts:
**Index** Property (Control Array)
**Index** Property (Custom Controls)

## Sorted

The **Sorted** keyword is used in these contexts:

**Sorted** Property (**ListView** Control)

**Sorted** Property (**TreeView** Control)

## Style

The **Style** keyword is used in these contexts:
**Style** Property (**Button** Object)
**Style** Property (**Panel** Object)
**Style** Property (**StatusBar** Control)
**Style** Property (**TabStrip** Control)
**Style** Property (**TreeView** Control)

## Width

The **Width** keyword is used in these contexts:
**Width** Property (**Panel** Object)
**Height**, **Width** Properties (Custom Controls)

## Alignment

The **Alignment** keyword is used in these contexts:
**Alignment** Property (**ColumnHeader** Object)
**Alignment** Property (**Panel** Object)

## AutoSize

The **AutoSize** keyword is used in these contexts:
**Autosize** Property (Custom Controls)
**Autosize** Property (**Panel** Object)

## Caption

The **Caption** keyword is used in these contexts:

**Caption** Property (Custom Controls)

**Caption** Property (**Tab** Object)

## Change

The **Change** keyword is used in these contexts:
Change Event (Custom Controls)
Change Event (**Toolbar** Control)

## ImageList

The **ImageList** keyword is used in these contexts:
**ImageList** Control
**ImageList** Property (Custom Controls)

## Parent

The **Parent** keyword is used in these contexts:

**Parent** Property (Custom Controls)

**Parent** Property (**Node** Object)

## Change Event (Custom Controls)

See Also

Indicates that the contents of a control have changed. How and when this event occurs varies with the control.

### Syntax

**Private Sub** *object*_**Change(**[*index* **As Integer**]**)**

The Change event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a Masked edit, Gauge, RichTextBox, Slider, or Toolbar control. |
| *index* | An integer that uniquely identifies a control if it's in a control array. |

### Remarks

- Masked edit and RichTextBox
- changes the contents of the text box. Occurs when a DDE link updates data, when a user changes the text, or when you change the **Text** property setting through code.
- Slider
- generated when the **Value** property changes, either through code, or when the user moves the control's slider.
- Toolbar
- generated after the end user customizes a Toolbar control's toolbar using the Customize Toolbar dialog box.

The Change event procedure can synchronize or coordinate data display among controls. For example, you can use a Slider control's Change event procedure to update the control's **Value** property setting in a TextBox control. Or you could use a Change event procedure to display data and formulas in a work area and results in another area.

**Note** A Change event procedure can sometimes cause a cascading event. This occurs when the control's Change event alters the control's contents by setting a property in code that determines the control's value, such as the **Text** property setting for a TextBox control. To prevent a cascading event:

- If possible, avoid writing a Change event procedure for a control that alters that control's contents. If you do write such a procedure, be sure to set a flag that prevents further changes while the current change is in progress.
- Avoid creating two or more controls whose Change event procedures affect each other, for example, two TextBox controls that update each other during their Change events.

**See Also**

**Text** Property (Masked Edit Control)
**Text** Property
**Value** Property

## Click Event (Custom Controls)

See Also

Occurs when the user presses and then releases a mouse button over an object. It can also occur when the value of a control is changed.

### Syntax

**Private Sub** *object*_**Click(**[*index* **As Integer**]**)**

The Click event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D command button, 3D frame, Gauge, Graph, Key state, ListView**,** Outline, ProgressBar, RichTextBox, Slider, StatusBar, TabStrip, Toolbar, or TreeView control. |
| *index* | An integer that uniquely identifies a control if it's in a control array. |

### Remarks

For a control, this event occurs when the user:

- Clicks a control with the left or right mouse button. With a 3D check box, 3D command button, or 3D option button control, the Click event occurs only when the user clicks the left mouse button.
- Presses the SPACEBAR when a 3D command button, 3D option button, or 3D check box control has the focus.
- Presses ENTER when a form has a 3D command button control with its **Default** property set to **True**.
- Presses ESC when a form has a Cancel button
- for example, a 3D command button control with its **Cancel** property set to **True**.
- Presses an access key for a control. For example, if the caption of a 3D command button control is "&Go", pressing ALT+G triggers the event.

You can also trigger the Click event in code by:

- Setting a 3D command button control's **Value** property to **True**.
- Setting a 3D option button control's **Value** property to **True**.
- Changing a 3D check box control's **Value** property setting.

Typically, you attach a Click event procedure to a 3D command button control to carry out commands and command-like actions. For the other applicable controls, use this event to trigger actions in response to a change in the control.

You can use a control's **Value** property to test the state of the control from code.

**Note** To distinguish between the left, right, and middle mouse buttons, use the MouseDown and MouseUp events.

**See Also**

**Cancel** Property (3D Command Button Control)
**Default** Property (3D Command Button Control)
MouseDown, MouseUp Events
**Value** Property (3D Controls)
**Value** Property, Gauge Control
**Value** Property, Key State Control

## DragDrop Event (Custom Controls)

See Also

Occurs when a drag-and-drop operation is completed as a result of dragging a control over a form or control and releasing the mouse button or using the **Drag** method with its *action* argument set to 2 (Drop).

### Syntax

**Private Sub** *object*_**DragDrop(**[*index* **As Integer**,]*source* **As Control**, *x* **As Single**, *y* **As Single)**

The DragDrop event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. |
| *index* | An integer that uniquely identifies a control if it's in a control array. |
| *source* | The control being dragged. You can include properties and methods with this argument■ for example, `Source.Visible = 0`. |

*x*, *y* A number that specifies the current horizontal (*x*) and vertical (*y*) position of the mouse pointer within the target form or control. These coordinates are always expressed in terms of the target's coordinate system as set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties.

### Remarks

Use a DragDrop event procedure to control what happens after a drag operation is completed. For example, you can move the source control to a new location or copy a file from one location to another.

When multiple controls can potentially be used in a *source* argument:

- Use the **TypeOf** keyword with the **If** statement to determine the type of control used with *source.*
- Use the control's **Tag** property to identify a control, and then use a DragDrop event procedure.

**Note** Use the **DragMode** property and **Drag** method to specify the way dragging is initiated. Once dragging has been initiated, you can handle events that precede a DragDrop event with a DragOver event procedure.

**See Also**

**Drag** Method
**DragMode** Property
DragOver Event
**Tag** Property

## DragOver Event (Custom Controls)

See Also

Occurs when a drag-and-drop operation is in progress. You can use this event to monitor the mouse pointer as it enters, leaves, or rests directly over a valid target. The mouse pointer position determines the target object that receives this event.

### Syntax

**Private Sub** *object*_**DragOver(**[*index* **As Integer**,]*source* **As Control**, *x* **As Single**, *y* **As Single**, *state* **As Integer)**

The DragOver event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. |
| *index* | An integer that uniquely identifies a control if it's in a control array. |
| *source* | The control being dragged. You can refer to properties and methods with this argument■ for example, `Source.Visible = False`. |

*x*, *y* A number that specifies the current horizontal (*x*) and vertical (*y*) position of the mouse pointer within the target form or control. These coordinates are always expressed in terms of the target's coordinate system as set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties.
*state* An integer that corresponds to the transition state of the control being dragged in relation to a target form or control, as described in Settings.

### Settings

The settings for *state* are:

| Setting | Description |
|---|---|
| 0 | Enter. The source control is being dragged within the range of a target. |
| 1 | Leave. The source control is being dragged out of the range of a target. |
| 2 | Over. The source control has moved from one position in the target to another. |

### Remarks

Use a DragOver event procedure to determine what happens after dragging is initiated and before a control drops onto a target. For example, you can verify a valid target range by highlighting the target (set the **BackColor** or **ForeColor** property from code) or by displaying a special drag pointer (set the **DragIcon** or **MousePointer** property from code).

Use the *state* argument to determine actions at key transition points. For example, you might highlight a possible target when *state* is set to 0 (Enter) and restore the object's previous appearance when *state* is set to 1 (Leave).

When *state* is set to 0 (Enter) and an object receives a DragOver event :

- If the source control is dropped on the object, that object receives a DragDrop event.
- If the source control isn't dropped on the object, that object receives another DragOver event when *state* is set to 1 (Leave).

**Note** Use the **DragMode** property and **Drag** method to specify the way dragging is initiated. For suggested techniques with the *source* argument, see Remarks for the DragDrop event topic.

**See Also**

**BackColor**, **ForeColor** Properties
**Drag** Method
DragDrop Event
**DragIcon** Property
**DragMode** Property
**MousePointer** Property

## GotFocus Event (Custom Controls)

See Also

Occurs when an object receives the focus, either by user action, such as tabbing to or clicking the object, or by changing the focus in code using the **SetFocus** method.

### Syntax

**Private Sub** *object*_**GotFocus(**[*index* **As Integer**]**)**

The GotFocus event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D option button, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Outline, RichTextBox, Slider, SSTab, TabStrip, or TreeView control. |
| *index* | An integer that uniquely identifies a control if it's in a control array. |

### Remarks

Typically, you use a GotFocus event procedure to specify the actions that occur when a control first receives the focus.   For example, by attaching a GotFocus event procedure to each control on a form, you can guide the user by displaying brief instructions or status bar messages.   You can also provide visual cues by enabling, disabling, or showing other controls that depend on the control that has the focus.

**Note**   To customize the keyboard interface in Visual Basic for moving the focus, set the tab order or specify access keys for controls on a form.

**See Also**

**SetFocus** Method

## KeyDown, KeyUp Events (Custom Controls)

See Also

Occur when the user presses (KeyDown) or releases (KeyUp) a key while an object has the focus.

**Note** To interpret ANSI characters, use the KeyPress event.

### Syntax

**Private Sub** *object*_**KeyDown(**[*index* **As Integer**,]*keycode* **As Integer**, *shift* **As Integer)**

**Private Sub** *object*_**KeyUp(**[*index* **As Integer**,]*keycode* **As Integer**, *shift* **As Integer)**

The KeyDown and KeyUp event syntaxes have these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D option button, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Outline, RichTextBox, Slider, SSTab, TabStrip, or TreeView control. |
| *index* | An integer that uniquely identifies a control if it's in a control array. |
| *keycode* | A key code, such as **vbKeyF1** (the F1 key) or **vbKeyHome** (the HOME key). To specify key codes, use the constants in the object library in the Object Browser. |
| *shift* | An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2 ). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6. |

### Remarks

For both events, the object with the focus receives all keystrokes. Although the KeyDown and KeyUp events can apply to most keys, they're most often used for:

- Extended character keys such as function keys.
- Navigation keys.
- Combinations of keys with standard keyboard modifiers.
- Distinguishing between the numeric keypad and regular number keys.

Use KeyDown and KeyUp event procedures if you need to respond to both the pressing and releasing of a key.

KeyDown and KeyUp aren't invoked for:

- The ENTER key if the form has a 3D command button control with the **Default** property set to **True**.
- The ESC key if the form has a 3D command button control with the **Cancel** property set to **True**.
- The TAB key.

KeyDown and KeyUp interpret the uppercase and lowercase of each character by means of two arguments: *keycode*, which indicates the physical key (thus returning A and a as the same key) and *shift*, which indicates the state of *shift*+*key* and therefore returns either A or a.

