


Custom Control Help

See Also [Other InformationSources](#)

When you add a custom control to your project, its icon is displayed in the Toolbox. You can select the custom control by clicking this icon.

The toolbox icons for the custom controls are listed in the following table.

	3D check box		MAPI session
	3D command button		MAPI message
	3D frame		Masked edit
	3D group push button		Multimedia MCI
	3D option button		Outline
	3D panel		Pen BEdit
	Animated button		Pen HEdit
	Communications		Pen ink-on-bitmap
	Gauge		Pen on-screen keyboard
	Graph		Picture clip
	Key status		Spin button

See Also

[Object Type](#)

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

[The VB.LIC File](#)

[Using Custom Controls with Visual C++](#)

Object Type

In Visual Basic, a controls 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 controls object type, search Help for the **If** keyword.

In Visual C++, you can return the controls object class name, or object type, by using **CVBControl::GetVBXClass**. Refer to the *Class Library Reference* for more information.

The object type, or class name, for each control in the Visual Control Pack is listed in the following table. (In Visual Basic, the object type for a control also appears in the Properties window.)

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
Gauge	Gauge
Graph	Graph
Key status	MhState
MAPI	MapiSession, MapiMessages
Masked edit	MaskedTextBox
Multimedia MCI	MMControl
Outline	Outline
Pen edit	VBedit, VHedit
Pen ink-on-bitmap	InkOnBitmap
Pen on-screen keyboard	SKBButton
Picture clip	PictureClip
Spin button	SpinButton

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

To run your application under Microsoft Windows outside Visual Basic or Visual C++, create an executable (.EXE) file. You create executable files for applications that use custom controls the same as you would for any other application. There are a few issues to consider, however, when running such an application.

See Also

[Visual Basic Executable \(.EXE\) Files](#)

[Visual C++ 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 .VBX file associated with it must be on your systems 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 cannot be found, the Visual Basic run-time DLL generates the error message dialog box, File Not Found. If you want to distribute an application that uses custom controls, it is recommended that your installation procedure copy all required .VBX files into the users 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 (VBRUN300.DLL).
- Any .VBX files.
- Additional DLLs as required by your application or by custom controls.

Visual C++ Executable (.EXE) Files

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

If a custom control cannot be found, no error is generated and the application may no longer behave correctly. You can, however, add error handling code to ensure that a .VBX file loads correctly. The following code is from VBCIRCLE.CPP, which is part of the VBCIRCLE sample application:

```
// Check to see that the VBX control is present
if (LoadVBXFile("Circ3.vbx") > HINSTANCE_ERROR)
    UnloadVBXFile("Circ3.vbx");
else
{
    AfxMessageBox("Cannot Load Circ3.VBX\nPlace the file on the path", MB_OK |
    MB_ICONEXCLAMATION);
    return FALSE;
}
```

If you want to distribute an application that uses custom controls, it is recommended that your installation procedure copy all required .VBX files into the users Microsoft Windows \ SYSTEM directory.

You can freely distribute any application you create with Visual Basic custom controls to any Microsoft Windows user. Users will need copies of the following:

- The Microsoft Foundation Class (MFC) run-time file (MFC200.DLL), if the application does not statically link in the MFC200 library.
- Any .VBX files.
- Additional DLLs as required by your application or by custom controls.

Required Custom Control Files

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

Control	Required files
3D check box	THREED.VBX
3D command button	THREED.VBX
3D frame	THREED.VBX
3D group push button	THREED.VBX
3D option button	THREED.VBX
3D panel	THREED.VBX
Animated button	ANIBUTON.VBX
Communications	MSCOMM.VBX
Gauge	GAUGE.VBX
Graph	GRAPH.VBX, GSW.EXE, GSWDLL.DLL
Key status	KEYSTAT.VBX
MAPI ¹	MSMAPI.VBX
Masked edit	MSMASKED.VBX
Multimedia MCI ²	MCI.VBX
Outline	MSOUTLIN.VBX
Pen edit ³	PENCTRLS.VBX
Pen ink-on-bitmap ³	PENCTRLS.VBX
Pen on-screen keyboard ³	PENCTRLS.VBX
Picture clip	PICCLIP.VBX
Spin button	SPIN.VBX

¹ MAPI-compliant electronic mail system required

² Multimedia PC required

³ Windows for Pen Computing required

The VB.LIC File

When you install the Professional Edition, the design-time license file, VB.LIC, is installed in your Microsoft Windows \SYSTEM subdirectory. The VB.LIC file allows you to use the Professional Edition custom controls at design time in both Visual Basic and Visual C++. The Professional Edition VB.LIC file is downward compatible with any previously installed VB.LIC files.

Note You are *not* allowed to include VB.LIC with any application that you develop and distribute.

Other Information Sources

All help and text files included with the Standard Edition are also included with the Professional Edition.

Standard Edition

[Help Files](#)

[Text Files](#)

Professional Edition

[Help Files](#)

[Text Files](#)

Note When Visual Basic is installed, as each Help file is installed, it is listed in WINHELP.INI, located in your Windows directory. If you move a Help file to a different directory, be sure to change the path in WINHELP.INI. If a Help file does not exist, is not in your path, or is not in the directory specified in WINHELP.INI, WinHelp displays an appropriate message.

Standard Edition Help Files

You can click on any help file to go to the main table of contents of that file. If the file is not available, an error occurs.

Visual Basic Help—Documents Visual Basic for Windows.

SetupWizard—Documents the SetupWizard application. For information about the Setup Toolkit, search for Setup in the Visual Basic help file.

Data Manager—Documents the Data Manager application.

Note When Visual Basic is installed, as each Help file is installed, it is listed in WINHELP.INI, located in your Windows directory. If you move a Help file to a different directory, be sure to change the path in WINHELP.INI. If a Help file does not exist, is not in your path, or is not in the directory specified in WINHELP.INI, WinHelp displays an appropriate message.

Professional Edition Help Files

In addition to the help files provided with the Standard Edition, these help files are included with the Professional Edition. You can click on any help file to go to the main table of contents of that file. If the file is not available, an error occurs.

Crystal Reports—Documents the Crystal Reports application.

Custom Control Reference—Documents each of the custom controls provided with the Professional Edition.

Help Compiler Reference—Documents the Windows Help application for Help writers and programmers.

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

KnowledgeBase—A collection of articles from Microsoft Technical Support with tips, ideas and solutions.

ODBC Installation Help—Documents the installation tools for ODBC.

Oracle ODBC Driver—Documents the ODBC driver for Oracle databases.

SQL Server ODBC Driver—Documents the ODBC driver for SQL Server databases.

Visual Basic API Reference—Documents the Custom Control Development Kit.

Windows 3.1 API for Visual Basic—Declarations, structures, and constants for the Windows API as used in Visual Basic.

Windows 3.1 SDK Help—Documents Windows functions as used in the C programming language.

Note When Visual Basic is installed, as each Help file is installed, it is listed in WINHELP.INI, located in your Windows directory. If you move a Help file to a different directory, be sure to change the path in WINHELP.INI. If a Help file does not exist, is not in your path, or is not in the directory specified in WINHELP.INI, WinHelp displays an appropriate message.

Standard Edition Text Files

You can click on any of these files to launch Notepad and load the file.

README.TXT—Information on last minute changes to Visual Basic, as well as additional information.

CONSTANT.TXT—Global symbolic constants for Visual Basic properties, events, functions and statements.

DATACONS.TXT—Global symbolic constants for the data access features of Visual Basic.

EXTERNAL.TXT—Additional README information about connecting to external databases.

PACKING.LST—List of all files on the distribution disks provided with Visual Basic.

Professional Edition Text Files

In addition to the text files provided with the Standard Edition, these text files are included with the Professional Edition. You can click on any file to launch Notepad and load that file. If the file is too large for Notepad, you may have to use a different word processor.

BTRIEVE.TXT – Supplementary information on importing, exporting, or attaching Btrieve tables with Visual Basic.

ORACLE.TXT – Setup information for the ODBC Oracle driver to run with your ORACLE RDBMS software. If you installed ODBC, this file is in your Windows\SYSTEM directory.

PERFORM.TXT – Performance tuning tips for Visual Basic version 3.0 and Microsoft Access (TM) Relational Database System for Windows version 1.1.

SAMPLES.TXT – List of applications written in Visual Basic that demonstrate techniques discussed in the printed documentation.

WIN30API.TXT – Global symbolic constants for Windows 3.0 API functions.

WIN31API.TXT – Global symbolic constants for Windows 3.1 API functions.

WINMMSYS.TXT – Type declarations and global symbolic constants for Windows 3.1 multimedia API functions.



3D Check Box Control

[Properties](#)

[Methods](#)

[Events](#)

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

THREED.VBX

Object Type

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&). In Visual C++, you should call the **`CWinApp::SetDialogBkColor`** member function in your application's **`CWinApp::InitInstance`** member function. For more information, see the Visual C++ *Class Library Reference*.

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 20, *Accessing Databases With the Data Control*, in the *Programmers Guide*.

Distribution Note When you create and distribute applications that use the 3D check box control, you should install the file `THREED.VBX` 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 (*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*Alignment</u>	<u>*Font3D</u>	Height	TabIndex
Caption	FontBold	HelpContextID	TabStop
DataChanged	FontItalic	hWnd	Tag
DataField	FontName	Index	Top
DataSource	FontSize	Left	Value
DragIcon	FontStrikethru	MousePointer	Visible
DragMode	FontUnderline	Name	Width
Enabled	ForeColor	Parent	

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 and Visual C++.

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 (*). For documentation of the remaining events, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*Click</u>	DragOver	KeyDown	KeyUp
DragDrop	GotFocus	KeyPress	LostFocus

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	SetFocus	

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

Alignment Property, 3D Check Box Control

Description

Sets or returns the alignment of text in the check box.

Visual Basic

```
[form.]CheckBox3d.Alignment[ = setting%]
```

Visual C++

```
pCheckBox3d->GetNumProperty("Alignment")  
pCheckBox3d->SetNumProperty("Alignment", setting)
```

Remarks

The following table lists the Alignment property settings for the 3D check box.

Setting	Description
0	(Default) Caption appears to the right of the check box.
1	Caption appears to the left of the check box.

Data Type

Integer (Enumerated)

Font3D Property, 3D Check Box Control

Description

Sets or returns the three-dimensional style of the check box caption.

Visual Basic

[*form.*]CheckBox3d.**Font3D**[= *setting%*]

Visual C++

pCheckBox3d->**GetNumProperty**("Font3D")
pCheckBox3d->**SetNumProperty**("Font3D", *setting*)

Remarks

The following table lists the Font3D property settings for the 3D check box.

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)

Click Event, 3D Check Box Control

Description

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**.

Visual Basic

Sub *CheckBox3d_Click* (*Value As Integer*)

Visual C++

Function Signature:

void *CMyDialog::OnClickCheckBox3d* (**UINT, int, CWnd*, LPVOID lpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM(short, lpParams)

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 check box, *Value* = **True**. When the user does not select it, *Value* = **False**.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.



3D Command Button Control

[Properties](#)

[Methods](#)

[Events](#)

[Error Messages](#)

Description

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

THREED.VBX

Object Type

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&). In Visual C++, you should call the **`CWinApp::SetDialogBkColor`** member function in your application's **`CWinApp::InitInstance`** member function. For more information, see the Visual C++ *Class Library Reference*.

Distribution Note When you create and distribute applications that use the 3D command button control, you should install the file THREED.VBX 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 (*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*AutoSize</u>	FontName	Left	Tag
<u>*BevelWidth</u>	FontSize	MousePointer	Top
Caption	FontStrikethru	Name	Value
DragIcon	FontUnderline	<u>*Outline</u>	Visible
DragMode	ForeColor	Parent	Width
Enabled	Height	<u>*Picture</u>	
<u>*Font3D</u>	HelpContextID	<u>*RoundedCorner</u>	
		<u>S</u>	
FontBold	hWnd	TabIndex	
FontItalic	Index	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 and Visual C++.

Events

All of the events for this control are listed in the following table. For documentation of these events, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

Click	DragOver	KeyDown	KeyUp
DragDrop	GotFocus	KeyPress	LostFocus

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	SetFocus	

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

AutoSize Property, 3D Command Button Control

Description

Determines whether the command button is automatically sized to its picture or the picture is sized to the button.

Visual Basic

```
[form.]CommandButton3d.AutoSize [ = setting%]
```

Visual C++

```
pCommandButton3d->GetNumProperty("AutoSize")  
pCommandButton3d->SetNumProperty("AutoSize", setting)
```

Remarks

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

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.

Data Type

Integer (Enumerated)

BevelWidth Property, 3D Command Button Control

Description

Sets or returns the width of the bevel along the four sides of the command button to determine the height of the three-dimensional shadow effect.

Visual Basic

```
[form.]CommandButton3d.BevelWidth[ = width%]
```

Visual C++

```
pCommandButton3d->GetNumProperty("BevelWidth")  
pCommandButton3d->SetNumProperty("BevelWidth", width)
```

Remarks

The setting for 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.

Data Type

Integer

Font3D Property, 3D Command Button Control

Description

Sets or returns the three-dimensional style of the command button caption.

Visual Basic

[*form.*]CommandButton3d.**Font3D**[= *setting%*]

Visual C++

pCommandButton3d->**GetNumProperty**("Font3D")
pCommandButton3d->**SetNumProperty**("Font3D", *setting*)

Remarks

The following table lists the Font3D property settings for the 3D command button control.

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)

Outline Property, 3D Command Button Control

Description

Determines whether the command button is displayed with a 1-pixel black border around its outer edge.

Visual Basic

```
[form.]CommandButton3d.Outline[ = {True | False}]
```

Visual C++

```
pCommandButton3d->GetNumProperty("Outline")  
pCommandButton3d->SetNumProperty("Outline", {TRUE | FALSE})
```

Remarks

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

Setting	Description
True	(Default) A 1-pixel black border is drawn around the button.
False	No border is drawn around the button.

Data Type

Integer (Boolean)

Picture Property, 3D Command Button Control

Example

Description

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

Visual Basic

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

Visual C++

```
pCommandButton3d->GetPictureProperty("Picture")  
pCommandButton3d->SetPictureProperty("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 CF_BITMAP and CF_DIB, as defined in the CONSTANT.TXT file.

Visual C++ provides three functions that allow you to manipulate Picture property values. These functions are **AfxSetPict**, **AfxGetPict**, and **AfxReferencePict**. Refer to Technical Note 27: *Emulation Support for Visual Basic Custom Controls*, which you can access by selecting TN027 in the Visual C++ online help file, MFCNOTES.HLP.

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

Visual Basic Example

The following example pastes a bitmap from the Clipboard onto a command button. To find the value of the CF_ formats, look at the CONSTANT.TXT file, or load that file into the global module. 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.

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

RoundedCorners Property, 3D Command Button Control

Description

Determines whether the command button is displayed with rounded corners.

Visual Basic

```
[form.]CommandButton3d.RoundedCorners[ = {True | False}]
```

Visual C++

```
pCommandButton3d->GetNumProperty("RoundedCorners")  
pCommandButton3d->SetNumProperty("RoundedCorners", {TRUE | FALSE})
```

Remarks

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

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)

Error Messages, 3D Command Button Control

The following table lists the trappable errors for the 3D command button.

Error number	Message explanation
30000	Only picture formats '.BMP' & '.ICO' supported. This error results when 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. This error results when the bevel width is set to an invalid value.



3D Frame Control

[Properties](#)

[Methods](#)

[Events](#)

Description

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

THREED.VBX

Object Type

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&). In Visual C++, you should call the **`CWinApp::SetDialogBkColor`** member function in your application's **`CWinApp::InitInstance`** member function. For more information, see the Visual C++ *Class Library Reference*.

Distribution Note When you create and distribute applications that use the 3D frame control, you should install the file THREED.VBX 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 (*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

Align	FontName	Left	Visible
<u>*Alignment</u>	FontSize	MousePointer	Width
Caption	FontStrikethru	Name	
DragIcon	FontUnderline	Parent	
DragMode	ForeColor	<u>*ShadowColor</u>	
Enabled	Height	<u>*ShadowStyle</u>	
<u>*Font3D</u>	HelpContextID	TabIndex	
FontBold	hWnd	Tag	
FontItalic	Index	Top	

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 and Visual C++.

Events

All of the events for this control are listed in the following table. For documentation of these events, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

DragDrop DragOver

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	Move	Refresh	ZOrder
-------------	-------------	----------------	---------------

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

Alignment Property, 3D Frame Control

Description

Sets or returns the alignment of text in the frame.

Visual Basic

[*form.*]Frame3d.Alignment[= *setting%*]

Visual C++

pFrame3d->GetNumProperty("Alignment")

pFrame3d->SetNumProperty("Alignment", *setting*)

Remarks

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

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.

Data Type

Integer (Enumerated)

Font3D Property, 3D Frame Control

Description

Sets or returns the three-dimensional style of the frame caption.

Visual Basic

[*form.*]Frame3d.**Font3D**[= *setting*%]

Visual C++

pFrame3d->**GetNumProperty**("Font3D")
pFrame3d->**SetNumProperty**("Font3D", *setting*)

Remarks

The following table lists the Font3D property settings for the 3D frame control.

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 in conjunction with all the other Font properties. Settings 2 and 4 (heavy shading) look best with larger, bolder fonts.

Data Type

Integer (Enumerated)

ShadowColor Property, 3D Frame Control

Description

Sets or returns the color used to draw the dark shading lines that make up the frame.

Visual Basic

[form.]Frame3d.ShadowColor [= *setting%*]

Visual C++

pFrame3d->GetNumProperty("ShadowColor")

pFrame3d->SetNumProperty("ShadowColor", setting)

Remarks

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

Setting	Description
---------	-------------

0	(Default) Dark gray
---	---------------------

1	Black
---	-------

The dark gray setting looks good in most situations. If you would like the frame to have a crisper look, or if you want to be consistent with the ShadowColor property setting on a Panel that resides on the same form, choose setting 1 (black).

Data Type

Integer (Enumerated)

ShadowStyle Property, 3D Frame Control

Description

Determines whether the frame appears inset or raised.

Visual Basic

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

Visual C++

pFrame3d->GetNumProperty("ShadowStyle")

pFrame3d->SetNumProperty("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](#)

[Error Messages](#)

Description

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

THREED.VBX

Object Type

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 file THREED.VBX 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 (*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

Align	<u>*GroupNumber</u>	<u>*Outline</u>	Top
<u>*AutoSize</u>	Height	Parent	Value
BackColor	HelpContextID	<u>*PictureDisabled</u>	Visible
<u>*BevelWidth</u>	hWnd	<u>*PictureDn</u>	Width
DragIcon	Index	<u>*PictureDnChange</u>	
		<u>e</u>	
DragMode	Left	<u>*PictureUp</u>	
Enabled	MousePointer	<u>*RoundedCorner</u>	
		<u>S</u>	
<u>*GroupAllowAllUName</u>		Tag	
<u>p</u>			

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 and Visual C++.

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 (*). For documentation of the events not unique to this control, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

*Click DragDrop DragOver

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	Move	Refresh	ZOrder
-------------	-------------	----------------	---------------

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

AutoSize Property, 3D Group Push Button Control

Description

Determines whether the button is automatically sized to its picture or the picture is sized to the button.

Visual Basic