If you need to test for the *shift* argument, you can declare constants that define the bits within the argument by using constants listed in the object library in the Object Browser. The *shift* constants have the following values:

| Constant | Value |
|---|---|
| **vbShiftMask** | 1 |
| **vbCtrlMask** | 2 |
| **vbAltMask** | 4 |

The constants act as bit masks that you can use to test for any combination of keys. Place the constants at the procedure level or in the Declarations section of a module and use this syntax:

**Const** *constantname* **=** *expression*

You test for a condition by first assigning each result to a temporary integer variable and then comparing *shift* to a bit mask.   Use the **And** operator with the *shift* argument to test whether the condition is greater than 0, indicating that the modifier was pressed, for example:

```
ShiftDown = (Shift And vbShiftMask) > 0
```

In a procedure, you can test for any combination of conditions, for example:

```
If ShiftDown And CtrlDown Then
```

---

**Note**     If the **KeyPreview** property is set to **True**, a form receives these events before controls on the form receive the events.   Use the **KeyPreview** property to create global keyboard-handling routines.

---

**See Also**

**Cancel** Property (3D Command Button)

**Default** Property (3D Command Button)

## KeyPress Event (Custom Controls)

See Also

Occurs when the user presses and releases an ANSI key.

### Syntax

**Private Sub** *object*_**KeyPress(**[*index* **As Integer**,]*keyascii* **As Integer)**

The KeyPress event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D option button, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Outline, RichTextBox, Slider, SSTab, TabStrip or TreeView control. |
| *index* | An integer that uniquely identifies a control if it's in a control array. |
| *keyascii* | An integer that returns a standard numeric ANSI keycode. The *keyascii* argument is passed by reference; changing it sends a different character to the object. Changing *keyascii* to 0 cancels the keystroke so the object receives no character. |

### Remarks

The object with the focus receives the event. A form can receive the event only if it has no visible and enabled controls or if the **KeyPreview** property is set to **True**. A KeyPress event can involve any printable keyboard character, the CTRL key combined with a character from the standard alphabet or one of a few special characters, and the ENTER or BACKSPACE key. A KeyPress event procedure is useful for intercepting keystrokes entered in a **RichTextBox** control. It enables you to immediately test keystrokes for validity or to format characters as they're typed. Changing the value of the *keyascii* argument changes the character displayed.

You can convert the *keyascii* argument into a character by using the expression:

```
Chr(KeyAscii)
```

You can then perform string operations and translate the character back to an ANSI number that the control can interpret by using the expression:

```
KeyAscii = Asc(char)
```

Use KeyDown and KeyUp event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress doesn't indicate the physical state of the keyboard; instead, it passes a character.

KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters. KeyDown and KeyUp interpret the uppercase and lowercase of each character by means of two arguments: *keycode*, which indicates the physical key (thus returning A and a as the same key), and *shift*, which indicates the state of *shift+key* and therefore returns either A or a.

If the **KeyPreview** property is set to **True**, a form receives the event before controls on the form receive the event. Use the **KeyPreview** property to create global keyboard-handling routines.

**Note** The ANSI number for the keyboard combination of CTRL+@ is 0. Because Visual Basic recognizes a *keyascii* value of 0 as a zero-length string (""), avoid using CTRL+@ in your applications.

**See Also**

KeyDown, KeyUp Events

## LostFocus Event (Custom Controls)

See Also

Occurs when an object loses the focus, either by user action, such as tabbing to or clicking another object, or by changing the focus in code using the **SetFocus** method.

### Syntax

**Private Sub** *object*_**LostFocus(**[*index* **As Integer**]**)**

The LostFocus event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D option button, Animated button, Gauge, Graph, Key state, ListView, Masked Edit, Outline, RichTextBox, Slider, SSTab, TabStrip, or TreeView control. |
| *index* | An integer that uniquely identifies a control if it's in a control array. |

### Remarks

A LostFocus event procedure is primarily useful for verification and validation updates.   Using LostFocus can cause validation to take place as the user moves the focus from the control.   Another use for this type of event procedure is enabling, disabling, hiding, and displaying other objects as in a GotFocus event procedure.   You can also reverse or change conditions that you set up in the object's GotFocus event procedure.

**See Also**

GotFocus Event

**SetFocus** Method

## MouseDown, MouseUp Events (Custom Controls)

See Also

Occur when the user presses (MouseDown) or releases (MouseUp) a mouse button.

**Syntax**

**Private Sub** *object*_**MouseDown(**[*index* **As Integer**,]*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

**Private Sub** *object* _**MouseUp(**[*index* **As Integer**,]*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

The MouseDown and MouseUp event syntaxes have these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D panel, Gauge, Graph, ListView, Outline, ProgressBar, RichTextBox Slider, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. |
| *index* | Returns an integer that uniquely identifies a control if it's in a control array. |
| *button* | Returns an integer that identifies the button that was pressed (MouseDown) or released (MouseUp) to cause the event. The *button* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event. |
| *shift* | Returns an integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the *button* argument is pressed or released. A bit is set if the key is down. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2 ). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of *shift* would be 6. |
| *x*, *y* | Returns a number that specifies the current location of the mouse pointer. The *x* and *y* values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object. |

**Remarks**

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a given mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

The following applies to both Click and DblClick events:

- If a mouse button is pressed while the pointer is over a control, that object "captures" the mouse and receives all mouse events up to and including the last MouseUp event. This implies that the *x*, *y* mouse-pointer coordinates returned by a mouse event may not always be in the client area of the object that receives them.
- If mouse buttons are pressed in succession, the object that captures the mouse after the first press receives all mouse events until all buttons are released.

If you need to test for the *button* or *shift* arguments, you can use constants listed in the object library in the Object Browser to define the bits within the argument:

| Constant | Value | Description |
|---|---|---|
| **vbLeftButton** | 1 | Left button is pressed. |
| **vbRightButton** | 2 | Right button is pressed. |
| **vbMiddleButton** | 4 | Middle button is pressed. |
| **vbShiftMask** | 1 | SHIFT key is pressed. |
| **vbCtrlMask** | 2 | CTRL key is pressed. |
| **vbAltMask** | 4 | ALT key is pressed. |

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

---

**Note** You can use a MouseMove event procedure to respond to an event caused by moving the mouse. The *button* argument for MouseDown and MouseUp differs from the *button* argument used for MouseMove. For MouseDown and MouseUp, the *button* argument indicates exactly one button per event; for MouseMove, it indicates the current state of all buttons.

---

**See Also**

Click Event
Click Event, 3D Controls
DblClick Event
MouseMove Event

## MouseMove Event (Custom Controls)

See Also

Occurs when the user moves the mouse.

### Syntax

**Private Sub** *object*_**MouseMove(**[*index* **As Integer**,] *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

The MouseMove event syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D panel, Gauge, Graph, ListView, Outline, ProgressBar, RichTextBox, Slider, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. |
| *index* | An integer that uniquely identifies a control if it's in a control array. |
| *button* | An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down. The *button* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. It indicates the complete state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are pressed. |
| *shift* | An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys. A bit is set if the key is down. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2 ). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of *shift* would be 6. |
| *x*, *y* | A number that specifies the current location of the mouse pointer. The *x* and *y* values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object. |

### Remarks

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders.

If you need to test for the *button* or *shift* arguments, you can use constants listed in the object library in the Object Browser to define the bits within the argument:

| Constant | Value | Description |
|---|---|---|
| **vbLeftButton** | 1 | Left button is pressed. |
| **vbRightButton** | 2 | Right button is pressed. |
| **vbMiddleButton** | 4 | Middle button is pressed. |
| **vbShiftMask** | 1 | SHIFT key is pressed. |
| **vbCtrlMask** | 2 | CTRL key is pressed. |
| **vbAltMask** | 4 | ALT key is pressed. |

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

You test for a condition by first assigning each result to a temporary integer variable and then comparing the *button* or *shift* arguments to a bit mask. Use the **And** operator with each argument to test if the condition is greater than zero, indicating the key or button is pressed, as in the following code:

```
LeftDown = (Button And vbLeftButton) > 0
CtrlDown = (Shift And vbCtrlMask) > 0
```

Then, in a procedure, you can test for any combination of conditions, as follows:

```
If LeftDown And CtrlDown Then
```

**Note** You can use MouseDown and MouseUp event procedures to respond to events caused by pressing and releasing mouse buttons.

The *button* argument for MouseMove differs from the *button* argument for MouseDown and MouseUp. For MouseMove, the *button* argument indicates the current state of all buttons; a single MouseMove event can indicate that some, all, or no buttons are pressed. For MouseDown and MouseUp, the *button* argument indicates exactly one button per event.

Any time you move a window inside a MouseMove event, it can cause a cascading event. MouseMove events are generated when the window moves underneath the pointer. A MouseMove event can be generated even if the mouse is perfectly stationary.

**See Also**

MouseDown, MouseUp Events

## DblClick Event (Custom Controls)

See Also

Occurs when the user presses and releases a mouse button and then presses and releases it again over an object.

For a control, it occurs when the user double-clicks a control with the left mouse button.

### Syntax

**Private Sub** *object*_**DblClick (***index* **As Integer)**

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D frame, 3D panel, Gauge, Graph, Outline, ListView, RichTextBox, SSTab, StatusBar, TabStrip, Toolbar or TreeView control. |
| *index* | Identifies the control if it's in a control array. |

### Remarks

The argument *Index* uniquely identifies a control if it's in a control array. You can use a DblClick event procedure for an implied action, such as double-clicking an icon to open a window or document. You can also use this type of procedure to carry out multiple steps with a single action, such as double-clicking to select an item in a list box and to close the dialog box.

For those objects that receive Mouse events, the events occur in this order: MouseDown, Click, MouseUp, and DblClick.

If DblClick doesn't occur within the system's double-click time limit, the object recognizes another Click event. The double-click time limit may vary because the user can set the double-click speed in the Control Panel. When you're attaching procedures for these related events, be sure that their actions don't conflict. Controls that don't receive DblClick events may receive two clicks instead of a DblClick.

**Note** To distinguish between the left, right, and middle mouse buttons, use the MouseDown and MouseUp events.

**See Also**

Click Event

MouseDown, MouseUp Events

## Refresh Method (Custom Controls)

Forces a complete repaint of a control.

### Syntax

*object*.**Refresh**

The *object* placeholder represents an object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, RichTextBox, Slider, Spin button, StatusBar, TabStrip, Toolbar or TreeView control.

### Remarks

For example, you can use the **Refresh** method to:

- Update the contents of a file-system list box, such as a **TreeView** control.
- Update the data structures of a **ListView** control.

Generally, painting a control is handled automatically while no events are occurring. However, there may be situations where you want the control updated immediately. For example, if you use a file list box, a directory list box, or a drive list box to show the current status of the directory structure, you can use **Refresh** to update the list whenever a change is made to the directory structure.

## SetFocus Method (Custom Controls)

See Also

Moves the focus to the specified control.

### Syntax

*object*.**SetFocus**

The *object* placeholder represents an object expression that evaluates to a 3D check box, 3D command button, 3D option button, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, RichTextBox, Slider, SSTab, TabStrip or TreeView control.