```
[form.]GroupPushButton.AutoSize[ = setting%]
```

Visual C++

```
pGroupPushButton->GetNumProperty("AutoSize")  
pGroupPushButton->SetNumProperty("AutoSize", setting)
```

Remarks

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

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.

Data Type

Integer (Enumerated)

BevelWidth Property, 3D Group Push Button Control

Description

Sets or returns the width of the bevel along the four sides of the 3D group push button.

Visual Basic

```
[form.]GroupPushButton.BevelWidth[ = width%]
```

Visual C++

```
pGroupPushButton->GetNumProperty("BevelWidth")  
pGroupPushButton->SetNumProperty("BevelWidth", width)
```

Remarks

BevelWidth determines the height of the three-dimensional shadow effect.

This property determines the number of pixels used to draw the bevel that surrounds the button. The valid range for BevelWidth is from 0 to 2.

Data Type

Integer

GroupAllowAllUp Property, 3D Group Push Button Control

Description

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

Visual Basic

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

Visual C++

```
pGroupPushButton->GetNumProperty("GroupAllowAllUp")  
pGroupPushButton->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

pGroupPushButton->**GetNumProperty**("GroupNumber")
pGroupPushButton->**SetNumProperty**("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. In Visual C++, all controls in a dialog or form view are siblings.

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

Outline Property, 3D Group Push Button Control

Description

Sets or returns a 1-pixel black border around the button's outer edge.

Visual Basic

```
[form.]GroupPushButton.Outline[ = {True | False}]
```

Visual C++

```
pGroupPushButton->GetNumProperty("Outline")
```

```
pGroupPushButton->SetNumProperty("Outline", {TRUE | FALSE})
```

Remarks

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

Setting	Description
True	(Default) A 1-pixel black border is drawn.
False	No border is drawn.

Data Type

Integer (Boolean)

PictureDisabled Property, 3D Group Push Button Control

Description

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

Visual Basic

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

Visual C++

```
pGroupPushButton->GetPictureProperty("PictureDisabled")  
pGroupPushButton->SetPictureProperty("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 CF_BITMAP and CF_DIB, as defined in CONSTANT.TXT, a file that specifies system defaults.

Visual C++ provides three functions that allow you to manipulate Picture property values. These functions are **AfxSetPict**, **AfxGetPict**, and **AfxReferencePict**. Refer to Technical Note 27: *Emulation Support for Visual Basic Custom Controls*, which you can access by selecting TN027 in the Visual C++ online Help file, MFCNOTES.HLP.

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

Description

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.

Visual Basic

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

Visual C++

```
pGroupPushButton->GetPictureProperty("PictureDn")  
pGroupPushButton->SetPictureProperty("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.

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 CF_BITMAP and CF_DIB as defined in the CONSTANT.TXT file.

Visual C++ provides three functions that allow you to manipulate Picture property values. These functions are **AfxSetPict**, **AfxGetPict**, and **AfxReferencePict**. Refer to Technical Note 27: *Emulation Support for Visual Basic Custom Controls*, which you can access by selecting TN027 in the Visual C++ online Help file, MFCNOTES.HLP.

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

Description

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

Visual Basic

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

Visual C++

```
pGroupPushButton->GetNumProperty("PictureDnChange")  
pGroupPushButton->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pGroupPushButton->GetPictureProperty("PictureUp")  
pGroupPushButton->SetPictureProperty("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.

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 CF_BITMAP and CF_DIB as defined in the CONSTANT.TXT file.

Visual C++ provides three functions that allow you to manipulate Picture property values. These functions are **AfxSetPict**, **AfxGetPict**, and **AfxReferencePict**. Refer to Technical Note 27: *Emulation Support for Visual Basic Custom Controls*, which you can access by selecting TN027 in the Visual C++ online Help file, MFCNOTES.HLP.

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

RoundedCorners Property, 3D Group Push Button Control

Description

Determines whether the 3D group push button is displayed with rounded corners.

Visual Basic

```
[form.]GroupPushButton.RoundedCorners = {True | False}
```

Visual C++

```
pGroupPushButton->GetNumProperty("RoundedCorners")
```

```
pGroupPushButton->SetNumProperty("RoundedCorners", {TRUE | FALSE})
```

Remarks

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

Setting	Description
True	(Default) The button's outline appears rounded (the four corner pixels are not drawn).
False	The button's outline appears square.

Data Type

Integer (Boolean)

Click Event, 3D Group Push Button Control

Description

Occurs when the user presses and then releases a mouse button over a 3D group push button. You can trigger the Click event for a group push button in code by setting the control's Value property to **True**.

Visual Basic

Sub *GroupPush3D_Click* (*Value As Integer*)

Visual C++

Function Signature:

void *CMyDialog::OnClickGroupPushButton* (**UINT, int, CWnd*, LPVOID lpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (**short, lpParams**)

Remarks

This event is the same as the standard Visual Basic Click event, except that the control's *Value* is passed as an argument. When the button is in the down position, *Value* = **True**. When it is in the up position, *Value* = **False**.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

Error Messages, 3D Group Push Button Control

The following table lists the trappable errors for the 3D group push button.

Error number	Message explanation
30001	Only picture format '.BMP' supported. This error results when 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. This error results when the GroupNumber property is set to an invalid value.
30007	Bevel width must be from 0 to 2. This error results when the bevel width is set to an invalid value.



3D Option Button Control

[Properties](#)

[Methods](#)

[Events](#)

Description

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

THREED.VBX

Object Type

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&). In Visual C++, you should call the **`CWinApp::SetDialogBkColor`** member function in your application's **`CWinApp::InitInstance`** member function. For more information, see the Visual C++ *Class Library Reference*.

Distribution Note When you create and distribute applications that use the 3D option button control, you should install the file THREED.VBX 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 (*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*Alignment</u>	FontName	Index	Top
Caption	FontSize	Left	Value
DragIcon	FontStrikethru	MousePointer	Visible
DragMode	FontUnderline	Name	Width
Enabled	ForeColor	Parent	
<u>*Font3D</u>	Height	TabIndex	
FontBold	HelpContextID	TabStop	
FontItalic	hWnd	Tag	

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 and Visual C++.

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 (*). For documentation of the remaining events, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*Click</u>	DragOver	KeyDown	KeyUp
<u>*DbClick</u>	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. 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	

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

Alignment Property, 3D Option Button Control

Description

Sets or returns the alignment of text in the option button.

Visual Basic

[*form.*]OptionButton3d.**Alignment**[= *setting%*]

Visual C++

pOptionButton3d->**GetNumProperty**("Alignment")
pOptionButton3d->**SetNumProperty**("Alignment", *setting*)

Remarks

The following table lists the Alignment property settings for the 3D option button control.

Setting	Description
0	(Default) Caption appears to the right of the option button.
1	Caption appears to the left of the option button.

Data Type

Integer (Enumerated)

Font3D Property, 3D Option Button Control

Description

Sets or returns the three-dimensional style of the option button caption.

Visual Basic

[*form.*]OptionButton3d.**Font3D**[= *setting*%]

Visual C++

pOptionButton3d->**GetNumProperty**("Font3D")

pOptionButton3d->**SetNumProperty**("Font3D", *setting*)

Remarks

The following table lists the Font3D property settings for the 3D option button control.

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)

Click Event, 3D Option Button Control

Description

Occurs when the user presses and then releases a mouse button over a 3D option button control. You can trigger the Click event in code by setting the control's Value property to **True**.

Visual Basic

Sub *OptionButton3d_Click* (*Value As Integer*)

Visual C++

Function Signature:

void *CMyDialog::OnClickOptionButton3d* (**UINT, int, CWnd*, LPVOID lpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (**short, lpParams**)

Remarks

This event is the same as the standard Visual Basic Click 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**.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

DbIcIck Event, 3D Option Button Control

Description

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

Visual Basic

Sub *OptionButton3d_DbIcIck* (*Value As Integer*)

Visual C++

Function Signature:

void *CMyDialog::OnDbIcIckOptionButton3d* (**UINT, int, CWND*, LPVOID lpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (**short, lpParams**)

Remarks

This event is the same as the standard Visual Basic DbIcIck 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**.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.



3D Panel Control

[Properties](#)

[Methods](#)

[Events](#)

[Error Messages](#)

Description

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

THREED.VBX

Object Type

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 20, Accessing Databases With the Data Control, in the *Programmers Guide*.

Distribution Note When you create and distribute applications that use the 3D panel control, you should install the file THREED.VBX 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 (*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*Alignment</u>	DragIcon	FontSize	<u>*Outline</u>
<u>*AutoSize</u>	DragMode	FontStrikethru	Parent
BackColor	Enabled	FontUnderline	<u>*RoundedCorners</u>
<u>*BevelInner</u>	<u>*FloodColor</u>	ForeColor	<u>*ShadowColor</u>
<u>*BevelOuter</u>	<u>*FloodPercent</u>	Height	TabIndex
<u>*BevelWidth</u>	<u>*FloodShowPct</u>	HelpContextID	Tag
<u>*BorderWidth</u>	<u>*FloodType</u>	hWnd	Top
Caption	<u>*Font3D</u>	Index	Visible
DataChanged	FontBold	Left	Width
DataField	FontItalic	MousePointer	
DataSource	FontName	Name	

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 and Visual C++.

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. For documentation of the events not unique to this control, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

DragDrop DragOver

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	Move	Refresh	ZOrder
-------------	-------------	----------------	---------------

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

Alignment Property, 3D Panel Control

Description

Sets or returns the alignment of text in the panel.

Visual Basic

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

Visual C++

pPanel3d->GetNumProperty("Alignment")
pPanel3d->SetNumProperty("Alignment", setting)

Remarks

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

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 Panel Control

Description

Determines whether the panel is automatically sized to its contents.

Visual Basic

```
[form.]Panel3d.AutoSize[ = setting%]
```

Visual C++

```
pPanel3d->GetNumProperty("AutoSize")  
pPanel3d->SetNumProperty("AutoSize", setting)
```

Remarks

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

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)

BevelInner Property, 3D Panel Control

Description

Determines the style of the inner bevel of the panel.

Visual Basic

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

Visual C++

```
pPanel3d->GetNumProperty("BevelInner")  
pPanel3d->SetNumProperty("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

Description

Determines the style of the outer bevel of the panel.

Visual Basic

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

Visual C++

```
pPanel3d->GetNumProperty("BevelOuter")  
pPanel3d->SetNumProperty("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)

BevelWidth Property, 3D Panel Control

Description

Sets or returns the width of the outer and inner bevels of the panel; determines the amount of the three-dimensional shadow effect.

Visual Basic

```
[form.]Panel3d.BevelWidth[ = width%]
```

Visual C++

```
pPanel3d->GetNumProperty("BevelWidth")  
pPanel3d->SetNumProperty("BevelWidth", width)
```

Remarks

The setting for 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

BorderWidth Property, 3D Panel Control

Description

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

Visual Basic

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

Visual C++

```
pPanel3d->GetNumProperty("BorderWidth")  
pPanel3d->SetNumProperty("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

Description

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).

Visual Basic

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

Visual C++

```
pPanel3d->GetNumProperty("FloodColor")  
pPanel3d->SetNumProperty("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. In Visual C++, standard RGB colors can be used.
System default colors	In Visual Basic, specified with system color constants from the CONSTANT.TXT file. In Visual C++, use the MAKESYSCOLOR macro defined in the AFXEXT.H file to create system color constants. Microsoft Windows substitutes the user's choices as specified through the user's control panel settings.

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

Description

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.

Visual Basic

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

Visual C++

```
pPanel3d->GetNumProperty("FloodPercent")  
pPanel3d->SetNumProperty("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.

```
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

Description

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).

Visual Basic

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

Visual C++

```
pPanel3d->GetNumProperty("FloodShowPct")  
pPanel3d->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pPanel3d->GetNumProperty("FloodType")  
pPanel3d->SetNumProperty("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)

Font3D Property, 3D Panel Control

Description

Sets or returns the three-dimensional style of the panel caption.

Visual Basic

[form.]Panel3d.Font3D[= setting%]

Visual C++

pPanel3d->GetNumProperty("Font3D")
pPanel3d->SetNumProperty("Font3D", setting)

Remarks

The following table lists the Font3D property settings for the 3D panel control.

Setting	Description
0	(Default) None. 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)

Outline Property, 3D Panel Control

Description

Determines whether the panel is displayed with a 1-pixel black border around its outer edge.

Visual Basic

```
[form.]Panel3d.Outline[ = {True | False}]
```

Visual C++

```
pPanel3d->GetNumProperty("Outline")  
pPanel3d->SetNumProperty("Outline", {TRUE | FALSE})
```

Remarks

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

Setting	Description
True	Draws a 1-pixel black border around the panel.
False	(Default) No border.

Data Type

Integer (Boolean)

RoundedCorners Property, 3D Panel Control

Description

Determines whether the panel is displayed with rounded corners.

Visual Basic

```
[form.]Panel3d.RoundedCorners = {True | False}
```

Visual C++

```
pPanel3d->GetNumProperty("RoundedCorners")  
pPanel3d->SetNumProperty("RoundedCorners", {TRUE | FALSE})
```

Remarks

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

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 The property has no effect when Outline is **False**.

Data Type

Integer (Boolean)

ShadowColor Property, 3D Panel Control

Description

Sets or returns the color used to draw the dark shading lines that make up the panel.

Visual Basic

```
[form.]Panel3d.ShadowColor[ = setting%]
```

Visual C++

```
pPanel3d->GetNumProperty("ShadowColor")  
pPanel3d->SetNumProperty("ShadowColor", setting)
```

Remarks

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

Setting	Description
---------	-------------

0	(Default) Dark gray.
---	----------------------

1	Black.
---	--------

The dark gray setting works well in most situations. If you want the panel to have a crisper look, or if you want to be consistent with the ShadowColor property setting on a frame on the same form, choose setting 1 (black).

If the BackColor property is set to a color other than the default light gray, setting ShadowColor to black sometimes looks better than dark gray.

Data Type

Integer (Enumerated)

Error Messages, 3D Panel Control

The following table lists the trappable errors for the 3D panel control.

Error number	Message explanation
30002	Bevel width must be from 0 to 30. This error results if the BevelWidth property is set to an invalid value.
30003	Border width must be from 0 to 30. This error results if the BorderWidth property is set to an invalid value.
30006	Flood percent must be from 0 to 100. This error results if the FloodPercent property is set to an invalid value.



Animated Button Control

[Properties](#)

[Methods](#)

[Events](#)

[Error Messages](#)

Description

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

ANIBUTTON.VBX

Object Type

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 file ANIBUTTON.VBX 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). 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 (*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

BackColor	FontItalic	Left	Tag
BorderStyle	FontName	MousePointer	<u>*TextPosition</u>
Caption	FontSize	Name	<u>*TextXpos</u>
<u>*CCBfileLoad</u>	FontStrikethru	Parent	<u>*TextYpos</u>
<u>*CCBfileSave</u>	FontUnderline	<u>*PictDrawMode</u>	Top
<u>*ClearFirst</u>	ForeColor	<u>*Picture</u>	<u>*Value</u>
<u>*ClickFilter</u>	<u>*Frame</u>	<u>*PictureXpos</u>	Visible
<u>*Cycle</u>	Height	<u>*PictureYpos</u>	Width
DragIcon	HelpContextID	<u>*SpecialOp</u>	
DragMode	hWnd	<u>*Speed</u>	
Enabled	<u>*HideFocusBox</u>	TabIndex	
FontBold	Index	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 and Visual C++.