### Remarks

The object must be a control that can receive the focus.   After invoking the **SetFocus** method, any user input is directed to the specified control.

You can only move the focus to a visible control.   Because controls on a form aren't visible until the form's Load event has finished, you can't use the **SetFocus** method to move the focus to the form being loaded in its own Load event unless you first use the **Show** method to show the form before the Form_Load event procedure is finished.

You also can't move the focus to a control if the **Enabled** property is set to **False**.   If the **Enabled** property has been set to **False** at design time, you must first set it to **True** before it can receive the focus using the **SetFocus** method.

**See Also**

**Enabled** Property

## Trappable Errors for Windows 95 Custom Controls

See Also

The following table lists the trappable errors for the Windows 95 custom controls.

| Error Number | Message Explanation |
|---|---|
| 35600 | Index out of bounds. |
| 35601 | Element not found. |
| 35602 | Key is not unique in collection. |
| 35603 | Invalid key. |
| 35605 | This item's control has been deleted. |
| 35606 | Control's collection has been modified. |
| 35607 | Required argument is missing. |
| 35609 | Property not accessible at design time. |
| 35610 | Invalid object. |
| 35611 | Property is read-only if image list contains images. |
| 35613 | ImageList must be initialized before it can be used. |
| 35614 | This would introduce a cycle. |
| 35615 | All images in list must be same size. |
| 35616 | Maximum Panels Exceeded. |
| 35617 | Image cannot be removed while another control is bound to this ImageList. |
| 35618 | Overlay parameter must identify one of the first 16 images in the ImageList. |

**See Also**

**ImageList** Control
**ListView** Control
**ProgressBar** Control
**RichTextBox** Control
**Slider** Control
**SSTab** Control
**StatusBar** Control
**TabStrip** Control
**Toolbar** Control
**TreeView** Control

## Clear Method (Outline Control)

Clears the contents of the **Outline** control.

**Syntax**

*object***.Clear**

## Drag Method (Custom Controls)

See Also

Begins, ends, or cancels a drag operation on any object except the **ImageList** control. Doesn't support named arguments.

### Syntax

*object*.**Drag** *action*

The **Drag** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. |
| *action* | Optional. A constant or value that specifies the action to perform, as described in Settings. If *action* is omitted, the default is to begin dragging the object. |

### Settings

The settings for *action* are:

| Constant | Value | Description |
|---|---|---|
| **vbCancel** | 0 | Cancel drag operation. |
| **vbBeginDrag** | 1 | Begin dragging *object*. |
| **vbEndDrag** | 2 | End dragging and drop *object*. |

### Remarks

These constants are listed in the object library in the Object Browser.

Using the **Drag** method to control a drag-and-drop operation is required only when the **DragMode** property of the object is set to Manual (0). However, you can use **Drag** on an object whose **DragMode** property is set to Automatic (1 or **vbAutomatic**).

If you want the mouse pointer to change shape while the object is being dragged, use either the **DragIcon** or **MousePointer** property. The **MousePointer** property is only used if no **DragIcon** is specified.

**See Also**

**DragIcon** Property
**DragMode** Property
**MousePointer** Property

## Move Method (Custom Controls)

See Also

Moves a control. Doesn't support named arguments.

### Syntax

*object***.Move** *left*, *top*, *width*, *height*

The **Move** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Optional. An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, Slider, Spin button, RichTextBox, TabStrip, or Toolbar control. If *object* is omitted, the form with the focus is assumed to be *object*. |
| *left* | Required. Single-precision value indicating the horizontal coordinate (x-axis) for the left edge of *object*. |
| *top* | Optional. Single-precision value indicating the vertical coordinate (y-axis) for the top edge of *object*. |
| *width* | Optional. Single-precision value indicating the new width of *object*. |
| *height* | Optional. Single-precision value indicating the new height of *object*. |

### Remarks

Only the *left* argument is required. However, to specify any other arguments, you must specify all arguments that appear in the syntax before the argument you want to specify. For example, you can't specify *width* without specifying *left* and *top*. Any trailing arguments that are unspecified remain unchanged.

The 3D panel, ProgressBar, StatusBar, and ToolBar controls can be moved only when their **Align** properties are set to 0 (None).

**See Also**

**Align** Property

## ZOrder Method (Custom Controls)

Places a specified control at the front or back of the z-order within its graphical level. Doesn't support named arguments.

### Syntax

*object*.**ZOrder** *position*

The **ZOrder** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | Optional. An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, Slider, Spin button, RichTextBox, SSTab, TabStrip, Toolbar, or TreeView control. If *object* is omitted, the form with the focus is assumed to be *object*. |
| *position* | Optional. Integer indicating the position of *object* relative to other instances of the same *object*. If position is 0 or omitted, *object* is positioned at the front of the z-order. If position is 1, *object* is positioned at the back of the z-order. |

### Remarks

The z-order of objects can be set at design time by choosing the Bring To Front or Send To Back menu command from the Edit menu.

## Item Method (Custom Controls)

See Also Example

Returns a specific item of a collection object by position, index, or key.

**Syntax**

*object*.**Item(***index***)**

The **Item** method syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to one of the following collections: **Buttons**, **ColumnHeaders**, **ListImages**, **ListItems**, **Nodes**, **Panels**, **Tabs**. |
| *index* | An integer or unique string that specifies a member of the collection. The integer must be a number from 1 to the value of the collection's **Count** property. The string must correspond to the key specified when the member was added to the collection. |

**Remarks**

If the value provided as *index* does not match any existing member of the collection, an error occurs.

The **Item** method is the default method for a collection. Therefore, the following lines of code are equivalent:

```
Print MyCollection.Item(1)
Print MyCollection(1)
```

**See Also**

**Count** Property

**Key** Property

■

## Item Method Example

This example adds several panels to a **StatusBar** control and uses the **Item** method to access the **Index** and **Enabled** properties of the control. To try the example, place a **StatusBar** control on a form and paste the code into the form's Declarations section. Run the example and click on the form.

```
Private Sub Form_Load()
   Dim sbrX As Panel ' Create a Panel object variable.
   Dim I As Integer  ' Create an integer variable as counter.

   For I = 1 to 6                    ' Create 6 Panel objects.
      Set sbrX = StatusBar1.Panels.Add() ' Create a Panel.
      sbrX.Style = I                 ' Use I to set the style of the panel.
   Next I
   StatusBar1.Panels.Remove 1        ' Remove first, default Panel.
End Sub

Private Sub Form_Click()
   Dim I As Integer    ' Counter variable.
   Dim strX As String  ' String variable to contain a message.
   For I = 1 to Statusbar1.Panels.Count
      strX = strX & StatusBar1.Panels.Item(I).Index & ": "
      ' The previous line is equivalent to this:
      ' strX = StrX & StatusBar1.Panels(I).Index & ": "

      strX = strX & StatusBar1.Panels.Item(I).Enabled & Chr(10)
      ' The previous line is equivalent to this:
      ' strX = StrX & StatusBar1.Panels(I).Enabled & Chr(10)

      StatusBar1.Panels.Item.(I).Width = Form1.Width / 6
      ' The previous line is equivalent to this:
      ' Statusbar1.Panels(I).Width = Form1.Width / 6
   Next I
   ' Display the result.
   MsgBox strX
End Sub
```

## BackColor, ForeColor Properties (Custom Controls)

- **BackColor**

■returns or sets the background color of an object.

- **ForeColor**

■returns or sets the foreground color used to display text and graphics in an object.

### Syntax

*object*.**BackColor** [= *color*]

*object*.**ForeColor** [= *color*]

The **BackColor** and **ForeColor** property syntaxes have these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to one of the following controls: 3D panel, Animated button, ListView, Masked edit, Outline, SSTab, or Spin button control.<br>**BackColor** property only: 3D group push button, ImageList, Key state, or RichTextBox control.<br>**ForeColor** property only: 3D check box, 3D command button, 3D frame, 3D option button, or Multimedia MCI control. |
| *color* | A value or constant that determines the background or foreground colors of an object, as described in Settings. |

### Settings

Visual Basic uses the Microsoft Windows operating environment red-green-blue (RGB) color scheme. The settings for *color* are:

| Setting | Description |
|---|---|
| Normal RGB colors | Colors specified by using the Color palette or by using the **RGB** or **QBColor** functions in code. |
| System default colors | Colors specified by system color constants listed in the object library in the Object Browser. The Windows operating environment substitutes the user's choices as specified in the Control Panel settings. |

At design time, the default settings are:

- **BackColor**

■the system default color specified by the constant **vbWindowBackground**.

- **ForeColor**

■the system default color specified by the constant **vbWindowText**.

### Remarks

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte isn't 0, Visual Basic uses the system colors, as defined in the user's Control Panel settings and by constants listed in the object library in the Object Browser.

To display text in the Windows operating environment, both the text and background colors must be solid. If the text or background colors you've selected aren't displayed, one of the selected colors may be dithered■that is, comprised of up to three different-colored pixels. If you choose a dithered color for either the text or background, the nearest solid color will be substituted.

For the **ImageList** control, before drawing a masked image on a solid-color background, you should use the **BackColor** property to set the background color of the **ImageList** to the same color as the destination. This eliminates the need to create transparent areas in the image and enables images to be displayed simply by retrieving the image with the **Item** method, resulting in a significant increase in performance.

## BorderStyle Property (Custom Contols)

Returns or sets the border style for an object.

### Syntax

*object*.**BorderStyle** [= *value*]

The **BorderStyle** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an Animated button, Graph, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Toolbar, or TreeView control. |
| *value* | A value or constant that determines the border style, as described in Settings. |

### Settings

The settings for *value* are:

| Constant | Value | Description |
|---|---|---|
| **ccNone** | 0 | (Default) No border or border-related elements. |
| **ccFixedSingle** | 1 | Fixed single.   Except for the **ProgressBar** control, can include Control-menu box, title bar, Maximize button, and Minimize button.   Resizable only using Maximize and Minimize buttons. |

**Note**   The cc prefix refers to the Windows 95 controls: **ListView**, **ProgressBar**, **RichTextBox**, **Slider**, **Toolbar**, and **TreeView**.   For the other controls, prefixes for the settings change with the specific control or group of controls.   However, the description remains the same unless indicated.

### Remarks

Setting **BorderStyle** for a **ProgressBar** control decreases the size of the chunks the control displays.

## FontBold, FontItalic, FontStrikethru, FontUnderline Properties (Custom Controls)

See Also

Return or set font styles in the following formats: **Bold**, *Italic*, ~~Strikethru~~, and Underline.

**Note** The **FontBold, FontItalic, FontStrikethru,** and **FontUnderline** properties are included for compatibility with earlier versions of Visual Basic. For additional functionality, use the new **Font** object properties.

### Syntax

*object*.**FontBold** [= *boolean*]

*object*.**FontItalic** [= *boolean*]

*object*.**FontStrikethru** [= *boolean*]

*object*.**FontUnderline** [= *boolean*]

The **FontBold**, **FontItalic**, **FontStrikethru**, and **FontUnderline** properties syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D option button, 3D panel, Animated button, Masked edit, or Outline control. |
| *boolean* | A Boolean expression specifying the font style, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default for **FontBold**) Turns on the formatting in that style. |
| **False** | Turns off the formatting in that style. |

### Remarks

Use these font properties to format text, either at design time using the Properties window or at run time using code. Font changes take effect on the screen immediately.