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 (*). documentation of the remaining events, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*Click</u>	DragOver	KeyDown	KeyUp
DragDrop	GotFocus	KeyPress	LostFocus

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	SetFocus	

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

CCBfileLoad Property, Animated Button Control

Description

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

Visual Basic

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

Visual C++

```
pAniButton->SetStrProperty("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

Description

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

Visual Basic

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

Visual C++

```
pAniButton->SetStrProperty("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

Description

Determines whether the control is cleared between frames.

Visual Basic

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

Visual C++

```
pAniButton->GetNumProperty("ClearFirst")  
pAniButton->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

pAniButton->GetNumProperty("ClickFilter")
pAniButton->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pAniButton->GetNumProperty("Cycle")  
pAniButton->SetNumProperty("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 property Value 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

Description

Indicates the current frame.

Visual Basic

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

Visual C++

pAniButton->GetNumProperty("Frame")
pAniButton->SetNumProperty("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 to 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 Example, Animated Button Control

Visual Basic Example

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

```
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

Description

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**.

Visual Basic

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

Visual C++

```
pAniButton->GetNumProperty("HideFocusBox")
```

```
pAniButton->SetNumProperty("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

Description

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.

Visual Basic

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

Visual C++

```
pAniButton->GetNumProperty("PictDrawMode")  
pAniButton->SetNumProperty("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

Description

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, in Visual Basic:

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

And, in Visual C++:

```
HPIC hPict = pAniButton1->GetPictureProperty ("Picture"); pAniButton2->SetPictureProperty ("Picture",hPict);.
```

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

PictureXpos Property, Animated Button Control

Description

Controls the horizontal placement of the image in the control.

Visual Basic

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

Visual C++

```
pAniButton->GetNumProperty("PictureXpos")  
pAniButton->SetNumProperty("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

Description

Controls the vertical placement of the image in the control.

Visual Basic

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

Visual C++

```
pAniButton->GetNumProperty("PictureYpos")  
pAniButton->SetNumProperty("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

Description

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.

Visual Basic

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

Visual C++

```
pAniButton->SetNumProperty("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

Description

Specifies the approximate delay, in milliseconds, between frames.

Visual Basic

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

Visual C++

```
pAniButton->GetNumProperty("Speed")  
pAniButton->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pAniButton->GetNumProperty("TextPosition")  
pAniButton->SetNumProperty("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

Description

Controls the horizontal placement of the text caption.

Visual Basic

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

Visual C++

pAniButton->**GetNumProperty**("TextXpos")
pAniButton->**SetNumProperty**("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

Description

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

Visual Basic

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

Visual C++

pAniButton->**GetNumProperty**("TextYpos")
pAniButton->**SetNumProperty**("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

Description

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. This property is not available at design time.

Visual Basic

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

Visual C++

pAniButton->**GetNumProperty**("Value")
pAniButton->**SetNumProperty**("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 of a control is changed, the display may not be updated until subsequent events have occurred (such as the **DoEvents()** function in Visual Basic or the **Yield()** function in Visual C++).

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

Data Type

Integer (Enumerated)

Click Event, Animated Button Control

Description

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

Visual Basic

Sub *AniButton_Click* (*Value As Integer*)

Visual C++

Function Signature:

void *CMyDialog::OnClickAniButton* (**UINT, int, CWnd*, LPVOID lpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (**short, lpParams**)

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.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

Error Messages, Animated Button Control

The following table lists the trappable errors for the animated button control.

Error number	Message explanation
321	Invalid file format. This error occurs when attempting to load an invalid CCB file. The animated button can load CCB files created by the Custom Control Factory; however, these files may use features that are unique to the Custom Control Factory and are not supported by the animated button control. In this case, the unsupported features are ignored, and substitute property values are set as needed.
361	Can't load or unload this object. This error indicates that there is insufficient memory available to load or unload the specified CCB file or control . Loading an animated button control into a control array may cause this error to occur (especially if the controls contain many frames).
380	Invalid property value. Attempting to set a property to an invalid value causes this error.



Communications Control

[Properties](#)

[Events](#)

[Functions](#)

[Example](#)

Description

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

File Name

MSCOMM.VBX

Object Type

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.

The actual values of the event and error constants are defined in the Visual Basic CONSTANT.TXT file or the CONSTANT.H file included with the Visual Control Pack.

Distribution Note When you create and distribute applications that use the communications control, you should install the file MSCOMM.VBX 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

Visual Basic Example

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

```
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 (*). For documentation on the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*Break</u>	<u>*DSR Holding</u>	<u>*InputLen</u>	<u>*ParityReplace</u>
<u>*CD Holding</u>	<u>*DSR Timeout</u>	<u>*Interval</u>	<u>*PortOpen</u>
<u>*CD Timeout</u>	<u>*DTR Enable</u>	Left	<u>*RThreshold</u>
<u>*CommEvent</u>	<u>*Handshaking</u>	Name	<u>*RTSEnable</u>
<u>*CommID</u>	<u>*InBufferCount</u>	<u>*NullDiscard</u>	<u>*Settings</u>
<u>*CommPort</u>	<u>*InBufferSize</u>	<u>*OutBufferCount</u>	<u>*SThreshold</u>
<u>*CTSHolding</u>	Index	<u>*OutBufferSize</u>	Tag
<u>*CTSTimeout</u>	<u>*Input</u>	<u>*Output</u>	Top

Input is the default value of the control.

Note The Index property is only available in Visual Basic. The Name property is the same as the CtlName property in Visual Basic 1.0 and Visual C++.

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

Description

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

Visual Basic

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

Visual C++

pMComm->GetNumProperty("Break")

pMComm->SetNumProperty("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)

Break Example, Communications Control

Visual Basic Example

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

Description

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.

Visual Basic

[*form.*]MSComm.CDHolding

Visual C++

pMSComm->GetNumProperty("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 MSCOMM_ER_CDTO (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

Description

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 MSCOMM_ER_CDTO (Carrier Detect Timeout Error) and generating the OnComm event.

Visual Basic

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

Visual C++

```
pMSComm->GetNumProperty("CDTimeout")  
pMSComm->SetNumProperty("CDTimeout", milliseconds)
```

Remarks

When the Carrier Detect line is low (CDHolding = **False**) and CDTimeout number of milliseconds has passed, the communications control sets the CommEvent property to MSCOMM_ER_CDTO (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

Description

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

Visual Basic

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

Visual C++

pMSComm->**GetNumProperty("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 Visual Basic CONSTANT.TXT file or the CONSTANT.H file included with the Visual Control Pack.

Communications errors include the following settings.

Setting	Description
MSCOMM_ER_BREAK	A Break signal was received.
MSCOMM_ER_CDTO	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).
MSCOMM_ER_CTSTO	Clear To Send Timeout. The Clear To Send line was low for CTSTimeout number of milliseconds while trying to transmit a character.
MSCOMM_ER_DSRTO	Data Set Ready Timeout. The Data Set Ready line was low for DSRTIMEOUT number of milliseconds while trying to transmit a character.
MSCOMM_ER_FRAME	Framing Error. The hardware detected a framing error.
MSCOMM_ER_OVERRUN	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.
MSCOMM_ER_RXOVER	Receive Buffer Overflow. There is no room in the receive buffer.
MSCOMM_ER_RXPARITY	Parity Error. The hardware detected a parity error.
MSCOMM_ER_TXFULL	Transmit Buffer Full. The transmit buffer was full while trying to queue a character.

Communications events include the following settings.

Setting	Description
MSCOMM_EV_CD	Change in Carrier Detect line.
MSCOMM_EV_CTS	Change in Clear To Send line.
MSCOMM_EV_DSR	Change in Data Set Ready line. This event is only fired when DSR changes from -1 to 0.
MSCOMM_EV_EOF	End Of File (ASCII character 26) character received.
MSCOMM_EV_RING	Ring detected. Some UARTs (universal asynchronous receiver-transmitter) may not support this event.
MSCOMM_EV_RECEIVE	Received RThreshold number of characters. This event is generated continuously until you use the Input property to remove the data from the receive

buffer.

MSCOMM_EV_SEND There are fewer than SThreshold number of characters in the transmit buffer.

Data Type

Integer

CommID Property, Communications Control

Description

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

Visual Basic

[form.]MSComm.CommID

Visual C++

pMSComm->GetNumProperty("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

Description

Sets and returns the communications port number.

Visual Basic

```
[form.]MSCComm.CommPort[ = portNumber%]
```

Visual C++

```
pMSCComm->GetNumProperty("CommPort")  
pMSCComm->SetNumProperty("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

Description

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.

Visual Basic

[*form.*]MSComm.CTSHolding

Visual C++

pMSComm->GetNumProperty("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 high (CTSHolding = **True**) and the CTSTimeout number of milliseconds has passed, the communications control sets the CommEvent property to MSCOMM_ER_CTSTO (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

Description

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

Visual Basic

```
[form.]MComm.CTSTimeout[ = milliseconds&]
```

Visual C++

```
pMComm->GetNumProperty("CTSTimeout")  
pMComm->SetNumProperty("CTSTimeout", milliseconds)
```

Remarks

When the Clear To Send line is high (CTSHolding = **True**) and the CTSTimeout number of milliseconds has passed, the communications control sets the CommEvent property to MSCOMM_ER_CTSTO (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

DSR Holding Property, Communications Control

Description

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 and is read-only at run time.

Visual Basic

[*form.*]MSComm.DSRHolding

Visual C++

pMSComm->GetNumProperty("DSRHolding")

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 MSCOMM_ER_DSRTIMEOUT (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

Description

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

Visual Basic

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

Visual C++

```
pMSComm->GetNumProperty("DSRTimeout")  
pMSComm->SetNumProperty("DSRTimeout", milliseconds)
```

Remarks

When the Data Set Ready line is high (DSR Holding = **True**) and the DSRTimeout number of milliseconds has passed, the communications control sets the CommEvent property to MSCOMM_ER_DSRTO (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

Description

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.

Visual Basic

```
[form.]MComm.DTREnable[ = {True | False}]
```

Visual C++

```
pMComm->GetNumProperty("DTREnable")  
pMComm->SetNumProperty("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

Description

Sets and returns the hardware handshaking protocol.

Visual Basic

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

Visual C++

pMSComm->**GetNumProperty**("Handshaking")
pMSComm->**SetNumProperty**("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	Description
MSCOMM_HANDSHAKE_NONE	(Default) No handshaking.
MSCOMM_HANDSHAKE_XONXOFF	XON/XOFF handshaking.
MSCOMM_HANDSHAKE_RTS	RTS/CTS (Request To Send/Clear To Send) handshaking.
MSCOMM_HANDSHAKE_RTSEXONXOFF	Both Request To Send and XON/XOFF handshaking.

Data Type

Integer

InBufferCount Property, Communications Control

Description

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

Visual Basic

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

Visual C++

```
pMSComm->GetNumProperty("InBufferCount")  
pMSComm->SetNumProperty("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

Description

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

Visual Basic

```
[form.]MComm.InBufferSize[ = numBytes%]
```

Visual C++

```
pMComm->GetNumProperty("InBufferSize")  
pMComm->SetNumProperty("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 occur, increase the buffer size to handle your application's transmission rate.

Data Type

Integer

Input Property, Communications Control

Example

Description

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.

Visual Basic

[*form*.]MSComm.Input

Visual C++

pMSComm->GetStrProperty("Input")

Remarks

The Visual C++ syntax works for strings that do not contain Null characters, and strings containing Null characters will be truncated. For strings that contain Null characters, use either the **ComInput** function or the following syntax:

```
#define UM_INPUT WM_USER + 0x0B00  
pMSComm->SendMessage(UM_INPUT, lpData, cbData)
```

lpData points to a data buffer to hold the input data, and *cbData* is the size of the buffer, in bytes. This message will return the actual number of bytes of user input.

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

Visual Basic Example

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

Description

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

Visual Basic

```
[form.]MComm.InputLen[ = numChars%]
```

Visual C++

```
pMComm->GetNumProperty("InputLen")  
pMComm->SetNumProperty("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

Visual Basic Example

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

Description

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

Visual Basic

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

Visual C++

```
pMSComm->GetNumProperty("Interval")  
pMSComm->SetNumProperty("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

Description

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

Visual Basic

```
[form.]MComm.NullDiscard[ = {True | False}]
```

Visual C++

```
pMComm->GetNumProperty("NullDiscard")  
pMComm->SetNumProperty("NullDiscard", {TRUE | FALSE})
```

Remarks

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

Setting	Description
True	Null characters are <i>not</i> 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

Description

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.

Visual Basic

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

Visual C++

```
pMSComm->GetNumProperty("OutBufferCount")  
pMSComm->SetNumProperty("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

Description

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

Visual Basic

```
[form.]MComm.OutBufferSize[ = NumBytes%]
```

Visual C++

```
pMComm->GetNumProperty("OutBufferSize")  
pMComm->SetNumProperty("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 Note that 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

Description

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

Visual Basic

```
[form.]MSComm.Output[ = outString]
```

Visual C++

```
pMSComm->SetStrProperty("Output", outString)
```

Remarks

The Visual C++ syntax will work for strings that do not contain Null characters. For strings containing Null characters, use either the **ComOutput** function or the following syntax:

```
#define UM_OUTPUT WM_USER + 0x0B01
```

```
pMSComm->SendMessage(UM_OUTPUT, lpData, cbData)
```

lpData points to the string being sent, and *cbData* is the length of the string, in bytes. The return value from this message is the actual number of bytes sent.

Data Type

String

Output Example, Communications Control

Visual Basic Example

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

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

ParityReplace Property, Communications Control

Description

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

Visual Basic

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

Visual C++

```
pMSComm->GetStrProperty("ParityReplace")  
pMSComm->SetStrProperty("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 parity checking.

Data Type

String

PortOpen Property, Communications Control

Example

Description

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

Visual Basic

```
[form.]MComm.PortOpen[ = {True | False}]
```

Visual C++

```
pMComm->GetNumProperty("PortOpen")  
pMComm->SetNumProperty("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)

PortOpen Example, Communications Control

Visual Basic Example

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

Description

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

Visual Basic

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

Visual C++

```
pMSComm->GetNumProperty("RThreshold")  
pMSComm->SetNumProperty("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

Description

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.

Visual Basic

```
[form.]MComm.RTSEnable[ = {True | False}]
```

Visual C++

```
pMComm->GetNumProperty("RTSEnable")  
pMComm->SetNumProperty("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

Description

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

Visual Basic

```
[form.]MComm.Settings[ = paramString$]
```

Visual C++

```
pMComm->GetStrProperty("Settings")  
pMComm->SetStrProperty("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 described 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

Settings Example, Communications Control

Visual Basic Example

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

Description

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

Visual Basic

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

Visual C++

pMSComm->**GetNumProperty**("SThreshold")
pMSComm->**SetNumProperty**("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 MSCOMM_EV_SEND, and the OnComm event is generated. The MSCOMM_EV_SEND event is only fired once, when the number of characters crosses the SThreshold. For example, if SThreshold equals five, the MSCOMM_EV_SEND 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

Description

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

Visual Basic

Sub *MSComm_OnComm* (**)**

Visual C++

Function Signature:

void *CMyDialog::OnOnCommMSComm* (**UINT, int, CWnd*, LPVOID**)

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 MSCOMM_EV_RECEIVE and MSCOMM_EV_SEND events, respectively.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

OnComm Event Example, Communications Control

Visual Basic Example

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.

```
Sub Comm_OnComm ()
    Select Case Comm1.CommEvent
        ' Errors
        Case MSCOMM_ER_BREAK      ' A Break was received.
            ' Code to handle a BREAK goes here.
        Case MSCOMM_ER_CDTO      ' CD (RLSD) Timeout.
        Case MSCOMM_ER_CTSTO     ' CTS Timeout.
        Case MSCOMM_ER_DSRTO     ' DSR Timeout.
        Case MSCOMM_ER_FRAME     ' Framing Error
        Case MSCOMM_ER_OVERRUN   ' Data Lost.
        Case MSCOMM_ER_RXOVER    ' Receive buffer overflow.
        Case MSCOMM_ER_RXPARITY  ' Parity Error.
        Case MSCOMM_ER_TXFULL    ' Transmit buffer full.
        ' Events
        Case MSCOMM_EV_CD        ' Change in the CD line.
        Case MSCOMM_EV_CTS      ' Change in the CTS line.
        Case MSCOMM_EV_DSR      ' Change in the DSR line.
        Case MSCOMM_EV_RING     ' Change in the Ring Indicator.
        Case MSCOMM_EV_RECEIVE  ' Received RThreshold # of chars.
        Case MSCOMM_EV_SEND     ' There are SThreshold number of
                                ' characters in the transmit buffer.

    End Select
End Sub
```

ComInput Function, Communications Control

Description

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

Visual Basic

ComInput(By Val *hWnd* As Integer, *lpData* As Any, By Val *cbData* As Integer) As Integer

Visual C++

int ComInput(*hWnd*, *lpData*, *cbData*)

Parameter	Type	Description
<i>hWnd</i>	HWND	Window handle of the control.
<i>lpData</i>	LPSTR	Long pointer to the start of the data buffer.
<i>cbData</i>	int	The length of <i>lpData</i> in bytes.