**Note** Fonts available in Visual Basic vary depending on your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which actual fonts exist.

In general, you should change the **FontName** property before you set size and style attributes with the **FontSize**, **FontBold**, **FontItalic**, **FontStrikethru**, and **FontUnderline** properties. However, when you set TrueType fonts to smaller than 8 points, you should set the point size with the **FontSize** property, then set the **FontName** property, and finally set the size again with the **FontSize** property. The Microsoft Windows operating environment uses a different font for TrueType fonts that are smaller than 8 points.

**See Also**

**Font** Property

**FontName** Property

**FontSize** Property

## FontName Property (Custom Control)

See Also

Returns or sets the font used to display text in a control or in a run-time drawing or printing operation.

---

**Note** The **FontName** property is included for compatibility with earlier versions of Visual Basic. For additional functionality, use the new **Font** object properties.

---

### Syntax

*object*.**FontName** [= *font*]

The **FontName** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D option button, 3D panel, Animated button, Masked edit, or Outline control. |
| *font* | A string expression specifying the font name to use. |

### Remarks

The default for this property is determined by the system. Fonts available with Visual Basic vary depending on your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which fonts exist.

In general, you should change **FontName** before setting size and style attributes with the **FontSize**, **FontBold**, **FontItalic**, **FontStrikethru**, and **FontUnderline** properties.

---

**Note** At run time, you can get information on fonts available to the system through the **FontCount** and **Fonts** properties.

---

**See Also**

**Font** Property
**FontBold**, **FontItalic**, **FontStrikethru**, **FontUnderline** Properties
**FontSize** Property

## FontSize Property (Custom Control)

See Also

Returns or sets the size of the font to be used for text displayed in a control or in a run-time drawing or printing operation.

**Note** The **FontSize** property is included for compatibility with earlier versions of Visual Basic. For additional functionality, use the new **Font** object properties.

### Syntax

*object*.**FontSize** [= *points*]

The **FontSize** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D option button, 3D panel, Animated button, Masked edit, or Outline control. |
| *points* | A numeric expression specifying the font size to use, in points. |

### Remarks

Use this property to format text in the font size you want. The default is determined by the system. To change the default, specify the size of the font in points.

The maximum value for **FontSize** is 2160 points.

**Note** Fonts available with Visual Basic vary depending on your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which fonts exist.

In general, you should change the **FontName** property before you set size and style attributes with the **FontSize**, **FontBold**, **FontItalic**, **FontStrikethru**, and **FontUnderline** properties. However, when you set TrueType fonts to smaller than 8 points, you should set the point size with the **FontSize** property, then set the **FontName** property, and finally set the size again with the **FontSize** property. The Microsoft Windows operating environment uses a different font for TrueType fonts that are smaller than 8 points.

**See Also**

**Font** Property

**FontBold**, **FontItalic**, **FontStrikethru**, **FontUnderline** Properties

**FontName** Property

## Height, Width Properties (Custom Controls)

See Also

Return or set the dimensions of an object.

### Syntax

*object*.**Height** [= *number*]

*object*.**Width** [= *number*]

The **Height** and **Width** properties syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button 3D panel, Animated button, Graph, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. Also applies to a Button object of a Toolbar control, ListItem object of a ListView control or Tab object of a TabStrip control. |
| | An object expression that evaluates to a Key state control, and a ColumnHeader object of a ListView control for the **Width** property only. |
| *number* | A numeric expression specifying the dimensions of an object. Measurements are from the center of the object's border so that objects with different border widths align correctly. These properties use the scale units of the object's container. |

### Remarks

The values for these properties change as the object is resized. Maximum limits of these properties for all objects are system-dependent.

Use the **Height**, **Width**, **Left**, and **Top** properties for operations or calculations based on an object's total area, such as sizing or moving the object. For the **TabStrip** control, use the **ClientLeft**, **ClientTop**, **ClientHeight**, and **ClientWidth** properties for operations or calculations based on an object's internal area, such as drawing or moving objects within another object.

For **ListItem** objects in the **ListView** control, the **Height**, **Width**, **Left**, and **Top** properties are read-only in List and Report views.

For a **Tab** object in a **TabStrip** control, the **Height** and **Width** properties are read-only and always reflect the current height and width of each tab. These properties, along with **Left** and **Top**, are useful if you want to return the coordinates of the active tab in order to cover it with another object, such as a **PictureBox** control.

**See Also**

**ClientHeight**, **ClientWidth**, **ClientLeft**, **ClientTop** Properties
**Left**, **Top** Properties
**View** Property

## Left, Top Properties (Custom Controls)

See Also

- **Left**

■returns or sets the distance between the internal left edge of an object and the left edge of its container.

- **Top**

■returns or sets the distance between the internal top edge of an object and the top edge of its container.

### Syntax

*object*.**Left** [= *number*]

*object*.**Top** [= *number*]

The **Left** and **Top** properties syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar or TreeView control. Also applies to a Button object of a Toolbar control, ListItem object of a ListView control or Tab object of a TabStrip control. |
| | An object expression that evaluates to ColumnHeader objects of a ListView control or a Panel object of a StatusBar control for the **Left** property only. |
| *number* | A numeric expression specifying distance. |

### Remarks

The **Left** and **Top** properties are measured in units whose size depends on the coordinate system of the object's container. The values for these properties change as the object is moved by the user or by code.

For both properties, you can specify a single-precision number.

Use the **Left**, **Top**, **Height**, and **Width** properties for operations based on an object's external dimensions, such as moving or resizing.

For a **ListItem** object in a **ListView** control, the **Left** and **Top** properties are read-only in List and Report views. They are read/write in Icon and SmallIcon views. For **ColumnHeader** objects, the **Left** property is read-only.

For a **Tab** object in a **TabStrip** control, the **Left** and **Top** properties are read-only and always reflect the current position of each tab. These properties, along with **Height** and **Width**, are useful if you want to return the coordinates of the active tab in order to cover it with another object, such as a **PictureBox** control.

**See Also**

**Height**, **Width** Properties

**View** Property

## ListCount Property (Outline Control)

See Also

Returns the number of items in the list portion of a control.

### Syntax

*object*.**ListCount**

The *object* placeholder represents an object expression that evaluates to an Outline control.

### Remarks

If no item is selected, the **ListIndex** property value is ▪1.   The first item in the list is **ListIndex** = 0, and **ListCount** is always one more than the largest **ListIndex** value.

**See Also**

**ListIndex** Property (**Outline** Control)

## ListIndex Property (Outline Control)

See Also

Returns or sets the index of the currently selected item in the control.   Not available at design time.

**Syntax**

*object*.**ListIndex** [= *index*]

The **ListIndex** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an Outline control. |
| *index* | A numeric expression specifying the index of the current item, as described in Settings. |

**Settings**

The settings for *index* are:

| Setting | Description |
|---|---|
| ■1 | Indicates no item is currently selected. |
| *n* | A number indicating the index of the currently selected item. |

**Remarks**

The first item in the list is **ListIndex** is equal to 0, and **ListCount** is always one more than the largest **ListIndex** value.

**See Also**

**ListCount** Property (**Outline** Control)

## TabIndex Property (Custom Controls)

See Also

Returns or sets the tab order of an object within its parent form.

### Syntax

*object*.**TabIndex** [= *index*]

The **TabIndex** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button 3D panel, Animated button, Gauge, Graph, Key state, Masked edit, Multimedia MCI, Outline, ListView, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. |
| *index* | An integer from 0 to (*n*▪1), where *n* is the number of controls on the form that have a **TabIndex** property. Assigning a **TabIndex** value of less than 0 generates an error. |

### Remarks

By default, Visual Basic assigns a tab order to controls as you draw them on a form. Each new control is placed last in the tab order. If you change the value of a control's **TabIndex** property to adjust the default tab order, Visual Basic automatically renumbers the **TabIndex** of other controls to reflect insertions and deletions. You can make changes at design time using the Properties window or at run time in code.

All controls except menus and timers are included in the tab order. At run time, invisible or disabled controls remain in the tab order but are skipped during tabbing.

The **TabIndex** property isn't affected by the **ZOrder** method.

**See Also**

**ZOrder** Method

## Tag Property (Custom Controls)

Returns or sets any extra data needed for your program. Unlike other properties, the value of the **Tag** property isn't used by Visual Basic; you can use this property to identify objects.

### Syntax

*object*.**Tag** [= *expression*]

The **Tag** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button 3D panel, Animated button, Communications, Gauge, Graph, ImageList, Key state, ListView, MAPI session, MAPI message, Masked edit, Multimedia MCI, Outline, Picture clip, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. Also applies to a Button object of a Toolbar control, ColumnHeader object of a ListView control, ListItem object of a ListView control, Node object of a TreeView control, or Tab object of a TabStrip control. |
| *expression* | A string expression identifying the object. The default is a zero-length string (""). |

### Remarks

The **Tag** property is a user-defined property.

You can use this property to assign an identification string to an object without affecting any of its other property settings or causing side effects. The **Tag** property is useful when you need to check the identity of a control that is passed as a variable to a procedure.

## Visible Property (Custom Controls)

Returns or sets a value indicating whether an object is visible or hidden.

### Syntax

*object*.**Visible** [= *boolean*]

The **Visible** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Outline, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. Also applies to a Button object of a Toolbar control, or Node object of a TreeView control, or Panel object of a StatusBar control. |
| *boolean* | A Boolean expression specifying whether the object is visible or hidden, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) Object is visible. |
| **False** | Object is hidden. |

### Remarks

To hide an object at start up, set the **Visible** property to **False** at design time. Setting this property in code enables you to hide and later redisplay a control at run time in response to a particular event.

## Picture Property (Custom Controls)

See Also

Returns or sets a graphic to be displayed in a control.

### Syntax

*object*.**Picture** [= *picture*]

The **Picture** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **ListImage** or **Panel** object. |
| *picture* | A string expression specifying a file containing a graphic, as described in Settings. |

### Settings

The settings for *picture* are:

| Setting | Description |
|---|---|
| (None) | (Default) No picture. |
| (Bitmap, icon) | Specifies a graphic. At run time, you can also set this property using the **LoadPicture** function on a bitmap or icon. |

### Remarks

At design time, you can transfer a graphic with the **Clipboard** using the Copy, Cut, and Paste commands on the Edit menu. At run time, you can use Clipboard methods such as **GetData**, **SetData**, and **GetFormat** with the nontext Clipboard constants **vbCFBitmap**, **vbCFMetafile**, and **vbCFDIB**, which are listed in the object library in the Object Browser.

When setting the **Picture** property at design time, the graphic is saved and loaded with the form. If you create an executable file, the file contains the image. When you load a graphic at run time, the graphic isn't saved with the application. Use the **SavePicture** statement to save a graphic from a form or picture box into a file.

**Note** At run time, the **Picture** property can be set to any other object's **DragIcon**, **Icon**, **Image**, or **Picture** property, or you can assign it the graphic returned by the **LoadPicture** function.