Remarks

This function is equivalent to the Input property.

In Visual Basic 1.0 and Visual C++, 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

Description

Writes a string of characters to the transmit buffer.

Visual Basic

ComOutput(By Val *hWnd* As Integer, *lpData* As Any, By Val *cbData* As Integer) As Integer

Visual C++

int ComOutput(*hWnd*, *lpData*, *cbData*)

Parameter	Type	Description
<i>hWnd</i>	HWND	Window handle of the control.
<i>lpData</i>	LPSTR	Long pointer to the start of the data buffer.
<i>cbData</i>	int	The length of <i>lpData</i> in bytes.

Remarks

This function is equivalent to the Output property.

In Visual Basic 1.0 and Visual C++, 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](#)

Description

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

File Name

GAUGE.VBX

Object Type

Gauge

Remarks

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

Several bitmaps that can be used with the gauge control are included with the Visual Control Pack.

Note In Visual Basic, 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 file GAUGE.VBX 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 (*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

AutoSize	Index	Name	<u>*Value</u>
<u>*BackColor</u>	<u>*InnerBottom</u>	<u>*NeedleWidth</u>	Visible
DragIcon	<u>*InnerLeft</u>	Parent	<u>*Width</u>
DragMode	<u>*InnerRight</u>	*Picture	
Enabled	<u>*InnerTop</u>	<u>*Style</u>	
<u>*ForeColor</u>	Left	TabIndex	
<u>*Height</u>	<u>*Max</u>	TabStop	
HelpContextID	<u>*Min</u>	Tag	
hWnd	MousePointer	Top	

Value is the default value of the control.

Note The DragIcon, DragMode, HelpContextID, Index, and Parent properties are only available in Visual Basic. Name is the equivalent of the CtlName property in Visual Basic 1.0 and Visual C++.

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 (*). For documentation of the remaining events, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*Change</u>	DragOver	KeyUp	MouseUp
Click	GotFocus	LostFocus	
DbtClick	KeyDown	MouseDown	
DragDrop	KeyPress	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. For documentation of the following methods, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

Drag	Refresh	ZOrder
Move	SetFocus	

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

BackColor Property, Gauge Control

Description

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

Visual Basic

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

Visual C++

```
pGauge->GetNumProperty("BackColor")  
pGauge->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pGauge->GetNumProperty("ForeColor")  
pGauge->SetNumProperty("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

Description

Determine the height and width of the gauge control.

Visual Basic

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

Visual C++

```
pGauge->GetNumProperty("Height")  
pGauge->SetNumProperty("Height", setting)  
pGauge->GetNumProperty("Width")  
pGauge->SetNumProperty("Width", setting)
```

Remarks

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

Data Type

Integer

InnerBottom Property, Gauge Control

Description

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

Visual Basic

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

Visual C++

```
pGauge->GetNumProperty("InnerBottom")  
pGauge->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

pGauge->GetNumProperty("InnerLeft")
pGauge->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pGauge->GetNumProperty("InnerRight")  
pGauge->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

pGauge->**GetNumProperty**("InnerTop")
pGauge->**SetNumProperty**("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

Description

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

Visual Basic

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

Visual C++

```
pGauge->GetNumProperty("Max")  
pGauge->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pGauge->GetNumProperty("Min")  
pGauge->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pGauge->GetNumProperty("NeedleWidth")  
pGauge->SetNumProperty("NeedleWidth", width)
```

Data Type

Integer

Picture Property, Gauge Control

Description

Specifies a bitmap to display on the gauge.

Visual Basic

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

Visual C++

```
pGauge->GetPictureProperty("Picture")  
pGauge->SetPictureProperty("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.

In Visual Basic, 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 CF_BITMAP and CF_DIB, as defined in the CONSTANT.TXT file. 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.

Visual C++ provides three functions that allow you to manipulate Picture property values. These functions are **AfxSetPict**, **AfxGetPict**, and **AfxReferencePict**. Refer to Technical Note 27: *Emulation Support for Visual Basic Custom Controls*, which you can access by selecting TN027 in the Visual C++ online help file, MFCNOTES.HLP.

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

Description

Sets or returns the type of gauge.

Visual Basic

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

Visual C++

pGauge->GetNumProperty("Style")
pGauge->SetNumProperty("Style", *setting*)

Remarks

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

Setting	Description
0	(Default) Horizontal linear 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 *Innerportion* 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 *Innerportion* 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

Description

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

Visual Basic

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

Visual C++

```
pGauge->GetNumProperty("Value")  
pGauge->SetNumProperty("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

Description

Occurs when the control's Value property changes.

Visual Basic

Sub *Gauge_Change* ()

Visual C++

Function Signature:

void *CMyDialog::OnChangeGauge* (**UINT, int, CWnd*, LPVOID**)

Remarks

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.



Graph Control

[See Also](#)

[Properties](#)

[Methods](#)

[Events](#)

Description

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

GRAPH.VBX

Object Type

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 files GRAPH.VBX, GSWDLL.DLL, and GSW.EXE 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 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

Visual Basic Example

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 Visual Basic code 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 (*). For documentation of all other properties, see Appendix A of the *Custom Control Reference*.

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

QuickData is the default value of the control.

Note The DragIcon, DragMode, HelpContextID, and Index properties are only available in Visual Basic. Name is equivalent to the CtlName property in Visual Basic 1.0 and Visual C++.

Events

All of the events for this control are listed in the following table. For documentation on the following events, see Appendix A, "Standard Properties, Events, and Methods," of the *Custom Control Reference*.

Click	DragOver	KeyPress	MouseDown
DbClick	GotFocus	KeyUp	MouseMove
DragDrop	KeyDown	LostFocus	MouseUp

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 on the methods that are not unique to this control, see Appendix A, "Standard Properties, Events, and Methods," of the *Custom Control Reference*.

Drag	PrintForm	SetFocus
Hide	Refresh	ZOrder

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

AutoInc Property, Graph Control

Example

Description

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.

Visual Basic

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

Visual C++

pGraph->**GetNumProperty("AutoInc")**

pGraph->**SetNumProperty("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

Visual Basic Example

```
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

Description

Selects the background color of the graph.

Visual Basic

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

Visual C++

pGraph->GetNumProperty("Background")
pGraph->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pGraph->GetStrProperty("BottomTitle")  
pGraph->SetStrProperty("BottomTitle", string)
```

Remarks

This property is ignored for Pie charts.

Data Type

String

BottomTitle Example, Graph Control

Visual Basic Example

The following code places the title, "Title," at the bottom of a graph (Graph2) 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.

```
Sub Command1_Click ()
    Graph2.RandomData = 1
    If Graph2.BottomTitle = "" Then
        Graph2.BottomTitle = "Title"
    Else
        Graph2.BottomTitle = ""
    End If
    Graph2.DrawMode = 2
End Sub
```

ColorData Property, Graph Control

Description

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.

Visual Basic

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

Visual C++

```
pGraph->GetNumProperty("ColorData")  
pGraph->SetNumProperty("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

Description

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

Visual Basic

*[form.]*Graph.CtlVersion

Visual C++

*pGraph->*GetStrProperty("CtlVersion")

Data Type

String

DataReset Property, Graph Control

Description

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

Visual Basic

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

Visual C++

```
pGraph->GetNumProperty("DataReset")  
pGraph->SetNumProperty("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

Description

Defines the drawing mode for the graph control.

Visual Basic

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

Visual C++

pGraph->GetNumProperty("DrawMode")
pGraph->SetNumProperty("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. DrawModes 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

Description

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.

Visual Basic

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

Visual C++

pGraph->GetNumProperty("DrawStyle")
pGraph->SetNumProperty("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

Description

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.

Visual Basic

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

Visual C++

pGraph->GetNumProperty("ExtraData")
pGraph->SetNumProperty("ExtraData", *setting*)

Remarks

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

Setting	Description
0	(Default) Not exploded
1	Exploded

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

Visual Basic Example

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).

```
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

Description

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

Visual Basic

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

Visual C++

pGraph->GetNumProperty("FontFamily")
pGraph->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pGraph->GetNumProperty("FontSize")  
pGraph->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

pGraph->GetNumProperty("FontStyle")

pGraph->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pGraph->GetNumProperty("FontUse")  
pGraph->SetNumProperty("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 underline.

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

Description

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

Visual Basic

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

Visual C++

pGraph->**GetNumProperty**("Foreground")

pGraph->**SetNumProperty**("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

Description

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

Visual Basic

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

Visual C++

```
pGraph->GetStrProperty("GraphCaption")  
pGraph->SetStrProperty("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

Visual Basic Example

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

Description

Sets the data to be graphed.

Visual Basic

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

Visual C++

```
pGraph->GetFloatProperty("GraphData")  
pGraph->SetFloatProperty("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

Visual Basic Example

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).

```
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

Description

Sets the characteristics of each type of graph.

Visual Basic

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

Visual C++

pGraph->GetNumProperty("GraphStyle")
pGraph->SetNumProperty("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	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 the each data set is a different color.
	1 Horizontal	
	2 Stacked	
	3 Horizontal stacked	
	4 Stacked %	
	5 Horizontal stacked %	
3D bar	As preceding, plus:	Z-clustered means that the data points for successive sets are drawn in front of the previous one. This gives an illusion of depth.
	6 Z-clustered	
	7 Horizontal Z-clustered	
Gantt	0 (Default) Adjacent bars	Spaced bars have a gap of one bar's width between successive bars.
	1 Spaced bars	
Line, Log/Lin, and polar	0 (Default) Lines	You can create thick or patterned lines by setting the ThickLine or PatternLine
	1 Symbols	
	2 Sticks	

	3	Sticks and symbols	property to 1 (on).
	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	Scatter graphs require XPosData to be present.
		only	
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

Description

Places a text string above the graph.

Visual Basic

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

Visual C++

pGraph->**GetStrProperty**("GraphTitle")
pGraph->**SetStrProperty**("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

Visual Basic Example

The following code places the title, "Title," at the top of a graph (Graph2) 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.

```
Sub Command1_Click ()
    Graph2.RandomData = 1
    If Graph2.GraphTitle = "" Then
        Graph2.GraphTitle = "Title"
    Else
        Graph2.GraphTitle = ""
    End If
    Graph2.DrawMode = 2
End Sub
```

GraphType Property, Graph Control

Description

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

Visual Basic

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

Visual C++

```
pGraph->GetNumProperty("GraphType")  
pGraph->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

pGraph->GetNumProperty("GridStyle")

pGraph->SetNumProperty("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

Description

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.

Visual Basic

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

Visual C++

```
pGraph->GetStrProperty("ImageFile")  
pGraph->SetStrProperty("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](#)

Description

Sets the data array index style.

Visual Basic

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

Visual C++

```
pGraph->GetNumProperty("IndexStyle")  
pGraph->SetNumProperty("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 Property Example 1, Graph Control

Visual Basic Example

```
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

Visual Basic Example

```
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

Description

Determines the frequency of labels displayed on the X axis.

Visual Basic

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

Visual C++

```
pGraph->GetNumProperty("LabelEvery")  
pGraph->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

pGraph->GetNumProperty("Labels")
pGraph->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pGraph->GetStrProperty("LabelText")  
pGraph->SetStrProperty("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

Description

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

Visual Basic

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

Visual C++

```
pGraph->GetStrProperty("LeftTitle")  
pGraph->SetStrProperty("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

Visual Basic Example

The following code places the title, "Title," to the left of the vertical axis of a graph (Graph2) 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.

```
Sub Command1_Click ()
    If Graph2.LeftTitle = "" Then
        Graph2.LeftTitle = "Title"
    Else
        Graph2.LeftTitle = ""
    End If
    Graph2.DrawMode = 2
End Sub
```

LegendStyle Property, Graph Control

Description

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

Visual Basic

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

Visual C++

```
pGraph->GetNumProperty("LegendStyle")  
pGraph->SetNumProperty("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

Description

Allows you to enter text for legends.

Visual Basic

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

Visual C++

```
pGraph->GetStrProperty("LegendText")  
pGraph->SetStrProperty("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

Description

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

Visual Basic

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

Visual C++

pGraph->GetNumProperty("LineStats")
pGraph->SetNumProperty("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

Description

Specifies the number of data points in each data set.

Visual Basic

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

Visual C++

```
pGraph->GetNumProperty("NumPoints")  
pGraph->SetNumProperty("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

Description

Specifies the number of data sets to be graphed.

Visual Basic

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

Visual C++

pGraph->GetNumProperty("NumSets")
pGraph->SetNumProperty("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

Description

Allows you to select a specific set of palette colors.

Visual Basic

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

Visual C++

pGraph->GetNumProperty("Palette ")
pGraph->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pGraph->GetNumProperty("PatternData")  
pGraph->SetNumProperty("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

Description

Sets the style of the lines connecting the data points.

Visual Basic

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

Visual C++

```
pGraph->GetNumProperty("PatternedLines")  
pGraph->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

*pGraph->***GetPictureProperty("Picture")**

Data Type

Integer

Picture Example, Graph Control

Visual Basic Example

The following Visual Basic 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

Description

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

Visual Basic

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

Visual C++

pGraph->GetNumProperty("PrintStyle")
pGraph->SetNumProperty("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

Description

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

Visual Basic

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

Visual C++

```
pGraph->GetStrProperty("QuickData")  
pGraph->SetStrProperty("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 in Visual Basic, 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)). To create the string in Visual C++, separate each point in the data set with a tab character ('\t') and each data set by a CR+LF ("\r\n").

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

Visual Basic Example

```
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

Description

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.

Visual Basic

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

Visual C++

pGraph->GetNumProperty("RandomData")
pGraph->SetNumProperty("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

Description

If you set the SeeThru property to 1 (on), the graph background is not cleared. Instead, whatever was there before the 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.

Visual Basic

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

Visual C++

pGraph->**GetNumProperty**("SeeThru")
pGraph->**SetNumProperty**("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

Visual Basic Example

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
```

```
Sub Form_Load ()  
    Flag = 0  
    Graph1.SeeThru = 1  
End Sub
```

```
Sub Picture1_Paint ()  
    If Flag = 1 Then  
        Flag = 0  
        Picture1.Refresh  
        Graph1.Refresh  
    Else  
        Flag = 1  
    End If  
End Sub
```

SymbolData Property, Graph Control

Description

Selects symbols to be used for line, log/lin, scatter, and polar graphs.

Visual Basic

```
[form.]Graph.SymbolData[ = symbol%]
```

Visual C++

```
pGraph->GetNumProperty("SymbolData")  
pGraph->SetNumProperty("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

Description

Sets the width of the lines. For illustrations, see the *Custom Control Reference*.

Visual Basic

```
[form.]Graph.ThickLines[ = setting%]
```

Visual C++

```
pGraph->GetNumProperty("ThickLines")  
pGraph->SetNumProperty("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

Description

Sets the current point number manually so that a particular data point can be changed.

Visual Basic

```
[form.]Graph.ThisPoint[ = point%]
```

Visual C++

```
pGraph->GetNumProperty("ThisPoint")  
pGraph->SetNumProperty("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

Visual Basic Example

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).

```
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

Description

Allows you to manually control the current set number so that a particular data set can be changed.

Visual Basic

```
[form.]Graph.ThisSet[ = set%]
```

Visual C++

```
pGraph->GetNumProperty("ThisSet")  
pGraph->SetNumProperty("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

Visual Basic Example

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).

```
Sub Form_Load ()
    Graph1.NumPoints = 5
    Graph1.NumSets = 3
    Graph1.AutoInc = 1
    For I% = 1 To Graph2.NumPoints * Graph2.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

Description

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.

Visual Basic

```
[form.]Graph.TickEvery[ = interval%]
```

Visual C++

```
pGraph->GetNumProperty("TickEvery")  
pGraph->SetNumProperty("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

Description

Determines whether axis ticks are displayed.

Visual Basic

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

Visual C++

pGraph->GetNumProperty("Ticks")
pGraph->SetNumProperty("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

Description

Gives an independent X value for a graph.

Visual Basic

[*form.*]Graph.XPosData[= *xvalue!*]

Visual C++

pGraph->**GetFloatProperty**("XPosData")
pGraph->**SetFloatProperty**("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

Visual Basic Example

```
Graph2.AutoInc = 0
Graph2.NumPoints = 10
Graph2.NumSets = 2
For I% = 1 To 2
    Graph2.ThisSet = I%
    For J% = 1 To 10
        Graph2.ThisPoint = J%
        If I% = 1 Then Graph2.GraphData = 5 - J%
        If I% = 2 Then Graph2.GraphData = J% - 5
        Graph2.XPosData = J% / I%
    Next J%
Next I%
Graph2.DrawMode = 2
```

YAxisMax, YAxisMin Properties

Description

Specifies the maximum Y-axis value (YAxisMax) and minimum Y-axis value (YAxisMin) on your graph.

Visual Basic

```
[form.]Graph.YAxisMax[ = max!]  
[form.]Graph.YAxisMin[ = min!]
```

Visual C++

```
pGraph->GetFloatProperty("YAxisMax")  
pGraph->SetFloatProperty("YAxisMax", max)  
pGraph->GetFloatProperty("YAxisMin")  
pGraph->SetFloatProperty("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 effect when YAxisStyle = 2 (user-defined). For more information, see the YAxisStyle property.