**See Also**

**Add** Method (**ListImages** Collection)
**Add** Method (**Panels** Collection)
**DragIcon** Property
**Icon** Property
**Image** Property
**ImageList** Control
**ListImage** Object, **ListImages** Collection
**Panel** Object, **Panels** Collection

## Icon Property (Custom Controls)

See Also

Returns the icon displayed when a form is minimized at run time.

### Syntax

*object*.**Icon**

The *object* placeholder represents an object expression that evaluates to a **Form** object.

### Remarks

Use this property to specify an icon for any form that the user can minimize at run time.

For example, you can assign a unique icon to a form to indicate the form's function. Specify the icon by loading it using the Properties window at design time. The file you load must have the .ICO filename extension and format. If you don't specify an icon, the Visual Basic default icon for forms is used.

You can use the Visual Basic Icon Library (in the ICONS subdirectory) as a source for icons. When you create an executable file, you can assign an icon to the application by using the **Icon** property of any form in that application.

---

**Note** To see a form's icon, the form must be minimized and the **BorderStyle** property must be set to 1 (**ccFixedSingle**).

At run time, you can assign an object's **Icon** property to another object's **DragIcon** or **Icon** property. You can also assign an icon returned by the **LoadPicture** function. Using **LoadPicture** without an argument assigns an empty (null) icon to the form, which enables you to draw on the icon at run time.

---

**See Also**

**BorderStyle** Property
**DragIcon** Property
**Icon** Property

## Cancel Property (3D Command Button Control)

See Also

Returns or sets a value indicating whether a command button is the Cancel button on a form.

### Syntax

*object*.**Cancel** [= *boolean*]

The **Cancel** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D command button control. |
| *boolean* | A Boolean expression specifying whether the object is the Cancel button, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | The 3D command button control is the Cancel button. |
| **False** | (Default) The 3D command button control isn't the Cancel button. |

### Remarks

Use the **Cancel** property to give the user the option of canceling uncommitted changes and returning the form to its previous state.

Only one 3D command button control on a form can be the Cancel button.   When the **Cancel** property is set to **True** for one command button, it's automatically set to **False** for all other command buttons on the form.   When a 3D command button control's **Cancel** property setting is **True** and the form is the active form, the user can choose the command button by clicking it, pressing the ESC key, or pressing ENTER when the button has the focus.

**See Also**

**Default** Property

KeyDown, KeyUp Events

KeyPress Event

## Align Property (Custom Controls)

See Also

Returns or sets a value that determines whether an object is displayed in any size anywhere on a form or whether it's displayed at the top, bottom, left, or right of the form and is automatically sized to fit the form's width.

### Syntax

*object*.**Align** [**=** *integer*]

The **Align** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D panel, ProgressBar, StatusBar, or Toolbar control. |
| *integer* | An integer specifying how an object is displayed, as described in Settings. |

### Settings

The settings for *integer* are:

| Setting | Description |
|---|---|
| 0 | (Default) None. Size and location can be set at design time or in code. |
| 1 | Top. Object is at the top of the form, and its width is equal to the form's **ScaleWidth** property setting. |
| 2 | Bottom. Object is at the bottom of the form, and its width is equal to the form's **ScaleWidth** property setting. |
| 3 | Left. Object is at the left of the form, and its width is equal to the form's **ScaleWidth** property setting. |
| 4 | Right. Object is at the right of the form, and its width is equal to the form's **ScaleWidth** property setting. |

### Remarks

You can use the **Align** property to quickly create a toolbar or status bar at the top or bottom of a form. As a user changes the size of the form, an object with **Align** set to 1 or 2 automatically resizes to fit the width of the form.

**See Also**

**Negotiate** Property

## AutoSize Property (Custom Controls)

Returns or sets a value that determines whether a control is automatically resized to display its entire contents.

### Syntax

*object*.**AutoSize** [**=** *boolean*]

The **AutoSize** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to the Gauge and Key state controls. |
| *boolean* | A Boolean expression specifying whether the control is resized, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | Automatically resizes the control to display its entire contents. |
| **False** | (Default) Keeps the size of the control constant.   Contents are clipped when they exceed the area of the control. |

## DragIcon Property (Custom Controls)

See Also

Returns or sets the icon to be displayed as the pointer in a drag-and-drop operation.

### Syntax

*object*.**DragIcon** [**=** *icon*]

The **DragIcon** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, Spin button, RichTextBox, Slider, StatusBar, SSTab, TabStrip, Toolbar, or TreeView control. |
| *icon* | Any code reference that returns a valid icon, such as a reference to a form's icon (`Form1.Icon`), a reference to another control's **DragIcon** property (`Text1.DragIcon`), or the **LoadPicture** function, as described in Settings. |

### Settings

The settings for *icon* are:

| Setting | Description |
|---|---|
| (none) | (Default) An arrow pointer inside a rectangle. |
| Icon | A custom mouse pointer. You specify the icon by setting it using the Properties window at design time. You can also use the **LoadPicture** function at run time. The file you load must have the .ICO filename extension and format. |

### Remarks

You can use the **DragIcon** property to provide visual feedback during a drag-and-drop operation▪for example, to indicate that the source control is over an appropriate target. **DragIcon** takes effect when the user initiates a drag-and-drop operation. Typically, you set **DragIcon** as part of a MouseDown or DragOver event procedure.

**Note** At run time, the **DragIcon** property can be set to any object's **DragIcon** or **Icon** property, or you can assign it an icon returned by the **LoadPicture** function.

**See Also**

**Drag** Method
DragOver Event
**Icon** Property
MouseDown, MouseUp Events

## DragMode Property (Custom Controls)

See Also

Returns or sets a value that determines whether manual or automatic drag mode is used for a drag-and-drop operation.

### Syntax

*object*.**DragMode** [**=** *number*]

The **DragMode** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, Spin button, RichTextBox, Slider, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. |
| *number* | An integer specifying the drag mode, as described in Settings. |

### Settings

The settings for *number* are:

| Setting | Description |
|---|---|
| 0 | (Default) Manual. Requires using the **Drag** method to initiate a drag-and-drop operation on the source control. |
| 1 | Automatic. Clicking the source control automatically initiates a drag-and-drop operation. |

### Remarks

When **DragMode** is set to 1 (Automatic), the control doesn't respond as usual to mouse events. Use the 0 (Manual) setting to determine when a drag-and-drop operation begins or ends. You can use this setting to initiate a drag-and-drop operation in response to a keyboard or menu command or to enable a source control to recognize a MouseDown event prior to a drag-and-drop operation.

Clicking while the mouse pointer is over a target object or form during a drag-and-drop operation generates a DragDrop event for the target object. This ends the drag-and-drop operation. A drag-and-drop operation may also generate a DragOver event.

**Note** While a control is being dragged, it can't recognize other user-initiated mouse or keyboard events (KeyDown, KeyPress or KeyUp, MouseDown, MouseMove, or MouseUp). However, the control can receive events initiated by code or by a DDE link.

**See Also**

**Drag** Method
DragDrop Event
DragOver Event
KeyDown, KeyUp Events
KeyPress Event
MouseDown, MouseUp Events
MouseMove Event

## hWnd Property (Custom Controls)

Returns a handle to a control.

### Syntax

*object*.**hWnd**

The *object* placeholder represents an object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, Picture clip, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control.

### Remarks

The Microsoft Windows operating environment identifies each control in an application by assigning it a handle, or **hWnd**. The **hWnd** property is used with Windows API calls. Many Windows operating environment functions require the **hWnd** of the active window as an argument.

**Note** Because the value of this property can change while a program is running, never store the **hWnd** value in a variable.

## ItemData Property (Outline Control)

Returns or sets a specific number for each item in an Outline control.   Not available at design time.

**Syntax**

*object*.**ItemData(***index***)** [**=** *number*]

The **ItemData** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an Outline control. |
| *index* | The number of a specific item in the control. |
| *number* | The number to be associated with the specified item. |

**Remarks**

This property is an array of long integer values with the same number of items as a control's **List** property.   You can use the numbers associated with each item to identify the items in code.

**Note**     When you insert an item into a list with the **AddItem** method, an item is automatically inserted in the **ItemData** array as well.   However, the value isn't reinitialized to zero; it retains the value that was in that position before you added the item to the list.   When you use the **ItemData** property, be sure to set its value when adding new items to a list.

**See Also**

**AddItem** Method

**List** Property

## Locked Property (RichTextBox Control)

See Also

Returns or sets a value indicating whether the contents in a **RichTextBox** control can be edited.

### Syntax

*object*.**Locked** [= *boolean*]

The **Locked** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *boolean* | A Boolean expression specifying whether the contents of the control can be edited, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | You can scroll and highlight the text in the control, but you can't edit it.   The program can still modify the text by changing the **Text** property. |
| **False** | (Default)   You can edit the text in the control. |

**See Also**

**Text** Property

## Default Property (3D command button)

See Also

Returns or sets a value that determines which 3D command button control is the default command button on a form.

### Syntax

*object*.**Default** [**=** *boolean*]

The **Default** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D command button control. |
| *boolean* | A Boolean expression specifying whether the command button is the default, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | The 3D command button is the default command button. |
| **False** | (Default) The 3D command button isn't the default command button. |

### Remarks

Only one command button on a form can be the default command button. When **Default** is set to **True** for one command button, it's automatically set to **False** for all other command buttons on the form. When the command button's **Default** property setting is **True** and its parent form is active, the user can choose the command button (invoking its Click event) by pressing ENTER. Any other control with the focus doesn't receive a keyboard event (KeyDown, KeyPress, or KeyUp) for the ENTER key unless the user has moved the focus to another command button on the same form. In this case, pressing ENTER chooses the command button that has the focus instead of the default command button.

For a form or dialog box that supports an irreversible action such as a delete operation, make the Cancel button the default command button by setting its **Default** property to **True**.

**See Also**

**Cancel** Property
KeyDown, KeyUp Events
KeyPress Event

## MousePointer Property (Custom Controls)

See Also

Returns or sets a value indicating the type of mouse pointer displayed when the mouse is over a particular part of an object at run time.

### Syntax

*object*.**MousePointer** [**=** *value*]

The **MousePointer** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Key state, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. |
| *value* | A value or constant specifying the type of mouse pointer displayed, as described in Settings. |

### Settings

The settings for *value* are (for all controls except **RichTextBox**):

| Constant | Value | Description |
|---|---|---|
| **ccDefault** | 0 | (Default) Shape determined by the object. |
| **ccArrow** | 1 | Arrow. |
| **ccCross** | 2 | Cross (cross-hair pointer). |
| **ccIbeam** | 3 | I Beam. |
| **ccIcon** | 4 | Icon (small square within a square). |
| **ccSize** | 5 | Size (four-pointed arrow pointing north, south, east, and west). |
| **ccSizeNESW** | 6 | Size NE SW (double arrow pointing northeast and southwest). |
| **ccSizeNS** | 7 | Size N S (double arrow pointing north and south). |
| **ccSizeNWSE** | 8 | Size NW, SE (double arrow pointing northwest and southeast). |
| **ccSizeEW** | 9 | Size E W (double arrow pointing east and west). |
| **ccUpArrow** | 10 | Up Arrow. |
| **ccHourglass** | 11 | Hourglass (wait). |
| **ccNoDrop** | 12 | No Drop. |
| **ccArrowHourglass** | 13 | Arrow and hourglass. (Only available in 32-bit Visual Basic.) |
| **ccArrowQuestion** | 14 | Arrow and question mark. (Only available in 32-bit Visual Basic.) |
| **ccSizeAll** | 15 | Size all. (Only available in 32-bit Visual Basic.) |
| **ccCustom** | 99 | Custom icon specified by the **MouseIcon** property. |

### Remarks

You can use this property when you want to indicate changes in functionality as the mouse pointer passes over controls on a form or dialog box. The Hourglass setting (11) is useful for indicating that the user should wait for a process or operation to finish.

**Note** If your application doesn't call the **DoEvents** function and isn't a 32-bit application, it overrides all **MousePointer** settings for all controls and other applications. If your application calls DoEvents, the **MousePointer** property may temporarily change when over a custom control.

The cc prefix refers to the Windows 95 controls. Prefixes for the settings change with the specific control or group of controls. However, the description remains the same unless indicated.

**See Also**

**DragIcon** Property
**MouseIcon** Property
MouseMove Event
**RichTextBox** Control Constants

## SelLength, SelStart, SelText Properties (Custom Controls)

See Also

- **SelLength**

■returns or sets the number of characters selected.

- **SelStart**

■returns or sets the starting point of text selected; indicates the position of the insertion point if no text is selected.

- **SelText**

■returns or sets the string containing the currently selected text; consists of a zero-length string ("") if no characters are selected.

These properties aren't available at design time.

### Syntax

*object*.**SelLength** [**=** *number*]

*object*.**SelStart** [**=** *index*]

*object*.**SelText** [**=** *value*]

The **SelLength**, **SelStart**, and **SelText** property syntaxes have these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a RichTextBox control. |
| | **SelLength** and **SelStart** properties only: Masked edit or Slider control. |
| *number* | A numeric expression specifying the number of characters selected. |
| *index* | A numeric expression specifying the starting point of the selected text. |
| *value* | A string expression containing the selected text. |

### Remarks

Use these properties for tasks such as setting the insertion point, establishing an insertion range, selecting substrings in a control, or clearing text.   Used in conjunction with the **Clipboard** object, these properties are useful for copy, cut, and paste operations.

When working with these properties:

- Setting **SelLength** less than 0 causes a run-time error.
- Setting **SelStart** greater than the text length sets the property to the existing text length; changing **SelStart** changes the selection to an insertion point and sets **SelLength** to 0.
- Setting **SelText** to a new value sets **SelLength** to 0 and replaces the selected text with the new string.

**See Also**

**SelText** Property(Masked edit)

**Text** Property

## TabStop Property (Custom Controls)

See Also

Returns or sets a value indicating whether a user can use the TAB key to give the focus to an object.

### Syntax

*object*.**TabStop** [**=** *boolean*]

The **TabStop** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D option button, Animated button, Gauge, Graph, Key state, ListView**,** Masked edit, Multimedia MCI, Outline, RichTextBox, Slider, SSTab, TabStrip, or TreeView control. |
| *boolean* | A Boolean expression specifying whether the object is a tab stop, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) Designates the object as a tab stop. |
| **False** | Bypasses the object when the user is tabbing, although the object still holds its place in the actual tab order, as determined by the **TabIndex** property. |

### Remarks

This property enables you to add or remove a control from the tab order on a form. For example, if you're using a **PictureBox** control to draw a graphic, set its **TabStop** property to **False**, so the user can't tab to the **PictureBox**.

**See Also**

**TabIndex** Property

## MouseIcon Property (Custom Controls)

See Also

Sets a custom mouse icon.

### Syntax

*object*.**MouseIcon = LoadPicture(***pathname***)**

*object*.**MouseIcon =** *picture*

The **MouseIcon** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to one of the following controls: 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Key status, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Spin button, StatusBar, SSTab, TabStrip, ToolBar or TreeView. |
| *pathname* | A string expression specifying the path and filename of the file containing the custom icon. |
| *picture* | An object expression that evaluates to a Picture, most commonly the **Picture** property from a **Form** object, **PictureBox** control, or **Image** control. |

### Remarks

The **MouseIcon** property provides a custom icon that is used when the **MousePointer** property is set to 99.

Although Visual Basic 4.0 does not create cursor (.CUR) files, you can use the **MouseIcon** property to load either cursor or icon files. This provides your program with easy access to custom cursors of any size, with any desired hot spot location. The 32-bit version of Visual Basic does not load animated cursor (.ANI) files, even though 32-bit versions of Windows support these cursors.

**See Also**

**DragIcon** Property
**Icon** Property
**MousePointer** Property
**Picture** Property

## MaxLength Property (RichTextBox Control)

See Also

Returns or sets a value indicating whether there is a maximum number of characters a **RichTextBox** control can hold and, if so, specifies the maximum number of characters.

### Syntax

*object*.**MaxLength** [**=** *integer*]

The **MaxLength** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **RichTextBox** control. |
| *integer* | An integer specifying the maximum number of characters a user can enter in the control. The default for the **MaxLength** property is 0, indicating no maximum other than that created by memory constraints on the user's system.   Any number greater than 0 indicates the maximum number of characters. |

### Remarks

Use the **MaxLength** property to limit the number of characters a user can enter in a **RichTextBox**.

If text that exceeds the **MaxLength** property setting is assigned to a **RichTextBox** from code, no error occurs; however, only the maximum number of characters is assigned to the **Text** property, and extra characters are truncated.   Changing this property doesn't affect the current contents of a **RichTextBox**, but will affect any subsequent changes to the contents.

**See Also**

**MultiLine** Property

**Text** Property

## MultiLine Property (RichTextBox Control)

See Also

Returns or sets a value indicating whether a **RichTextBox** control can accept and display multiple lines of text.   Read-only at run time.

### Syntax

*object*.**MultiLine**

The *object* placeholder represents an object expression that evaluates to a **RichTextBox** control.

### Settings

The **MultiLine** property settings are:

| Setting | Description |
|---|---|
| **True** | Allows multiple lines of text. |
| **False** | (Default) Ignores carriage returns and restricts data to a single line. |

### Remarks

A multiple-line **RichTextBox** control wraps text as the user types text extending beyond the text box.

You can also add scroll bars to a larger **RichTextBox** control using the **ScrollBars** property.   If no **HScrollBar** control (horizontal scroll bar) is specified, the text in a multiple-line **RichTextBox** automatically wraps.

**Note**   On a form with no default button, pressing ENTER in a multiple-line **RichTextBox** control moves the focus to the next line.   If a default button exists, you must press CTRL+ENTER to move to the next line.

**See Also**

**ScrollBars** Property

## Appearance Property (Custom Controls)

See Also

Returns or sets the paint style of a control on a **Form** object at run time.   Read-only at run time.

### Syntax

[*object*]**.Appearance**

The *object* placeholder represents an object expression that evaluates to a ListView, Masked edit, ProgressBar, RichTextBox, or TreeView control.

### Settings

The **Appearance** property settings are:

| Setting | Description |
|---|---|
| 0 | Flat.   Paints controls and forms with without visual effects. |
| 1 | (Default) 3D.   Paints controls with three-dimensional effects. |

### Remarks

If set to 1 at design time, the **Appearance** property draws the control with three-dimensional effects. Setting the **Appearance** property to 1 also causes the form and its controls to have their **BackColor** property set to the color selected for Button Face in the Color option of the operating system's Control Panel.

**See Also**

**BackColor**, **ForeColor** Properties

## ScrollBars Property (RichTextBox Control)

See Also

Returns or sets a value indicating whether a **RichTextBox** control has horizontal or vertical scroll bars. Read-only at run time.

### Syntax

*object*.**ScrollBars**

The *object* placeholder represents an object expression that evaluates to a **RichTextBox** control.

### Settings

The **ScrollBars** property settings are:

| Constant | Value | Description |
|---|---|---|
| **rtfNone** | 0 | (Default) No scroll bars shown. |
| **rtfHorizontal** | 1 | Horizontal scroll bar only. |
| **rtfVertical** | 2 | Vertical scroll bar only. |
| **rtfBoth** | 3 | Both horizontal and vertical scroll bars shown. |

### Remarks

For a **RichTextBox** control with setting 1 (Horizontal), 2 (Vertical), or 3 (Both), you must set the **MultiLine** property to **True**.

At run time, the Microsoft Windows operating environment automatically implements a standard keyboard interface to allow navigation in **RichTextBox** controls with the arrow keys (UP ARROW, DOWN ARROW, LEFT ARROW, and RIGHT ARROW), the HOME and END keys, and so on.

Scroll bars are displayed only if the contents of the **RichTextBox** extend beyond the control's borders. If **ScrollBars** is set to **False**, the control won't have scroll bars, regardless of its contents.

**See Also**

**MultiLine** Property

## Caption Property (Custom Controls)

See Also

Returns or sets the caption for an object. For the **SSTab** control, returns or sets the caption for the active tab.

### Syntax

*object*.**Caption** [**=** *string*]

The **Caption** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a Button or Tab object, or to a 3D check box, 3D command button, 3D frame, 3D option button, 3D panel, Animated button, or SSTab control. |
| *string* | A string expression that evaluates to the text displayed as the caption. |

### Remarks

When you create a new object, its default caption is the default **Name** property setting. This default caption includes the object name and an integer, such as Tab1. For a more descriptive label, set the **Caption** property.

You can use the **Caption** property to assign an access key to a control. In the caption, include an ampersand (&) immediately preceding the character you want to designate as an access key. The character is underlined. Press ALT plus the underlined character to move the focus to that control. To include an ampersand in a caption without creating an access key, include two ampersands (&&). A single ampersand is displayed in the caption and no characters are underlined.

When using the **SSTab** control, you can use the **Caption** property at design time to set the **TabCaption()** property to the tab specified by the **Tab** property.

**See Also**

**Name** Property
**Tab** Property (**SSTab** Control)
**TabCaption** Property

## Enabled Property (Custom Controls)

Returns or sets a value that determines whether an object can respond to user-generated events.

### Syntax

*object*.**Enabled** [**=** *boolean*]

The **Enabled** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a Button or Panel object, or to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Outline, ProgressBar, RichTextBox, Slider, Spin button, Status Bar, SSTab, TabStrip, Toolbar, or TreeView control. |
| *boolean* | A Boolean expression specifying whether *object* can respond to user-generated events, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) Allows *object* to respond to events. |
| **False** | Prevents *object* from responding to events. |

### Remarks

The **Enabled** property allows objects to be enabled or disabled at run time.   For example, you can disable objects that don't apply to the current state of the application.

## Font Property (Custom Controls)

Returns a **Font** object.

### Syntax

*object*.**Font**

The *object* placeholder represents an object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D option button, 3D panel, Animated button, ListView, Masked edit, Outline, RichTextBox, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control.

### Remarks

Use the **Font** property of an object to identify a specific **Font** object whose properties you want to use. For example, the following code changes the **Bold** property setting of a **Font** object identified by the **Font** property of a **TextBox** object:

```
txtFirstName.Font.Bold = True
```

## HelpContextID Property (Custom Controls)

Returns or sets an associated context number for an object. Used to provide context-sensitive Help for your application.