Data Type

Single

YAxisPos Property, Graph Control

Description

Specifies the position of the Y axis on your graph.

Visual Basic

```
[form.]Graph.YAxisPos[ = position%]
```

Visual C++

```
pGraph->GetNumProperty("YAxisPos")  
pGraph->SetNumProperty("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

Description

Specifies the method used to scale and range the Y axis on your graph.

Visual Basic

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

Visual C++

pGraph->GetNumProperty("YAxisStyle")
pGraph->SetNumProperty("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 affects.

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

Description

Specifies the number of ticks on the Y axis of your graph.

Visual Basic

[form.]Graph.YAxisTicks [= *ticks%*]

Visual C++

pGraph->GetNumProperty("YAxisTicks")
pGraph->SetNumProperty("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



Key Status Control

[Properties](#)

[Methods](#)

[Events](#)

Description

You can use the key status control to display or modify the CAPS LOCK, NUM LOCK, INS and SCROLL LOCK keyboard states.

File Name

KEYSTAT.VBX

Object Type

mhState

Remarks

Key status 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 status 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 status control, you should install the file KEYSTAT.VBX 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(*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

AutoSize	Index	TabIndex	Visible
BackColor	Left	TabStop	<u>*Width</u>
Enabled	MousePointer	Tag	
<u>*Height</u>	Name	<u>*TimerInterval</u>	
HelpContextID	Parent	Top	
hWnd	<u>*Style</u>	<u>*Value</u>	

Value is the default value of the control.

Note The HelpContextID, Index, and Parent properties are only available in Visual Basic. The Name property is equivalent to the CtlName property in Visual Basic 1.0 and Visual C++.

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(*). For documentation of the remaining events, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*Change</u>	GotFocus	KeyPress	LostFocus
Click	KeyDown	KeyUp	

Methods

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

Move	Refresh	SetFocus	ZOrder
-------------	----------------	-----------------	---------------

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

Height, Width Properties, Key Status Control

Description

Determine the height and width of the key status control.

Visual Basic

```
[form.]KeyStatus.Height[ = setting%]  
[form.]KeyStatus.Width[ = setting%]
```

Visual C++

```
pKeyStatus->GetNumProperty("Height")  
pKeyStatus->SetNumProperty("Height", setting)  
pKeyStatus->GetNumProperty("Width")  
pKeyStatus->SetNumProperty("Width", setting)
```

Remarks

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

Data Type

Integer

Style Property, Key Status Control

Description

Determines which keyboard state is associated with the key status control.

Visual Basic

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

Visual C++

pKeyStatus->GetNumProperty("Style")

pKeyStatus->SetNumProperty("Style", *setting*)

Remarks

The following table lists the Style property settings for the key status control.

Setting	Description
0	(Default) Capitals lock
1	Number lock
2	Insert state
3	Scroll lock

Data Type

Integer (Enumerated)

TimerInterval Property, Key Status Control

Description

Sets or returns the current timer interval setting for all key status controls. The default is 1000 milliseconds.

Visual Basic

```
[form.]KeyStatus.TimerInterval[ = milliseconds%]
```

Visual C++

```
pKeyStatus->GetNumProperty("TimerInterval")  
pKeyStatus->SetNumProperty("TimerInterval", milliseconds)
```

Remarks

This property determines the interval at which the key status is checked. If you are having performance problems, try setting TimerInterval to a higher value.

Only one timer operates all key status 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 Status Control

Description

Sets or returns the current status 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.

Visual Basic

```
[form.]KeyStatus.Value[ = {True | False}]
```

Visual C++

```
pKeyStatus->GetNumProperty("Value")  
pKeyStatus->SetNumProperty("Value", {TRUE | FALSE})
```

Remarks

The following table lists the Value property settings for the key status control.

Setting	Description
False	Key status is off (for example, Caps Lock is off).
True	Key status is on (for example, Caps Lock is on).

Data Type

Integer (Boolean)

Change Event, Key Status Control

Description

Occurs when the Value property changes.

Visual Basic

Sub *KeyStatus_Change* ()

Visual C++

Function Signature:

void *CMyDialog::OnChangeKeyStatus* (**UINT, int CWnd*, LPVOID**)

Remarks

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," of the *Custom Control Reference*.

■ MAPI

[Properties](#)

[Error Messages](#)

Description

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 set the appropriate Action property value.

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.

File Name

MSMAPI.VBX

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 (*). They are documented in the following sections. (Note that the list order is alphabetic from top to bottom, then left to right.) See the Visual Basic *Language Reference* or Help for documentation of the remaining properties.

About	<u>*LogonUI</u>	Tag
<u>*Action</u>	Name	Top
<u>*DownloadMail</u>	<u>*NewSession</u>	<u>*UserName</u>
Index	<u>*Password</u>	
Left	<u>*SessionID</u>	

Action Property

Description

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.

Usage

[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:

<u>Setting</u>	<u>Description</u>
SESSION_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. Depending on the value of the NewSession property, the session handle may refer to a newly created session or an existing session.
SESSION_SIGNOFF	Ends the messaging session and signs the user off the specified account.

Data Type

Integer (Enumerated)

DownloadMail Property

Description

Specifies whether new messages are downloaded from the mail server for the designated user.

Usage

[*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 <i>not</i> 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

Description

Specifies whether or not a dialog box is provided for sign-on.

Usage

[*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

Description

Specifies whether a new mail session should be established, even if a valid session currently exists.

Usage

[*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

Description

Specifies the account password associated with the UserName property.

Usage

[*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

Description

Stores the current messaging session handle. This property is not available at design time, and is read only at run time.

Usage

[form.]MapiSession.SessionID

Remarks

This property is set when you assign SESSION_SIGNON to the Action property. The SessionID property contains the unique and 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

Description

Specifies the account user name.

Usage

[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

▪ MAPI messages

[Properties](#)

[Error Messages](#)

Object Type

MapiMessages

Description

The MAPI messages control performs a variety of messaging system functions after a messaging session is established with the MAPI session control. The MAPI messages control is shown here, as it appears as an icon in the Toolbox, and on a form at design time.



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 RecipiIndex 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 MESSAGE_FETCH setting of the Action property. 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 CONSTANT.TXT file 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 (*). They are documented in the following sections. (Note that the list order is alphabetic from top to bottom, then left to right.) See the Visual Basic *Language Reference* or Help for documentation of the remaining properties.

About	<u>*FetchMsgType</u>	<u>*MsgReceiptRequested</u>
<u>*Action</u>	<u>*FetchSorted</u>	<u>*MsgSent</u>
<u>*AddressCaption</u>	<u>*FetchUnreadOnly</u>	<u>*MsgSubject</u>
<u>*AddressEditFieldCo</u> <u>unt</u>	Index	<u>*MsgType</u>
<u>*AddressLabel</u>	<u>*MsgConversationID</u>	Name
<u>*AddressModifiable</u>	<u>*MsgCount</u>	<u>*RecipAddress</u>
<u>*AddressResolveUI</u>	<u>*MsgDateReceived</u>	<u>*RecipCount</u>
<u>*AttachmentCount</u>	<u>*MsgID</u>	<u>*RecipDisplayName</u>
<u>*AttachmentIndex</u>	<u>*MsgIndex</u>	<u>*RecipIndex</u>
<u>*AttachmentName</u>	<u>*MsgNoteText</u>	<u>*RecipType</u>
<u>*AttachmentPathNa</u> <u>me</u>	<u>*MsgOrigAddress</u>	<u>*SessionID</u>
<u>*AttachmentPositio</u> <u>n</u>	<u>*MsgOrigDisplayName</u>	Tag
<u>*AttachmentType</u>	<u>*MsgRead</u>	Top

Action Property

Description

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.

Usage

[*form.*]MapiMessages.**Action**[= *setting%*]

Remarks

This property is used to select an action for the MAPI messages control. The following list contains the possible actions. For each action, the Buffer column indicates whether the action works in the compose buffer (C), the read buffer (R), or both (C/R).

Setting	Buffer	Description
MESSAGE_FETCH	C/R	Creates a <i>message set</i> from selected messages in the Inbox. The message set includes all messages of type FetchMsgType found in the Inbox, sorted as selected by the FetchSorted, and qualified by FetchUnreadOnly. Any attachment files in the read buffer are deleted when a subsequent fetch action occurs.
MESSAGE_SENDDLG	C	Sends a message using a dialog box. Prompts the user for the various components of a message (subject, recipients, text, and so on) and submits the message to the mail server for delivery.
MESSAGE_SENDDLG	C	All message properties associated with a message being built in the compose buffer (an outgoing message with MsgIndex = 1) form the basis for the displayed message dialog box. Changes made in the dialog box, however, do not alter information in the compose buffer.

MESSAGE_SEND	C	Sends a message without using a dialog box. Submits the outgoing message to the mail server for delivery. No dialog box is displayed, and an error occurs if a user attempts to send a message that has no recipients or if attachment path names are missing.
MESSAGE_SAVEMSG	C	Saves the message currently in the compose buffer (with MsgIndex = 1).
MESSAGE_COPY	R	Copies the currently indexed message to the compose buffer. Sets the MsgIndex property to 1.
MESSAGE_COMPOSE	R	Composes a message. Clears all of the components of the compose buffer. Sets the MsgIndex property to 1.
MESSAGE_REPLY	R	Replies to a message. Copies the currently indexed message to the compose buffer as a reply and adds RE: to the beginning of the Subject line. The currently indexed message originator becomes the outgoing message recipient, then text is copied, and so on. Sets the MsgIndex property to 1.
MESSAGE_REPLYALL	R	Replies to all message recipients. Same as Reply, except that all other To: and CC: recipients are maintained. Sets the MsgIndex property to 1.
MESSAGE_FORWARD	R	Forwards a message. Copies the currently indexed message to the compose buffer as a forwarded message and adds FW: to the beginning of the Subject line. Sets the MsgIndex property to 1.
MESSAGE_DELETE	R	Deletes a message. Deletes all components of the currently indexed message, reduces the MsgCount property by 1, and decrements the index number of each message after the deleted message by 1. If the deleted message was the last message in the set, this action decrements the MsgIndex by 1.
MESSAGE_SHOWADBOOK	C/R	Displays the mail Address Book dialog box. The user can use the Address Book to create or modify a recipient set. Any changes to the Address Book outside of the compose buffer (when MsgIndex does not equal 1) are not saved.
MESSAGE_SHOWDETAILS	C/R	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. At a minimum, it contains the display name and address of the recipient.
MESSAGE_RESOLVENAME	C/R	Resolves the name of the currently indexed recipient. Searches the Address Book for a match on the currently indexed recipient name. If no match is found, an error is returned. (No match is not considered ambiguous.) The AddressResolveUI property determines whether a dialog box is displayed to resolve ambiguous names. This action does not provide additional resolution of the message originator's name or address. This action may cause the RecipType property to change.
RECIPIENT_DELETE	C	Deletes the currently indexed recipient. Automatically reduces the RecipCount property by 1, and decrements the index number of each recipient after the deleted recipient 1. If the deleted recipient was the last recipient in the set, this action decrements the RecipIndex by 1.
ATTACHMENT_DELETE	C	Deletes the currently indexed attachment. Automatically

reduces the AttachmentCount property by 1, and decrements the index of each attachment after the deleted attachment is decremented by 1.

If the deleted attachment was the last attachment in the set, this action decrements the AttachmentIndex by 1.

To avoid using unnecessary disk space, you should delete all temporary attachment files associated with the compose buffer before selecting any actions that alter attachment data for the message in the compose buffer. These include the following actions:

- MESSAGE_COPY
- MESSAGE_COMPOSE
- MESSAGE_REPLY
- MESSAGE_REPLYALL
- MESSAGE_FORWARD
- ATTACHMENT_DELETE (delete only the temporary attachment file associated with the deleted attachment)

Data Type

Integer

AddressCaption Property

Description

Specifies the caption appearing at the top of the Address Book dialog box when the Action property is set to MESSAGE_SHOWADBOOK (show the address book).

Usage

[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

Description

Specifies the number of edit controls available to the user in the Address Book dialog box when the Action property is set to MESSAGE_SHOWADBOOK (show the address book).

Usage

[*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

Description

Specifies the appearance of the To edit control in the Address Book when the **Action** property is set to MESSAGE_SHOWADBOOK (show the address book).

Usage

[*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

Description

Specifies whether the Address Book can be modified.

Usage

[*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

Description

Specifies whether a dialog box is displayed for recipient name resolution during addressing when the Action property is set to MESSAGE_RESOLVENAME (resolve name of currently indexed recipient).

Usage

[*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

Description

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.

Usage

[*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

Description

Sets the currently indexed attachment. This property is not available at design time.

Usage

[*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, set the Action property to ATTACHMENT_DELETE (delete the currently indexed attachment). Attachments can be added or deleted only when the MsgIndex property is set to 1.

Data Type

Long

AttachmentName Property

Description

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.

Usage

`[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

Description

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.

Usage

`[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

Description

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.

Usage

`[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

Description

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.

Usage

[*form.*]MapiMessages.**AttachmentType**[= *type%*]

Remarks

The AttachmentType property settings are:

Setting	Description
----------------	--------------------

ATTACHTYPE_DATA	The attachment is a data file.
-----------------	--------------------------------

ATTACHTYPE_EOLE	The attachment is an embedded OLE object.
-----------------	---

ATTACHTYPE_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

Description

Specifies the message type to populate the message set.

Usage

[*form.*]MapiMessages.**FetchMsgType**[= *string*\$]

Remarks

This property determines which message types are added to the message set when the MAPI messages control is invoked with the Action property set to MESSAGE_FETCH. A null or empty string in this property specifies an interpersonal message type (IPM), which is the default.

Data Type

String

FetchSorted Property

Description

Specifies the message order when populating the message set with messages from the Inbox.

Usage

[*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

Description

Determines whether to restrict the messages in the message set to unread messages only.

Usage

[*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

Description

Specifies the conversation thread identification value for the currently indexed message. It is read-only unless MsgIndex is set to 1.

Usage

[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

Description

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.

Usage

[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

Description

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.

Usage

[*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

Description

Specifies the string identifier of the currently indexed message. This property is not available at design time and is read-only at run time.

Usage

[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

Description

Specifies the index number of the currently indexed message. This property is not available at design time.

Usage

[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

Description

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.

Usage

`[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

Description

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.

Usage

[*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

Description

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.

Usage

[*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

Description

Indicates whether the message has already been read. This property is not available at design time and is read-only at run time.

Usage

[*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

Description

Specifies whether a return receipt is requested for the currently indexed message. This property is not available at design time.

Usage

[*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

Description

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.

Usage

[*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

Description

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.

Usage

[*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

Description

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.

Usage

[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

Description

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.

Usage

[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

Description

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.

Usage

[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

Description

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.

Usage

`[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 `MESSAGE_RESOLVENAME` setting of the `Action` property uses the recipient name as it is stored here.

Data Type

String

RecipIndex Property

Description

Sets the currently indexed recipient. This property is not available at design time.

Usage

[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, set the Action property to RECIPIENT_DELETE (delete the current indexed recipient). Recipients can be added or deleted only when the MsgIndex property is set to 1.

Data Type

Long

RecipType Property

Description

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.

Usage

[form.]MapiMessages.**RecipType**[= setting%]

Remarks

The RecipType property settings are:

Setting	Description
RECIPTYPE_ORIG	The message originator.
RECIPTYPE_TO	The recipient is a primary recipient.
RECIPTYPE_CC	The recipient is a copy recipient.
RECIPTYPE_BCC	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

Description

Stores the current messaging session handle. This property is not available at design time.

Usage

[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

Error Messages

The following table lists the trappable errors for the MAPI session and the MAPI messages controls.

Error number	Message explanation
32000	Success. The action returned successfully.
32001	User abort. The user cancelled the process. The current action was not completed.
32002	Failure. An unspecified error occurred during the current action. For example, the action was unable to delete or address mail correctly.
32003	Login failure. There was no default sign-on, and the user failed to sign on correctly.
32004	Disk full. The disk is full. The current action could not create a disk file.
32005	Insufficient memory. There is insufficient memory to proceed with the current action.
32006	Access denied.
32008	Too many sessions. The user has too many sessions open at once.
32009	Too many files. Too many file attachments are contained in the message. The mail wasn't sent or read.
32010	Too many recipients. There are too many message recipients specified. Mail wasn't sent or read.
32011	Attachment not found. The specified attachment was not found. Mail wasn't sent.
32012	Attachment open failure. The attachment could not be located. Mail wasn't sent. Verify that the AttachmentPathName property is valid.
32013	Attachment write failure. An attachment could not be written to a temporary file. Check directory permissions.
32014	Unknown recipient. The recipient does not appear in the address list. Mail wasn't sent.
32015	Bad recipient type. The type of recipient was incorrect. Valid type values are 1 (primary recipient), 2 (copy recipient), and 3 (blind copy recipient).
32016	No messages. Unable to find the next message.
32017	Invalid message. An invalid message ID was used. The current action was not completed.
32018	Text too large. The text in the message was too large to be sent. The mail wasn't sent. Text is limited to 32K.
32019	Invalid session. 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.
- 32050** **Session ID already exists.**
The MAPI messages control is already using a valid session ID.
- 32051** **Read-only in read buffer.**
The property is read-only while in the read buffer (MsgIndex is not set to 1).
- 32052** **Valid in compose buffer only.**
The action is valid only in the compose buffer.
- 32053** **No valid session ID.**
The MAPI messages control does not have a valid session handle from the MAPI session control.
- 32054** **Originator information not available.**
You cannot see message originator information while in the compose buffer (MsgIndex set to 1).
- 32055** **Action not valid in compose buffer.**
The attempted action is not valid in the compose buffer (MsgIndex set to 1).
- 32056** **Control failure.**
An unspecified error occurred while using the control.
- 32057** **No recipients.**
There are no specified recipients for this action.
- 32058** **No attachments.**
There are no specified attachments for this action.

▪ Masked Edit

[See Also](#)

[Properties](#)

[Methods](#)

[Events](#)

Description

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

MSMASKED.VBX

Object Type

MaskedTextBox

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.

Bound Properties

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.

For more information on using bound controls, refer to Chapter 20, Accessing Databases with the Data Control, in the *Programmers Guide*.

See Also

[Bound Properties](#)

[Clearing Text](#)

[Editing with a Mask](#)

[Selecting Text](#)

Bound Properties

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.

For more information on using bound controls, refer to Chapter 20, *Accessing Databases with the Data Control*, in the *Programmers Guide*.

Clearing Text

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 = ""
```

Editing with a Mask

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.

Selecting Text

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, Events, and Methods

Proper verbage here; split PEM

All of the properties, events, and methods for this control are listed in the following table. Properties and events that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*). They are documented in the following sections. (Note that the list order is alphabetic from top to bottom, and then from left to right.) See the Visual Basic *Language Reference* or Help for documentation on the remaining properties, events, and all methods.

Properties

<u>*AutoTab</u>	FontBold	<u>HideSelection</u>	SelLength
BackColor	FontItalic	hWnd	SelStart
BorderStyle	FontName	Index	<u>*SelText</u>
<u>*ClipMode</u>	FontSize	Left	TabIndex
<u>*ClipText</u>	FontStrikethru	<u>*Mask</u>	TabStop
DataChanged	<u>*FontUnderline</u>	<u>*MaxLength</u>	Tag
DataField	ForeColor	MousePointer	<u>*Text</u>
DataSource	<u>*Format</u>	Name	Top
DragIcon	<u>*FormattedText</u>	Parent	Visible
DragMode	Height	<u>*PromptChar</u>	Width
Enabled	HelpContextID	<u>*PromptInclude</u>	

Text is the default value of the control

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

Events

Change	GotFocus	KeyUp
DragDrop	KeyDown	LostFocus
DragOver	KeyPress	<u>*ValidationError</u>

Methods

BringToFront	Move	SetFocus	ZOrder
Drag	Refresh		

AutoTab Property

Description

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.

Usage

[*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

Description

Determines whether to include or exclude the literal characters in the input mask when doing a cut or copy command.

Usage

[*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

Description

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.

Usage

[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

Description

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.

Usage

[*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

Description

Specifies the format for displaying and printing numbers, dates, times, and text.

Usage

[*form.*]MaskedEdit.Format [= *format\$*]

Remarks

The Format property defines the format expression 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.

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	dddd	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

Description

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.

Usage

[*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

HideSelection Property

Description

Specifies whether the selection in a masked edit control is hidden when the control loses focus.

Usage

[*form.*]MaskedEdit.HideSelection [= { **True** | **False** }]

Remarks

Normally, selected text in a control is hidden when it loses focus. This is the default action of the property.

When there is selected text that doesn't have the focus, setting the HideSelection property to **False** inhibits formatted display of the text the FormattedText property is ignored.

Setting	Description
False	The selection in the masked edit control remains highlighted when the control loses focus. When this value is specified, the control doesn't display the formatted text when it doesn't have the focus.
True	(Default) The selection in the masked edit control is hidden when the control loses the focus. The control displays formatted text according to the FormattedText property.

Data Type

Integer (Boolean)

Mask Property

Description

Determines the input mask for the control.

Usage

[*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.
(###) ###-####	Standard North American phone number.
(###) ###-#### Ext(#####)	Standard North American phone number with extension.
###-##-####	Social Security Number.
##-??-##	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.
A	Alphanumeric character placeholder. For example: a-z, A-Z, or 0-9.
?	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

Description

Sets or returns the maximum length of the masked edit control.

Usage

[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

Description

Sets or returns the character used to prompt a user for input.

Usage

[*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

Description

Specifies whether prompt characters are contained in the Text property value. Use the PromptChar property to change the value of the prompt character.

Usage

[*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

Description

Sets or returns the text contained in the control.

Usage

[*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

Description

Sets or returns the text contained in the control. This property is not available at design time.

Usage

[*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

String

ValidationError Event

Description

Occurs when the masked edit field receives invalid input, as determined by the input mask.

Syntax

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](#)

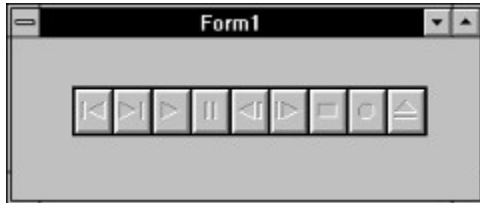
[Methods](#)

[Events](#)

Description

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

MCI.VBX

Object Type

MControl

Remarks

For this control to work, MCI services must be present. These services are provided in the Windows operating system version 3.1 and the Windows graphical environment version 3.0 with Multimedia Extensions version 1.0.

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. In Visual C++, place the MCI Open command in the **OnInitDialog** function for a dialog or the **OnInitUpdate** function for a form view.

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 file MCI.VBX 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.

```
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.

```
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 (*). For documentation on the remaining properties, see Appendix A in the *Custom Control Reference*. Properties beginning with *Button* are defined for each of the nine individual buttons in the multimedia MCI control.

<u>*AutoEnable</u>	<u>*Error</u>	Name	<u>*To</u>
BorderStyle	<u>*ErrorMessage</u>	<u>*Notify</u>	Top
<u>*ButtonEnabled</u>	<u>*FileName</u>	<u>*NotifyMessage</u>	<u>*Track</u>
<u>*ButtonVisible</u>	<u>*Frames</u>	<u>*NotifyValue</u>	<u>*TrackLength</u>
<u>*CanEject</u>	<u>*From</u>	<u>*Orientation</u>	<u>*TrackPosition</u>
<u>*CanPlay</u>	Height	<u>*Position</u>	<u>*Tracks</u>
<u>*CanRecord</u>	HelpContextID	<u>*RecordMode</u>	<u>*UpdateInterva</u>
			l
<u>*CanStep</u>	hWnd	<u>*Shareable</u>	<u>*UsesWindows</u>
<u>*Command</u>	<u>*hWndDisplay</u>	<u>*Silent</u>	<u>*Visible</u>
<u>*DeviceID</u>	Index	<u>*Start</u>	<u>*Wait</u>
<u>*DeviceType</u>	Left	TabIndex	Width
DragIcon	<u>*Length</u>	TabStop	
DragMode	<u>*Mode</u>	Tag	
<u>*Enabled</u>	MousePointer	<u>*TimeFormat</u>	

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 and Visual C++.

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 (*). For documentation on the remaining events, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

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*.

<u>*ButtonClick</u>	<u>*ButtonGotFocus</u>	<u>*Done</u>	DragOver
<u>*ButtonCompleted</u>	<u>*ButtonLostFocus</u>	DragDrop	<u>*StatusUpdate</u>

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 on 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	

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

AutoEnable Property, Multimedia MCI Control

Description

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.

Visual Basic

```
[form.]MMControl.AutoEnable[ = {True | False}]
```

Visual C++

```
pMMControl->GetNumProperty("AutoEnable")  
pMMControl->SetNumProperty("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 <i>ButtonEnabled</i> 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 *ButtonEnabled* properties. When the Enabled and AutoEnable properties are both **True**, the *ButtonEnable* 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

Description

Determines if a button in the control is enabled or disabled (dimmed).

Visual Basic

```
[form.]MMControl.ButtonEnabled[ = {True | False}]
```

Visual C++

```
pMMControl->GetNumProperty("ButtonEnabled")
```

```
pMMControl->SetNumProperty("ButtonEnabled", {TRUE | FALSE})
```

Remarks

The effects of the *ButtonEnabled* properties are superseded by the *Enabled* and *AutoEnable* properties. Individual *ButtonEnabled* 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 *ButtonEnabled* property settings for the multimedia MCI control.

Setting	Description
False	(Default) Disables (dims) the button specified by <i>Button</i> . This button's function is not available in the control.
True	Enables the button specified by <i>Button</i> . This button's function is available in the control.

Data Type

Integer (Boolean)

ButtonVisible Property, Multimedia MCI Control

Description

Determines if the specified button is displayed in the control.

Visual Basic

[*form.*]MMControl.**ButtonVisible**[= {**True** | **False**}]

Visual C++

pMMControl->**GetNumProperty**("ButtonVisible")

pMMControl->**SetNumProperty**("ButtonVisible", {**TRUE** | **FALSE**})

Remarks

The effects of the *ButtonVisible* properties are superseded by the *Visible* property. Individual *ButtonVisible* 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 *ButtonVisible* property settings for the multimedia MCI control.

Setting	Description
False	Does not display the button specified by <i>Button</i> . This button's function is not available in the control.
True	(Default) Displays the button specified by <i>Button</i> .

Data Type

Integer (Boolean)

CanEject Property, Multimedia MCI Control

Description

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.

Visual Basic

[form.]MMControl.CanEject

Visual C++

pMMControl->GetNumProperty("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

Description

Determines if the open MCI device can play. This property is not available at design time and is read-only at run time.

Visual Basic

[form.]MMControl.CanPlay

Visual C++

pMMControl->GetNumProperty("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

Description

Determines if the open MCI device can record. This property is not available at design time and is read-only at run time.

Visual Basic

[form.]MMControl.CanRecord

Visual C++

pMMControl->GetNumProperty("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

Description

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.

Visual Basic

[form.]MMControl.CanStep

Visual C++

pMMControl->GetNumProperty("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

Description

Specifies an MCI command to execute. This property is not available at design time.

Visual Basic

```
[form.]MMControl.Command[ = cmdstring$]
```

Visual C++

```
pMMControl->GetStrProperty("Command")  
pMMControl->SetStrProperty("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. Notify (False) Wait (True) Shareable DeviceType FileName
Close	Closes a device using the MCI_CLOSE command. Notify (False) Wait (True)
Play	Plays a device using the MCI_PLAY command. Notify (True) Wait (False) From 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. Notify (False) Wait (True)
Stop	Stops playing or recording using the MCI_STOP command. Notify (False) Wait (True)
Back	Steps backwards using the MCI_STEP command. Notify (False) Wait (True) Frames
Step	Steps forwards using the MCI_STEP command. Notify (False) Wait (True) 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.

	Notify (False) 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. Notify (False) 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. Notify (False) Wait (True) To
Record	Records using the MCI_RECORD command. Notify (True) Wait (False) From To RecordMode (0Insert)
Eject	Ejects media using the MCI_SET command. Notify (False) Wait (True)
Sound	Plays a sound using the MCI_SOUND command. Notify (False) Wait (False) FileName
Save	Saves an open file using the MCI_SAVE command. Notify (False) Wait (True) FileName

Data Type
String

DeviceID Property, Multimedia MCI Control

Description

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.

Visual Basic

```
[form.]MMControl.DeviceID[ = id%]
```

Visual C++

```
pMMControl->GetNumProperty("DeviceID")  
pMMControl->SetNumProperty("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

Description

Specifies the type of MCI device to open.

Visual Basic

```
[form.]MMControl.DeviceType[ = device$]
```

Visual C++

```
pMMControl->GetStrProperty("DeviceType")  
pMMControl->SetStrProperty("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

Description

Determines if the control can respond to user-generated events, such as the KeyPress and mouse events.

Visual Basic

```
[form.]MMControl.Enabled[ = {True | False}]
```

Visual C++

```
pMMControl->GetNumProperty("Enabled")  
pMMControl->SetNumProperty("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 *ButtonEnable* 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 *ButtonEnable* 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 <i>ButtonEnable</i> properties to enable or disable individual buttons in the control.

Data Type

Integer (Boolean)

Error Property, Multimedia MCI Control

Description

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.

Visual Basic

[form.]MMControl.**Error**

Visual C++

pMMControl->**GetNumProperty("Error")**

Remarks

If the last MCI command did not cause an error, this value is 0.

Data Type

Integer

ErrorMessage Property, Multimedia MCI Control

Description

Describes the error code stored in the Error property. This property is not available at design time and is read-only at run time.

Visual Basic

[form.]MMControl.ErrorMessage

Visual C++

pMMControl->GetStrProperty("ErrorMessage")

Data Type

String

FileName Property, Multimedia MCI Control

Description

Specifies the file to be opened by an Open command or saved by a Save command.

Visual Basic

```
[form.]MMControl.FileName[ = stringexpression]
```

Visual C++

```
pMMControl->GetStrProperty("FileName")  
pMMControl->SetStrProperty("FileName", stringexpression)
```

Remarks

The argument *stringexpression* specifies the file to be opened or saved.

Data Type

String

Frames Property, Multimedia MCI Control

Description

Specifies the number of frames the Step command steps forward or the Back command steps backward. This property is not available at design time.

Visual Basic

```
[form.]MMControl.Frames[ = frames&]
```

Visual C++

```
pMMControl->GetNumProperty("Frames")  
pMMControl->SetNumProperty("Frames", frames)
```

Remarks

The argument *frames&* specifies the number of frames to step forward or backward.

Data Type

Long

From Property, Multimedia MCI Control

Description

Specifies the starting point, using the current time format, for the Play or Record command. This property is not available at design time.

Visual Basic

```
[form.]MMControl.From[ = location&]
```

Visual C++

```
pMMControl->GetNumProperty("From")  
pMMControl->SetNumProperty("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

Description

Specifies the output window for MCI MMovie or Overlay devices that use a window to display output. This property is not available at design time.

Visual Basic

[*form.*]MMControl.**hWndDisplay**

Visual C++

pMMControl->**GetNumProperty("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 22, "Calling DLL Procedures," 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

Description

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.

Visual Basic

[form.]MMControl.Length

Visual C++

pMMControl->GetNumProperty("Length")

Data Type

Long

Mode Property, Multimedia MCI Control

Description

Specifies the current mode of an open MCI device. This property is not available at design time and is read-only at run time.

Visual Basic

[*form.*]MMControl.Mode

Visual C++

pMMControl->GetNumProperty("Mode")

Remarks

The following table lists the Mode property return values for the multimedia MCI control.

Value	Setting/Device mode
524	MCI_MODE_NOT_OPEN Device is not open.
525	MCI_MODE_STOP Device is stopped.
526	MCI_MODE_PLAY Device is playing.
527	MCI_MODE_RECORD Device is recording.
528	MCI_MODE_SEEK Device is seeking.
529	MCI_MODE_PAUSE Device is paused
530	MCI_MODE_READY Device is ready.

Data Type

Long

Notify Property, Multimedia MCI Control

Description

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.

Visual Basic

```
[form.]MMControl.Notify[ = {True | False}]
```

Visual C++

```
pMMControl->GetNumProperty("Notify")  
pMMControl->SetNumProperty("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

Description

Describes the notify code returned in the Done event. This property is not available at design time and is read-only at run time.

Visual Basic

[form.]MMControl.NotifyMessage

Visual C++

pMMControl->GetStrProperty("NotifyMessage")

Data Type

String

NotifyValue Property, Multimedia MCI Control

Description

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.

Visual Basic

[form.]MMControl.NotifyValue

Visual C++

pMMControl->GetNumProperty("NotifyValue")

Remarks

The following table lists the NotifyValue return values for the multimedia MCI control.

Value	Setting/Device mode
1	MCI_NOTIFY_SUCCESSFUL Command completed successfully.
2	MCI_NOTIFY_SUPERSEDED Command was superseded by another command.
4	MCI_NOTIFY_ABORTED Command was aborted by the user.
8	MCI_NOTIFY_FAILURE 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

Description

Determines whether buttons on the control are arranged vertically or horizontally.

Visual Basic

[*form.*]MMControl.**Orientation**[= *orientation%*]

Visual C++

pMMControl->**GetNumProperty("Orientation")**

pMMControl->**SetNumProperty("Orientation", *orientation*)**

Remarks

The following table lists the Orientation property settings for the multimedia MCI control.

Setting	Description
0	Buttons are arranged horizontally.
1	Buttons are arranged vertically.

Data Type

Integer (Enumerated)

Position Property, Multimedia MCI Control

Description

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.

Visual Basic

[form.]MMControl.**Position**

Visual C++

pMMControl->**GetNumProperty("Position")**

Data Type

Long

RecordMode Property, Multimedia MCI Control

Description

Specifies the current recording mode for those MCI devices that support recording.

Visual Basic

```
[form.]MMControl.RecordMode[ = mode%]
```

Visual C++

```
pMMControl->GetNumProperty("RecordMode")  
pMMControl->SetNumProperty("RecordMode", mode)
```

Remarks

The following table lists the RecordMode property settings for the multimedia MCI control.

Setting	Recording mode
----------------	-----------------------

0	Insert
---	--------

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

Description

Determines if more than one program can share the same MCI device.

Visual Basic

```
[form.]MMControl.Shareable[ = {True | False}]
```

Visual C++

```
pMMControl->GetNumProperty("Shareable")  
pMMControl->SetNumProperty("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

Description

Determines if sound plays.