### Syntax

*object*.**HelpContextID** [**=** *number*]

The **HelpContextID** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D option button, Animated button, Gauge, Graph, Key state, ListView, Masked edit, Multimedia MCI, Outline, RichTextBox, Slider, Spin button, SSTab, TabStrip, or TreeView control. If *object* is omitted, the form associated with the active form module is assumed to be *object.* |
| *number* | A numeric expression specifying the context number of the Help topic associated with *object*, as described in Settings. |

### Settings

The settings for *number* are:

| Setting | Description |
|---|---|
| 0 | (Default) No context number specified. |
| > 0 | An integer specifying a valid context number. |

### Remarks

For context-sensitive Help on an object in your application, you must assign the same context number to both *object* and to the associated Help topic when you compile your Help file.

If you've created a Microsoft Windows operating environment Help file for your application and set the application's **HelpFile** property, when a user presses the F1 key, Visual Basic automatically calls Help and searches for the topic identified by the current context number.

The current context number is the value of **HelpContextID** for the object that has the focus. If **HelpContextID** is set to 0, then Visual Basic looks in the **HelpContextID** of the object's container, then that object's container, and so on. If a nonzero current context number can't be found, the F1 key is ignored.

**Note** Building a Help file requires the Microsoft Windows Help Compiler, which is included with the Visual Basic Professional Edition.

## Index Property (Control Array)

See Also

Returns or sets the number that uniquely identifies a control in a control array.   Available only if the control is part of a control array.

### Syntax

*object*[**(***number***)**]**.Index**

The **Index** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Communications, Gauge, Graph, ImageList, Key state, ListView, MAPI Session, MAPI Messages, Masked edit, Multimedia MCI, Outline, Picture clip, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. |
| *number* | A numeric expression that evaluates to an integer that identifies an individual control within a control array, as described in Settings. |

### Settings

The settings for *number* are:

| Setting | Description |
|---|---|
| No value | (Default) Not part of a control array. |
| 0 to 32,767 | Part of an array.   Specifies an integer greater than or equal to 0 that identifies a control within a control array.   All controls in a control array have the same **Name** property. Visual Basic automatically assigns the next integer available within the control array. |

### Remarks

Because control array elements share the same **Name** property setting, you must use the **Index** property in code to specify a particular control in the array.   **Index** must appear as an integer (or a numeric expression evaluating to an integer) in parentheses next to the control array name■for example, `MyButtons(3)`.   You can also use the **Tag** property setting to distinguish one control from another within a control array.

When a control in the array recognizes that an event has occurred, Visual Basic calls the control array's event procedure and passes the applicable **Index** setting as an additional argument.   This property is also used when you create controls dynamically at run time with the **Load** statement or remove them with the **Unload** statement.

Although Visual Basic assigns, by default, the next integer available as the value of **Index** for a new control in a control array, you can override this assigned value and skip integers.   You can also set **Index** to an integer other than 0 for the first control in the array.   If you reference an **Index** value in code that doesn't identify one of the controls in a control array, a Visual Basic run-time error occurs.

**Note**     To remove a control from a control array, change the control's **Name** property setting, and delete the control's **Index** property setting.

**See Also**

**Name** Property

**Tag** Property

## Name Property (Custom Controls)

See Also

Returns the name used in code to identify an object.

### Syntax

*object*.**Name**

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Communications, Gauge, Graph, ImageList, Key state, ListView, MAPI Session, MAPI Messages, Masked edit, Multimedia MCI, Outline, Picture clip, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, ToolBar, or TreeView control. |

### Remarks

The default name for new objects is the kind of object plus a unique integer.   For example, the first new **ListView** control is ListView1, a new **ProgressBar** control is ProgressBar1, and the third **ImageList** control you create on a form is ImageList3.

An object's **Name** property must start with a letter and can be a maximum of 40 characters.   It can include numbers and underlined (_) characters but can't include punctuation or spaces.

You can create an array of controls of the same type by setting the **Name** property to the same value. For example, when you set the name of all option buttons in a group to MyOpt, Visual Basic assigns unique values to the **Index** property of each control to distinguish it from others in the array.   Two controls of different types can't share the same name.

**Note**   Although Visual Basic often uses the **Name** property setting as the default value for the **Caption** and **Text** properties, changing one of these properties doesn't affect the others.

**See Also**

**Caption** Property

**Caption** Property (**Tab** Object)

**Text** Property

**Text** Property (**Masked Edit** Control)

## Parent Property (Custom Controls)

Returns the form on which a control is located.

**Syntax**

*object*.**Parent**

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Communications, Graph, Gauge, ImageList, Key state, ListView, MAPI Messages, MAPI Session, Masked edit, Multimedia MCI, Outline, Picture clip, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. |

**Remarks**

Use the **Parent** property to access the properties, methods, or controls of a control's parent form, for example:

```
MyButton.Parent.MousePointer = 4
```

The **Parent** property is useful in an application in which you pass controls as arguments.   For example, you could pass a control variable to a general procedure in a module, and use the **Parent** property to access its parent form.

## Count Property (Custom Controls)

See Also

Returns the number of members in a collection.

### Syntax

*object*.**Count**

The object qualifier is an object expression that evaluates to one of the following collections: **Buttons**, **ColumnHeaders**, **ListImages**, **ListItems**, **Nodes**, **Panels**, **Tabs**.

### Remarks

The **Count** property is associated with the collection and not the control itself. For example, to get a count of the **Tab** objects in a **Tabs** collection in a **TabStrip** control, use the following code:

```
'To count the number of tabs.
x = TabStrip1.Tabs.Count
```

**See Also**

**Add** Method (**Buttons** Collection)
**Add** Method (**ColumnHeaders** Collection)
**Add** Method (**ListImages** Collection)
**Add** Method **(ListItems** Collection)
**Add** Method (**Nodes** Collection)
**Add** Method (**Panels** Collection)
**Add** Method (**Tabs** Collection)
**Item** Method
**Remove** Method

## Object Property (Custom Controls)

Returns a reference to a property or method of a custom control that has the same name as a property or method automatically extended to the control by Visual Basic.

**Syntax**

*object*.**Object**[.*property* | .*method*]

The **Object** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Communications, Graph, Gauge, ImageList, Key state, ListView, MAPI Messages, MAPI Session, Masked edit, Multimedia MCI, Outline, Picture clip, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. |
| *property* | Property of the custom control that is identical to the name of a Visual Basic-supplied property. |
| *method* | Method of the custom control that is identical to the name of a Visual Basic-supplied method. |

**Remarks**

The **Object** property returns the value of the property or method specified.

Visual Basic supplies a standard set of properties and methods to all custom controls in a Visual Basic project. It is possible for a custom control to define a property or method which has the same name as one of these standard properties or methods. When this occurs, Visual Basic automatically uses the property or method it supplies instead of the one with the same name defined in the custom control. The **Object** property allows you to bypass the standard property or method supplied by Visual Basic and use the identically named property or method defined in the custom control.

For example, the **Tag** property is a property supplied to all custom controls in a Visual Basic project. If a custom control in a project has the name `ctlDemo`, and you access the **Tag** property using this syntax:

```
ctlDemo.Tag
```

Visual Basic automatically uses the **Tag** property it supplies. However, if the custom control defines its own **Tag** property and you want to access that property, use the **Object** property in this syntax:

```
ctlDemo.Object.Tag
```

Visual Basic automatically extends some or all of the following properties and methods to custom controls in a Visual Basic project:

**Properties**

| | | | |
|---|---|---|---|
| **Align** | **DragIcon** | **LinkMode** | **TabIndex** |
| **Cancel** | **DragMode** | **LinkItem** | **TabStop** |
| **Container** | **Enabled** | **LinkTimeout** | **Tag** |
| **DataChanged** | **Height** | **LinkTopic** | **Top** |
| **DataField** | **HelpContextID** | **Name** | **Visible** |
| **DataSource** | **Index** | **Negotiate** | **WhatsThisHelpID** |
| **Default** | **Left** | **Parent** | **Width** |

**Methods**

| | | |
|---|---|---|
| **Drag** | **LinkRequest** | **SetFocus** |
| **LinkExecute** | **LinkSend** | **ShowWhatsThis** |
| **LinkPoke** | **Move** | **ZOrder** |

If you use a property or method of a custom control and don't get the behavior you expect, see if the property or method has the same name as one of those shown in the preceding list. If the names match, check the documentation provided with the custom control to see if the behavior matches that of the property or method supplied by Visual Basic.   If the behaviors aren't identical, you may need to use the **Object** property to access the custom control feature you want.

## Container Property (Custom Controls)

Example

Returns or sets the container of a control on a **Form**.   Not available at design time.

### Syntax

**Set** *object*.**Container** [= *container*]

The **Container** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D command button, 3D frame, 3D group push button, 3D option button, 3D panel, Animated button, Communications, Graph, Gauge, Key state, ListView, Masked edit, Multimedia MCI, Outline, ProgressBar, RichTextBox, Slider, Spin button, SSTab, StatusBar, TabStrip, Toolbar, or TreeView control. |
| *container* | An object expression that evaluates to an object that can serve as a container for other controls, as described in Remarks. |

### Remarks

The following custom controls can contain other controls:

- **3D frame** control.
- **3D panel** control.

■

## Container Property (Custom Controls) Example

This example demonstrates moving a **CommandButton** control from container to container on a **Form** object.   To try this example, put these controls on the Form■**3D frame**, **3D panel**, and **CommandButton**

■then run the example.

```
Private Sub Form_Click()
   Static intX As Integer
   Select Case intX
      Case 0
         Set Command1.Container = SSPanel1
         Command1.Top= 0
         Command1.Left= 0
      Case 1
         Set Command1.Container = SSFrame1
         Command1.Top= 0
         Command1.Left= 0
      Case 2
         Set Command1.Container = Form1
         Command1.Top= 0
         Command1.Left= 0
   End Select
   intX = intX + 1
End Sub
```

## Negotiate Property (Custom Controls)

See Also

Sets a value that determines whether a control that can be aligned is displayed when an active object on the form displays one or more toolbars.   Not available at run time.

### Settings

The **Negotiate** property settings are:

| Setting | Description |
|---|---|
| **True** | If the control is aligned within the form (the **Align** property is set to a nonzero value), the control remains visible when an active object on the form displays a toolbar. |
| **False** | (Default) The control isn't displayed when an active object on the form displays a toolbar. The toolbar of the active object is displayed in place of the control. |

### Remarks

The **Negotiate** property exists for all controls with an **Align** property.

**See Also**

**Align** Property

## DataChanged Property (Custom Controls)

See Also

Returns or sets a value indicating that the data in the bound control has been changed by some process other than that of retrieving data from the current record. Not available at design time.

### Syntax

*object*.**DataChanged** [**=** *value*]

The **DataChanged** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D panel, Masked edit, or RichTextBox control. |
| *value* | A Boolean expression that indicates whether data has changed, as described in Settings. |

### Settings

The settings for *value* are:

| Setting | Description |
|---|---|
| **True** | The data currently in the control isn't the same as in the current record. |
| **False** | (Default) The data currently in the control, if any, is the same as the data in the current record. |

### Remarks

When a **Data** control moves from record to record, it passes data from fields in the current record to controls bound to the specific field or the entire record. As data is displayed in the bound controls, the **DataChanged** property is set to **False**. If the user or any other operation changes the value in the bound control, the **DataChanged** property is set to **True**. Simply moving to another record doesn't affect the **DataChanged** property.