Visual Basic

```
[form.]MMControl.Silent[ = {True | False}]
```

Visual C++

```
pMMControl->GetNumProperty("Silent")
```

```
pMMControl->SetNumProperty("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

Description

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.

Visual Basic

[form.]MMControl.Start

Visual C++

pMMControl->GetNumProperty("Start")

Data Type

Long

TimeFormat Property, Multimedia MCI Control

Description

Specifies the time format used to report all position information.

Visual Basic

```
[form.]MMControl.TimeFormat[ = format&]
```

Visual C++

```
pMMControl->GetNumProperty("TimeFormat")  
pMMControl->SetNumProperty("TimeFormat", format)
```

Remarks

The following table lists the TimeFormat property settings for the multimedia MCI control.

Value	Setting/Time format
0	MCI_FORMAT_MILLISECONDS Milliseconds are stored as a 4-byte integer variable.
1	MCI_FORMAT_HMS Hours, minutes, and seconds are packed into a 4-byte integer. From least significant byte to most significant byte, the individual data values are: Hours (least significant byte) Minutes Seconds Unused (most significant byte)
2	MCI_FORMAT_MSF Minutes, seconds, and frames are packed into a 4-byte integer. From least significant byte to most significant byte, the individual data values are: Minutes (least significant byte) Seconds Frames Unused (most significant byte)
3	MCI_FORMAT_FRAMES Frames are stored as a 4-byte integer variable.
4	MCI_FORMAT_SMPTE_24 24-frame SMPTE packs the following values in a 4-byte variable from least significant byte to most significant byte: Hours (least significant byte) Minutes Seconds Frames (most significant byte) 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	MCI_FORMAT_SMPTE_25 25-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.
6	MCI_FORMAT_SMPTE_30 30-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.
7	MCI_FORMAT_SMPTE_30DROP 30-drop-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE.
8	MCI_FORMAT_BYTES Bytes are stored as a 4-byte integer variable.
9	MCI_FORMAT_SAMPLES

10 Samples are stored as a 4-byte integer variable.
MCI_FORMAT_TMSF
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

Description

Specifies the ending point, using the current time format, for the Play or Record command. This property is not available at design time.

Visual Basic

```
[form.]MMControl.To[ = location&]
```

Visual C++

```
pMMControl->GetNumProperty("To")  
pMMControl->SetNumProperty("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

Description

Specifies the track about which the TrackLength and TrackPosition properties return information. This property is not available at design time.

Visual Basic

```
[form.]MMControl.Track[ = track&]
```

Visual C++

```
pMMControl->GetNumProperty("Track")  
pMMControl->SetNumProperty("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

Description

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.

Visual Basic

[form.]MMControl.TrackLength

Visual C++

pMMControl->GetNumProperty("TrackLength")

Data Type

Long

TrackPosition Property, Multimedia MCI Control

Description

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.

Visual Basic

[form.]MMControl.TrackPosition

Visual C++

pMMControl->GetNumProperty("TrackPosition")

Data Type

Long

Tracks Property, Multimedia MCI Control

Description

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.

Visual Basic

[form.]MMControl.Tracks

Visual C++

pMMControl->GetNumProperty("Tracks")

Data Type

Long

UpdateInterval Property, Multimedia MCI Control

Description

Specifies the number of milliseconds between successive StatusUpdate events.

Visual Basic

```
[form.]MMControl.UpdateInterval[ = milliseconds%]
```

Visual C++

```
pMMControl->GetNumProperty("UpdateInterval")  
pMMControl->SetNumProperty("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

Description

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.

Visual Basic

[form.]MMControl.UsesWindows

Visual C++

pMMControl->GetNumProperty("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

Description

Determines if the multimedia MCI control is visible or invisible at run time.

Visual Basic

```
[form.]MMControl.Visible[ = {True | False}]
```

Visual C++

```
pMMControl->GetNumProperty("Visible")  
pMMControl->SetNumProperty("Visible", {TRUE | FALSE})
```

Remarks

The effect of the Visible property supersedes the effects of the individual *ButtonVisible* properties. When the multimedia MCI control is visible, the individual *ButtonVisible* properties govern the visibility of the associated buttons in the control. When the Visible property is **False**, the entire control is invisible, and the *ButtonVisible* 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 <i>ButtonVisible</i> property. This button's function is still available in the control.

Data Type

Integer (Boolean)

Wait Property, Multimedia MCI Control

Description

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.

Visual Basic

```
[form.]MMControl.Wait[ = {True | False}]
```

Visual C++

```
pMMControl->GetNumProperty("Wait")  
pMMControl->SetNumProperty("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

Description

Occurs when the user presses and releases the mouse button over one of the buttons in the multimedia MCI control.

Visual Basic

Sub *MMControl*._ButtonClick (*Cancel As Integer*)

Visual C++

Function Signature:

void *CMyDialog::OnButtonClickMMControl* (**UINT, int, CWnd*, LPVOID lpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (**LONG, lpParams**)

Remarks

Button may be any of the following: Back, Eject, Next, Pause, Play, Prev, Record, Step, or Stop.

Each of the *ButtonClick* 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 *ButtonClick* 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 <i>ButtonClick</i> 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 *ButtonClick* 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 *ButtonClick* 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.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

ButtonCompleted Event, Multimedia MCI Control

Description

Occurs when the MCI command activated by a multimedia MCI control button finishes.

Visual Basic

Sub *MMControl_ButtonCompleted* (*Errorcode As Long*)

Visual C++

Function Signature:

void *CMyDialog::OnButtonCompletedMMControl* (**UINT, int, CWnd*, LPVOID lpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (LONG, lpParams)

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 *ButtonClick* event, the *ButtonCompleted* event is not triggered.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

ButtonGotFocus Event, Multimedia MCI Control

Description

Occurs when a button in the multimedia MCI control receives the input focus.

Visual Basic

Sub *MMControl_ButtonGotFocus* ()

Visual C++

Function Signature:

void *CMyDialog::OnButtonGotFocusMMControl* (**UINT, int, CWnd*, LPVOID**)

Remarks

Button may be any of the following: Back, Eject, Next, Pause, Play, Prev, Record, Step, or Stop.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

ButtonLostFocus Event, Multimedia MCI Control

Description

Occurs when a button in the multimedia MCI control loses the input focus.

Visual Basic

Sub *MMControl_ButtonLostFocus* (**)**

Visual C++

Function Signature:

void *CMyDialog::OnButtonLostFocusMMControl* (**UINT, int, CWnd*, LPVOID**)

Remarks

Button may be any of the following: Back, Eject, Next, Pause, Play, Prev, Record, Step, or Stop.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

Done Event, Multimedia MCI Control

Description

Occurs when an MCI command for which the Notify property is **True** finishes.

Visual Basic

Sub *MMControl_Done* (*NotifyCode* **As Long**)

Visual C++

Function Signature:

void *CMyDialog::OnDoneMMControl* (**UINT, int, CWnd*, LPVOID lpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (LONG, lpParams)

Remarks

The *NotifyCode* argument indicates whether the MCI command succeeded. It can take any of the following settings.

Value	Setting/Result
1	MCI_NOTIFY_SUCCESSFUL Command completed successfully.
2	MCI_NOTIFY_SUPERSEDED Command was superseded by another command.
4	MCI_NOTIFY_ABORTED Command was aborted by the user.
8	MCI_NOTIFY_FAILURE Command failed.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

StatusUpdate Event, Multimedia MCI Control

Description

Occurs automatically at intervals given by the UpdateInterval property.

Visual Basic

Sub *MMControl_StatusUpdate* ()

Visual C++

Function Signature:

void *CMyDialog::OnStatusUpdateMMControl* (**UINT, int, CWnd*, LPVOID**)

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.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

Outline

[See Also](#)

[Properties](#)

[Events](#)

[Methods](#)

[Error Messages](#)

Description

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. This is what the control looks like as an icon in the Toolbox:



File Name

MSOUTLIN.VBX

Object Type

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.

Constants

The values of the property and error constants are defined in the Visual Basic CONSTANT.TXT file.

Distribution Note When you create and distribute applications that use the outline control, you should install the file MSOUTLIN.VBX in the customers 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 items level of subordination. Each level of indentation is a level of subordination you specify with the Indent property.

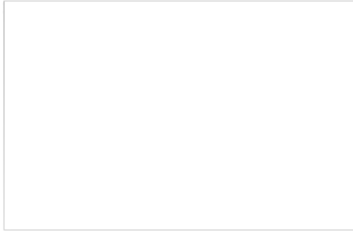
Plus/minus pictures indicate whether subordinate items are visible or hidden. When the plus sign is clicked, subordinate items become visible and a minus sign replaces the plus sign. When the minus sign is clicked, the subordinate items are hidden and a plus sign replaces the minus sign.

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.



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 controls 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 (*). For documentation on the remaining properties, see Appendix A, Standard Properties, Events, and Methods.

Properties

BackColor	FontUnderline	Left	* PictureOpen
BorderStyle	ForeColor	* List	* PicturePlus
DragIcon1	* FullPath	ListCount	* PictureType
DragMode1	* HasSubItems	ListIndex	* Style
Enabled	Height	MousePointer	TabIndex
* Expand	HelpContextID1	Name2	TabStop
FontBold	hWnd	Parent1	Tag
FontItalic	* Indent1	* PathSeparator	Top
FontName	Index	* PictureClosed	* TopIndex
FontSize	* IsItemVisible	* PictureLeaf	Visible
FontStrikethru	ItemData	* PictureMinus	Width

1 Available only in Visual Basic.

2 Equivalent to the CtlName property in Visual Basic 1.0 and Visual C++.

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 (*). For documentation on the remaining events, see Appendix A, Standard Properties, Events, and Methods.

Events

Click	DragOver ₁	KeyPress	MouseMove
* <u>Collapse</u>	* <u>Expand</u>	KeyUp	MouseUp
DbClick	GotFocus	LostFocus	* <u>PictureClick</u>
DragDrop ₁	KeyDown	MouseDown	* <u>PictureDbClick</u>

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 (*). For documentation on the remaining methods, see Appendix A, Standard Properties, Events, and Methods.

Methods

* <u>AddItem</u>	Drag ₁	Refresh	SetFocus ₁
Clear	Move	* <u>RemoveItem</u>	ZOrder ₁

Expand Property

Description

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.

Visual Basic

```
[form.]Outline1.Expand(index%) = {True | False}
```

Visual C++

```
pOutline->GetNumProperty("Expand", index)  
pOutline->SetNumProperty("Expand", {TRUE | FALSE}, index)
```

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 items 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

Description

Returns the *fully qualified* name of an item. The fully qualified name is the concatenation of the item with its parent item, the parent items 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.

Visual Basic

```
[form.]Outline1.FullPath(index)
```

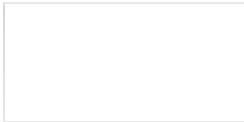
Visual C++

```
pOutline->GetStrProperty("FullPath")
```

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

Description

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.

Visual Basic

```
[form.]Outline1.HasSubItems(index)
```

Visual C++

```
pOutline->GetNumProperty("HasSubItems")
```

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) = MSOUTLINE_PICTURE_OPEN
    Else
        Outline1.PictureType(i) = MSOUTLINE_PICTURE_LEAF
    End If
Next
```

Data Type

Integer (Boolean)

Indent Property

Description

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.

Visual Basic

```
[form.]Outline1.Indent(index) [= indentation%]
```

Visual C++

```
pOutline->GetNumProperty("Break", index)  
pOutline->SetNumProperty("Break", indentation, index)
```

Remarks

If the value of *indentation%* is two or more than its parents 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:

```
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

Description

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.

Visual Basic

```
[form.]Outline1.IsItemVisible(index)
```

Visual C++

```
pOutline->GetNumProperty("IsItemVisible", index)  
pOutline->SetNumProperty("IsItemVisible", {TRUE | FALSE}, index)
```

Data Type

Integer (Boolean)

List Property

Description

Determines the items contained in the controls list portion. The list is a string array in which each element is a list item. Not available at design time.

Visual Basic

```
[form.]Outline1.List(index) [ = itemstring$]
```

Visual C++

```
pOutline->GetStrProperty("List", index)  
pOutline->SetStrProperty("List", itemstring, index)
```

Remarks

The outline controls 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

Description

Sets and returns the item delimiter string used when accessing the FullPath property. The default value is the backslash character (\).

Visual Basic

```
[form.]Outline1.PathSeparator[ = delimiter$]
```

Visual C++

```
pOutline->GetStrProperty("PathSeparator")  
pOutline->SetStrProperty("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

Description

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.

Visual Basic

```
[form.]Outline1.PictureClosed[ = picture%]  
[form.]Outline1.PictureOpen[ = picture%]  
[form.]Outline1.PictureLeaf[ = picture%]
```

Visual C++

```
pOutline->GetPictureProperty("PictureClosed")  
pOutline->SetPictureProperty("PictureClosed", picture)  
pOutline->GetPictureProperty("PictureOpen")  
pOutline->SetPictureProperty("PictureOpen", picture)  
pOutline->GetPictureProperty("PictureLeaf")  
pOutline->SetPictureProperty("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

Description

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.

Visual Basic

```
[form.]Outline1.PictureMinus[ = picture%]
```

```
[form.]Outline1.PicturePlus[ = picture%]
```

Visual C++

```
pOutline->GetPictureProperty("PictureMinus")
```

```
pOutline->SetPictureProperty("PictureMinus", picture)
```

```
pOutline->GetPictureProperty("PicturePlus")
```

```
pOutline->SetPictureProperty("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

Description

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.

Visual Basic

```
[form.]Outline1.PictureType(index)[ = type%]
```

Visual C++

```
pOutline->GetNumProperty("PictureType", index)  
pOutline->SetNumProperty("PictureType", type, index)
```

Remarks

The following table lists the PictureType property settings for the outline control.

Constant	Value	Description
MSOUTLINE_PICTURE_CLOSED	0	Use PictureClosed picture.
MSOUTLINE_PICTURE_OPEN	1	Use PictureOpen picture.
MSOUTLINE_PICTURE_LEAF	2	Use PictureLeaf picture.

If you dont set a value for PictureClosed, PictureOpen, and PictureLeaf, the outline control will use default pictures.

Data Type

Integer

Style Property

Description

Set and returns the style of graphics and text that appear for each item in the outline control.

Visual Basic

```
[form.]Outline1.Style[ = style%]
```

Visual C++

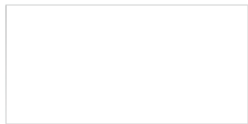
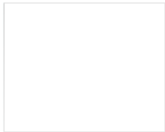
```
pOutline->GetNumProperty("Style")  
pOutline->SetNumProperty("Style", style)
```

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

Description

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.

Visual Basic

```
[form.]Outline1.TopIndex[ = top%]
```

Visual C++

```
pOutline->GetNumProperty("TopIndex")  
pOutline->SetNumProperty("TopIndex", top)
```

Data Type

Integer

Collapse Event

Description

Generated whenever an item is collapsed, which means the items subordinate items are hidden.

Visual Basic

Sub *Outline_Collapse* ([*Index As Integer*,] *I As Integer*)

Visual C++

Function Signature:

void *CMyDialog::OnOutlineCollapse* (**UINT, int, CWnd*, LPVOID / IpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (LONG, IpParams)

Remarks

This event passes *I*, the index of the item in the list that was closed.

Expand Event

Description

Generated whenever an item is expanded, which means the items subordinate items are visible.

Visual Basic

Sub *Outline_Expand* (*[Index As Integer,] I As Integer*)

Visual C++

Function Signature:

void *CMyDialog::OnOutlineExpand* (**UINT, int, CWnd*, LPVOID / IpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (LONG, IpParams)

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 items 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:

```
Sub Outline1_Expand (I As Long)
    If Outline1.HasSubItems(I) Then
        Outline1.PictureType(I) = MSOUTLINE_PICTURE_OPEN
    End If
End Sub
```

Note If you set an items Expand property to **True**, an Expand event will occur even if the item has no subordinate items.

PictureClick Event

Description

Generated whenever a type picture associated with an item is clicked.