When the **Data** control starts to move to a different record, the Validate event occurs. If **DataChanged** is **True** for any bound control, the **Data** control automatically invokes the **Edit** and **Update** methods to post the changes to the database.

If you don't wish to save changes from a bound control to the database, you can set the **DataChanged** property to **False** in the Validate event.

Inspect the value of the **DataChanged** property in your code for a control's Change event to avoid a cascading event. This applies to both bound and unbound controls.

**See Also**

**DataField** Property

**DataSource** Property

## DataField Property (Custom Controls)

See Also

Returns or sets a value that binds a control to a field in the current record.

### Syntax

*object*.**DataField** [**=** *value*]

The **DataField** property syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a 3D check box, 3D panel, Masked edit, or RichTextBox control. |
| *value* | A string expression that evaluates to the name of one of the fields in the **Recordset** object specified by a **Data** control's **RecordSource** and **DatabaseName** properties. |

### Remarks

Bound controls provide access to specific data in your database. Bound controls that manage a single field typically display the value of a specific field in the current record. The **DataSource** property of a bound control specifies a valid **Data** control name, and the **DataField** property specifies a valid field name in the **Recordset** object created by the **Data** control. Together, these properties specify what data appears in the bound control.

When you use a **QueryDef** object or SQL statement that returns the results of an expression, the field name is automatically generated by the Microsoft Jet database engine. For example, when you code an SQL aggregate function or an expression in your SQL query, unless you alias the aggregate fields using an AS clause, the field names are automatically generated. Generally, the expression field name is Expr1 followed by a three-character number starting with 000. The first expression returned would be named Expr1000.

It's recommended that you code your SQL queries to alias expression columns as shown below:

```
Data1.RecordSource = "Select AVG(Sales)  " _
   & " AS AverageSales From SalesTable"
MyText.DataField = "AverageSales"
MyText.DataSource = Data1
Data1.Refresh
```

**Note** Make sure the **DataField** property setting is valid for each bound control. If you change the setting of a **Data** control's **RecordSource** property and then use **Refresh**, the **Recordset** identifies the new object. This may invalidate the **DataField** settings of bound controls and produce a trappable error.

**See Also**

**DataChanged** Property
**DataSource** Property

## DataSource Property (Custom Controls)

See Also

Sets a value that specifies the **Data** control through which the current control is bound to a database. Not available at run time.

The **DataSource** property applies to the 3D check box, 3D panel, Masked edit, and RichTextBox controls.

### Remarks

To bind a control to a field in a database at run time, you must specify a **Data** control in the **DataSource** property at design time using the Properties window.

To complete the connection with a field in the **Recordset** managed by the **Data** control, you must also provide the name of a **Field** object in the **DataField** property. Unlike the **DataField** property, the **DataSource** property setting isn't available at run time.

**See Also**

**DataChanged** Property

**DataField** Property

## Alignment Property, 3DControls

Sets or returns the alignment of text in a 3D check box, 3D frame, 3D option button, or 3D panel control.

**Syntax**

[*form*.]*Object*.**Alignment**[ = *setting%*]

**Remarks**

For the 3D check box control, the Alignment property settings are:

| Setting | Description |
|---|---|
| 0 | (Default) Caption appears to the right of the check box. |
| 1 | Caption appears to the left of the check box. |

For the 3D frame control, the Alignment property settings are:

| Setting | Description |
|---|---|
| 0 | (Default) Caption appears left-justified within the top bar. |
| 1 | Caption appears right-justified within the top bar. |
| 2 | Caption appears centered within the top bar. |

For the 3D option button control, the Alignment property settings are:

| Setting | Description |
|---|---|
| 0 | (Default) Caption appears to the right of the option button. |
| 1 | Caption appears to the left of the option button. |

For the 3D panel control, the Alignment property settings are:

| Setting | Description |
|---|---|
| 0 | Caption appears left-justified at the top of the panel. |
| 1 | Caption appears left-justified in the middle of the panel. |
| 2 | Caption appears left-justified at the bottom of the panel. |
| 3 | Caption appears right-justified at the top of the panel. |
| 4 | Caption appears right-justified in the middle of the panel. |
| 5 | Caption appears right-justified at the bottom of the panel. |
| 6 | Caption appears centered at the top of the panel. |
| 7 | (Default) Caption appears centered in the middle of the panel. |
| 8 | Caption appears centered at the bottom of the panel. |

**Data Type**

**Integer** (Enumerated)

## AutoSize Property, 3D Controls

Determines how a control is sized to its picture or other contents. Applies to the 3D command button, 3D group push button, and 3D panel controls.

**Syntax**

[*form*.]*Object*.**AutoSize**[ = *setting%*]

**Remarks**

For the 3D command button control, the AutoSize property settings are:

| Setting | Description |
|---|---|
| 0 | (Default) No automatic sizing takes place. |
| 1 | Adjusts the picture size to the command button. This setting will shrink the picture to fit the size of the button. This option has no effect if the picture is an icon or if there is a caption specified for the command button. |
| 2 | Adjusts the command button size to the picture. This setting will resize the button to exactly fit the size of the picture. This option has no effect if there is a caption specified for the command button. |

For the 3D group push button control, the AutoSize property settings are:

| Setting | Description |
|---|---|
| 0 | No automatic sizing takes place. |
| 1 | Adjusts the picture size to the command button. This will stretch or shrink the bitmap to fit the size of the button. |
| 2 | (Default) Adjusts the button size to the picture. This will resize the button to exactly fit the size of the picture. |

For the 3D panel control, the AutoSize property settings are:

| Setting | Description |
|---|---|
| 0 | (Default) No automatic sizing takes place. |
| 1 | AutoSize panel width sized to caption. This setting adjusts the width of the panel to fit the caption within its inner bevel. The panel height remains unchanged. With this setting, the caption is displayed as a single line, regardless of its length. |
| 2 | AutoSize panel height sized to caption. This setting adjusts the height of the panel to fit the caption within its inner bevel. The panel width remains unchanged. With this setting, the caption may be displayed on multiple lines if it does not fit within the current width of the panel. |
| 3 | AutoSize child sized to panel. If a single control has been placed on the panel, this setting resizes the child control to fit exactly within the panel's inner bevel. This setting has no effect if there are no child controls, more than one child control, or if the panel has no bevels. This setting gives a three-dimensional look to standard controls such as list boxes and scroll bars. Note that if the child control has a fixed dimension (that is, the height of a combo box or drive box is fixed), that dimension of the panel is adjusted to fit it instead. |

**Data Type**

**Integer** (Enumerated)

## BevelWidth Property, 3D Controls

Sets or returns the height, width, or three-dimensional shadow effect of the bevel for the 3D command button, 3D group push button, and 3D panel controls.

### Syntax

[*form*.]*Object*.**BevelWidth**[ = *width%*]

### Remarks

For the 3D command button control, this property determines the number of pixels used to draw the bevel that surrounds the command button.   The bevel width can be set to a value between 0 and 10, inclusive.

For the 3D group push button, this property determines the height of the three-dimensional shadow effect setting the number of pixels used to draw the bevel that surrounds the button.   The bevel width can be set to a value between 0 and 2, inclusive.

For the 3D panel control, this property determines the number of pixels used to draw the inner and outer bevels that surround the panel.   Bevel width can be set to a value between 0 and 30, inclusive.   Use this property in conjunction with the BevelInner, BevelOuter, and BorderWidth properties.

### Data Type

**Integer**

## Click Event, 3D Controls

Occurs when the user presses and then releases a mouse button over a control. You can trigger the Click event in code by setting the control's Value property to **True**. Applies to the 3D checkbox, 3D group push button, 3D Option button, and 3D panel controls.

### Syntax

**Private Sub** *Object*_**Click** (*Value* **As Integer**)

### Remarks

This is the same as the standard Visual Basic Click event, except that the control's Value is passed as an argument. When the user selects the control, or when it is in the down position, Value = **True**. When the user does not select the control, or when it is in the up position, Value = **False**.

For the 3D checkbox control, the Click event is also generated when you change the **Value** property. For example, the following code will generate the Click event every time the form is clicked:

```
Private Sub Form_Click()
   ' The Click event will be generated whenever the Value
   ' property changes.
   SSCheck1.Value = Abs(SSCheck1.Value)-1
End Sub
```

## Outline Property, 3D Controls

Determines whether the control is displayed with a 1-pixel black border around its outer edge.   Applies to the 3D command button, 3D group push button, and 3D panel controls.

**Syntax**

[*form*.]*Object*.**Outline[ =** {**True | False**}**]**

**Remarks**

The following table lists the Outline property settings.

| Setting | Description |
|---|---|
| **True** | (Default) A 1-pixel black border is drawn around the control. |
| **False** | No border is drawn. |

**Data Type**

**Integer** (Boolean)

## Rounded Corners Property, 3D Controls

Determines whether the control is displayed with rounded corners.   Applies to the 3D command button, 3D group push button, and 3D panel controls.

**Syntax**

[*form*.]*Object*.**RoundedCorners[ =** {**True | False**}**]**

**Remarks**

The following table lists the RoundedCorners property settings.

| Setting | Description |
|---|---|
| **True** | (Default) The button's outline appears rounded (the four corner pixels are not drawn). |
| **False** | The button's outline appears square. |

**Note** This property has no effect when the Outline property is **False**.

**Data Type**

**Integer** (Boolean)

## ShadowColor Property, 3D Controls

Sets or returns the color used to draw the dark shading lines that make up the control. Applies to the 3D frame and 3D panel controls.

**Syntax**

[*form*.]*Object*.**ShadowColor[ =** *setting*%**]**

**Remarks**

The following table lists the ShadowColor property settings.

| Setting | Description |
|---|---|
| 0 | (Default) Dark gray |
| 1 | Black |

The dark gray setting looks good in most situations. If you would like the control to have a crisper look, or if you want to be consistent with another ShadowColor property setting the same form, choose setting 1 (black).

**Data Type**

**Integer** (Enumerated)

## Font3D Property, 3D Controls

Sets or returns the three-dimensional style of a 3D check box, 3D command button, 3D frame, 3D option button, or 3D panel control.

**Syntax**

[*form*.]*Object*.**Font3D**[ = *setting%*]

**Remarks**

The following table lists the Font3D property settings for the 3D controls.

| Setting | Description |
|---|---|
| 0 | (Default) No shading.   Caption is displayed flat (not three-dimensional). |
| 1 | Raised with light shading.   Caption appears raised off the screen. |
| 2 | Raised with heavy shading.   Caption appears more raised. |
| 3 | Inset with light shading.   Caption appears inset on the screen. |
| 4 | Inset with heavy shading.   Caption appears more inset. |

The Font3D property works with all the other Font properties.   Settings 2 and 4 (heavy shading) look best with larger, bolder fonts.

**Data Type**

**Integer** (Enumerated)