Visual Basic

Sub *Outline_PictureClick* (*[Index As Integer,] I As Integer*)

Visual C++

Function Signature:

void *CMyDialog::OnOutlinePictureClick* (**UINT, int, CWnd*, LPVOID / IpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (LONG, IpParams)

Remarks

This event passes *I*, the index of the item whose picture was clicked.

PictureDbClick Event

Description

Generated whenever a type picture associated with an item is double-clicked.

Visual Basic

Sub *Outline_PictureDbClick* (*[Index* **As Integer**,] *I* **As Integer**)

Visual C++

Function Signature:

void *CMyDialog::OnOutlinePictureDbClick* (**UINT, int, CWnd*, LPVOID / IpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (LONG, IpParams)

Remarks

This event passes *I*, the index of the item whose picture was double-clicked.

AddItem Method

Description

Adds an item to the outline control at run time.

Visual Basic

```
[form.]Outline1.AddItem item [, index%]
```

Visual C++

```
pOutline->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 items 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

Description

Removes an item and its subordinate items from the outline control at run time.

Visual Basic

[form.]Outline1.RemoveItem index%

Visual C++

pOutline->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.

Error Messages

The following table lists the trappable errors for the outline control.

Error number	Message explanation
---------------------	----------------------------

32000	MSOUTLINE_BADPICFORMAT Picture Format Not Supported. Only bitmap files (*.BMP) and icon files (*.ICO) are valid picture formats.
32001	MSOUTLINE_BADINDENTATION Bad Outline Indentation. The indentation value for the item is two or more than the indentation level of its parent.
32002	MSOUTLINE_MEM Out of Memory. Too many items (6537 maximum) are in the list, or too much string space has been used.
32003	MSOUTLINE_PARENTNOTEXPANDED Parent Not Expanded. An item must be visible (expanded) in order to expand its subordinate items.

■

■ Pen Edit Controls

[Properties](#)

[Methods](#)

[Events](#)

[Functions](#)

[Error Messages](#)

Description

The pen edit custom controls (BEdit and HEdit) allow you to develop pen-aware applications under the Microsoft Windows for Pen Computing environment. The application designer may substitute these custom controls for the standard input controls.

The HEdit control is a pen-enhanced version of the text box control. The BEdit, or boxed edit control, provides the application with comb or box style guides that accept pen input. Each segment or box accepts only a single character of input.

Note This control requires Microsoft Windows for Pen Computing.

File Name

PENCNTRL.VBX

Object Type

VBEdit, VHEdit

Remarks

The HEdit and BEdit controls are similar to the standard Visual Basic text box, except that the user enters data into these controls using a pen instead of typing at the keyboard.

The HEdit control accepts free-form input from the user. This control supports most of the Visual Basic text box properties; however, it does not have Dynamic Data Exchange (DDE) capabilities.

The BEdit control expands upon the properties of the HEdit control and allows for additional manipulation of the writing area. This control accepts a single character of input in each box. This increases the accuracy of the recognition and in most cases is preferable to the HEdit control.

Distribution Note When you create and distribute applications that use the pen edit controls, you should install the file PENCNTRL.VBX 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 these controls are listed in the following table. Properties that apply *only* to these controls, or that require special consideration when used with them, are marked with an asterisk (*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

Alignment	<u>*CombStyle</u>	hWnd	Parent
BackColor	<u>*DelayRecog</u>	Index	<u>*ScrollBars</u>
<u>*BorderStyle</u>	Enabled	<u>*InflateBottom</u>	SelLength
<u>*CellHeight</u>	<u>*EraseInk</u>	<u>*InflateLeft</u>	SelStart
<u>*CellWidth</u>	FontBold	<u>*InflateRight</u>	SelText
<u>*CharSet</u>	FontItalic	<u>*InflateTop</u>	TabIndex
<u>*CombBaseLine</u>	FontName	<u>*InkColor</u>	TabStop
<u>*CombColor</u>	Font Size	<u>*InkDataMode</u>	Tag
<u>*CombEndHeight</u>	FontStrikethru	<u>*InkDataString</u>	Text
<u>*CombEndMarker</u>	FontUnderline	<u>*InkWidth</u>	Top
<u>*CombHeight</u>	ForeColor	Left	Visible
<u>*CombNumCols</u>	Height	MultiLine	Width
<u>*CombNumRows</u>	HelpContextID	Name	
<u>*CombSpacing</u>	<u>*hInk</u>	<u>*OnTap</u>	

Text is the default value of the control.

Note The HelpContextID, 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 and Visual C++.

Events

All of the events for these controls are listed in the following table. Events that apply *only* to these controls, or that require special consideration when used with them, are marked with an asterisk (*). For documentation of the remaining events, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

Change	GotFocus	KeyPress	<u>*RcResult</u>
DragDrop	KeyDown	KeyUp	<u>*Update</u>
DragOver			

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

Methods

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

Move **Refresh** **SetFocus** **ZOrder**

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

Functions

All of the functions for these controls are listed in the following table. Functions that apply *only* to these controls, or that require special consideration when used with them, are marked with an asterisk (*). The following functions are only available in Visual Basic.

*CPointerToVBType *VBTypeToCPointer

BorderStyle Property, Pen Edit Controls

Description

Sets or returns the style of the control border.

Visual Basic

[*form.*]Penctrl.**BorderStyle**[= *setting%*]

Visual C++

pPenctrl->**GetNumProperty**("BorderStyle")

pPenctrl->**SetNumProperty**("BorderStyle", *setting*)

Remarks

The valid BorderStyle settings for a BEdit control are the same as for the standard Visual Basic text and picture box controls. For the HEdit control, a new setting, underline, has been added. The underline setting may only be used on a single-line HEdit control.

Setting	Description
0	None.
1	(Default) Fixed Single.
2	Underline (HEdit only).

Data Type

Integer (Enumerated)

CellHeight, CellWidth Properties, Pen Edit Controls

Description

Set or return the height and width of the encapsulating cell for a comb- or box-style BEdit control.

Visual Basic

```
[form.]BEdit.CellHeight[ = setting&]  
[form.]BEdit.CellWidth[ = setting&]
```

Visual C++

```
pBEdit->GetNumProperty("CellHeight")  
pBEdit->SetNumProperty("CellHeight", setting)  
pBEdit->GetNumProperty("CellWidth")  
pBEdit->SetNumProperty("CellWidth", setting)
```

Remarks

CellHeight determines the maximum settings for the CombBaseLine, CombEndHeight, and CombHeight properties. CellWidth limits the CombSpacing.

These properties use the ScaleMode property setting of the underlying form, frame, or picture box. For example, if a form's ScaleMode property is set to 3 (pixels), and a BEdit control is placed on that form, the CellHeight and CellWidth properties for that control are measured in pixels. In Visual C++, all dimensions are in pixels.

Data Type

Long

CharSet Property, Pen Edit Controls

Description

Sets or returns the set of characters that will be recognized.

Visual Basic

```
[form.]Penctrl.CharSet[ = setting%]
```

Visual C++

```
pPenctrl->GetNumProperty("CharSet")  
pPenctrl->SetNumProperty("CharSet", setting)
```

Remarks

At design time, the CharSet property is set using the custom dialog. At run time, you must calculate the character set by summing the value of your choices (inclusive **Or**) according to the Alphabet codes found in the PENAPI.TXT file (Visual Basic) or the PENWIN.H file (Visual C++).

The following table lists the CharSet property settings for the pen edit controls.

Setting	Description
ALC_DEFAULT	(Default) A recognizer-dependent set of characters. The default system-wide character set always includes alphanumeric and white-space characters, punctuation marks, and gestures.
ALC_LCALPHA	Lowercase alphabetic characters.
ALC_UCALPHA	Uppercase alphabetic characters.
ALC_NUMERIC	Numeric characters: 0 through 9.
ALC_PUNC	Punctuation: ! - ; ' " ? () & : .
ALC_MATH	Math symbols: % ^ * () ■ + = { } < > , / .
ALC_MONETARY	Monetary symbols (for example: \$, .)
ALC_OTHER	All symbols not included in the preceding sets, for example ■ [] _ ~.
ALC_WHITE	Spaces between characters.
ALC_GESTURE	Gestures.

Data Type

Integer

CombBaseLine Property, Pen Edit Controls

Description

Sets or returns the distance from the top of the encapsulating cell to the base of the box or comb guide of a BEdit control.

Visual Basic

```
[form.]BEdit.CombBaseLine[ = setting&]
```

Visual C++

```
pBEdit->GetNumProperty("CombBaseLine")  
pBEdit->SetNumProperty("CombBaseLine", setting)
```

Remarks

CombBaseLine may range from 0 to the CellHeight. CombBaseLine also restricts the CombEndHeight and CombHeight properties.

This property uses the ScaleMode property setting of the underlying form, frame, or picture box. For example, if a form's ScaleMode property is set to 3 (pixels), and a BEdit control is placed on that form, the CombBaseLine property for that control is measured in pixels. In Visual C++, all dimensions are in pixels.

Data Type

Long

CombColor Property, Pen Edit Controls

Description

Sets or returns the color of the comb or box guides displayed within a BEdit control.

Visual Basic

```
[form.]BEdit.CombColor[ = color&]
```

Visual C++

```
pBEdit->GetNumProperty("CombColor")  
pBEdit->SetNumProperty("CombColor", color)
```

Remarks

The following table lists the CombColor property settings for the pen edit controls.

Setting	Description
&H80000006&	(Default) Window frame color as set in the Windows Control Panel. This corresponds to the WINDOW_FRAME constant defined in the Visual Basic CONSTANT.TXT file or the Visual C++ WINDOWS.H file.
(color)	In Visual Basic, color specified by using the RGB scheme or the QBColor function in code. In Visual C++, use the MAKESYSOLOR macro defined in the AFXEXT.H file to create system color constants. Standard RGB colors can be used in Visual C++ as well.

Data Type

Long

CombEndHeight Property, Pen Edit Controls

Description

CombEndHeight sets or returns the height of the teeth in the comb in a comb-style BEdit control.

Visual Basic

```
[form.]BEdit.CombEndHeight[ = setting&]
```

Visual C++

```
pBEdit->GetNumProperty("CombEndHeight")  
pBEdit->SetNumProperty("CombEndHeight", setting)
```

Remarks

The CombEndHeight property sets the height of the beginning and end teeth of the comb. This property has no effect when the BEdit control is in box style.

Note The value of the CombHeight and CombEndHeight properties cannot exceed the CombBaseLine value.

In Visual Basic, these properties use the ScaleMode property setting of the underlying form, frame, or picture box. For example, if a form's ScaleMode property is set to 3 (pixels), and a BEdit control is placed on that form, the CombEndHeight and CombHeight properties for that control are measured in pixels. In Visual C++, all the dimensions are in pixels.

Data Type

Long

CombEndMarker Property, Pen Edit Controls

Description

Determines whether the BEdit control end-of-text marker is visible.

Visual Basic

[*form.*]BEdit.CombEndMarker[= {True | False}]

Visual C++

pBEdit->GetNumProperty("CombEndMarker")

pBEdit->SetNumProperty("CombEndMarker", {TRUE | FALSE})

Remarks

The following table lists the CombEndMarker property settings for the pen edit controls.

Setting	Description
---------	-------------

True	(Default) End-of-text marker is displayed.
------	--

False	End-of-text marker is not displayed.
-------	--------------------------------------

Data Type

Integer (Boolean)

CombHeight Property, Pen Edit Controls

Description

CombHeight sets or returns the height of the box in a box-style BEdit control.

Visual Basic

[*form*.]BEdit.CombHeight[= *setting*&]

Visual C++

pBEdit->GetNumProperty("CombHeight")
pBEdit->SetNumProperty("CombHeight", *setting*)

Remarks

The CombHeight property sets the height of the inner teeth of the comb. This setting is the height of the boxes when the BEdit control is in box style.

Note The value of the CombHeight and CombEndHeight properties cannot exceed the CombBaseLine value.

In Visual Basic, these properties use the ScaleMode property setting of the underlying form, frame, or picture box. For example, if a form's ScaleMode property is set to 3 (pixels), and a BEdit control is placed on that form, the CombEndHeight and CombHeight properties for that control are measured in pixels. In Visual C++, all the dimensions are in pixels.

Data Type

Long

CombNumCols, CombNumRows Properties, Pen Edit Controls

Description

Return the number of columns and rows displayed in a BEdit control. These properties are not available at design time and are read-only at run time.

Visual Basic

```
[form.]BEdit.CombNumCols  
[form.]BEdit.CombNumRows
```

Visual C++

```
pBEdit->GetNumProperty("CombNumCols")  
pBEdit->GetNumProperty("CombNumRows")
```

Remarks

The CombNumCols property returns the number of columns in a BEdit control. The CombNumRows property returns the number of rows. These settings are dependent on the dimensions of the control and on the CellHeight and CellWidth properties. CombNumRows will always be 1 if the MultiLine property is set to **False**.

Data Type

Integer

CombSpacing Property, Pen Edit Controls

Description

Sets or returns the distance between the side of the encapsulating cell walls and the side of the comb or box guide.

Visual Basic

```
[form.]BEdit.CombSpacing[ = setting&]
```

Visual C++

```
pBEdit->GetNumProperty("CombSpacing")  
pBEdit->SetNumProperty("CombSpacing", setting)
```

Remarks

The distance specified in the CombSpacing property is applied to both sides of the box and the comb guide area. CombSpacing cannot be greater than half the CellWidth.

This property uses the ScaleMode property setting of the underlying form, frame, or picture box. For example, if a form's ScaleMode property is set to 3 (pixels), and a BEdit control is placed on that form, the CombSpacing property for that control is measured in pixels. In Visual C++, all dimensions are in pixels.

Data Type

Long

CombStyle Property, Pen Edit Controls

Description

Sets or returns the BEdit style type.

Visual Basic

[*form*.]BEdit.CombStyle[= *setting*%]

Visual C++

pBEdit->GetNumProperty("CombStyle")
pBEdit->SetNumProperty("CombStyle", *setting*)

Remarks

The following table lists the CombStyle property settings for the pen edit controls.

Setting	Description
0	(Default) Comb style.
1	Box style.

Data Type

Integer (Enumerated)

DelayRecog Property, Pen Edit Controls

Description

Determines whether the control recognizes the user's writing immediately or leaves it as ink on the control for recognition at a later time.

Visual Basic

```
[form.]Penctrl.DelayRecog[ = {True | False}]
```

Visual C++

```
pPenctrl->GetNumProperty("DelayRecog")  
pPenctrl->SetNumProperty("DelayRecog", {TRUE | FALSE})
```

Remarks

The following table lists the DelayRecog property settings for the pen edit controls.

Setting	Description
---------	-------------

False	(Default) Recognition is not delayed.
--------------	---------------------------------------

True	Recognition is delayed; writing remains as ink.
-------------	---

When DelayRecog is set to **True**, all writing remains as ink on the control until the property is set to **False**. When it is changed from **True** to **False**, the OnTap property is examined. If OnTap is **True**, recognition of the collected ink takes place when the user taps on the control with the pen; otherwise, recognition occurs immediately.

When DelayRecog is set to **True**, the RcResult event occurs after the pen recognition time has elapsed. You can set the time-out for the entire system through the control panel.

Any writing performed on the control while DelayRecog is set to **False** is recognized according to the user preferences set in the control panel.

Data Type

Integer (Boolean)

EraseInk Property, Pen Edit Controls

Description

Setting the EraseInk property to **True** will erase any ink in a control if DelayRecog is **True**. This property is not available at design time.

Visual Basic

```
[form.]Penctrl.EraseInk[ = {True | False}]
```

Visual C++

```
pPenctrl->GetNumProperty("EraseInk")  
pPenctrl->SetNumProperty("EraseInk", {TRUE | FALSE})
```

Remarks

If a control has DelayRecog set to **True**, the action of setting EraseInk to **True** causes any ink in the control to be erased. The property setting reverts to **False** immediately after being set. It is used somewhat like a method rather than a property. Changing this property has no effect on a control if DelayRecog is set to **False**.

Data Type

Integer (Boolean)

hInk Property, Pen Edit Controls

Description

Returns a handle to an ink structure (**HPENDATA**) used by Microsoft Windows for Pen Computing. This property is not available at design time and is read-only at run time.

Visual Basic

[*form.*]Penctrl.hInk

Visual C++

pPenctrl->GetNumProperty("hInk")

Remarks

The operating environment provides the handle, which you can use to call Microsoft Windows for Pen Computing API functions that require a handle to a **PENDATA** structure. Refer to the *Microsoft Windows for Pen Computing Programmer's Reference* for details on handles to pen data.

Data Type

Integer

InflateBottom Property, Pen Edit Controls

Description

Sets or returns the area around a control's boundary onto which ink will be allowed.

Visual Basic

```
[form.]Penctrl.InflateBottom[ = setting&]
```

Visual C++

```
pPenctrl->GetNumProperty("InflateBottom")  
pPenctrl->SetNumProperty("InflateBottom", setting)
```

Remarks

This property defines a boundary around the control onto which ink can be placed after initially starting to draw ink within the control. This property must have a value greater than or equal to zero.

The settings are only applicable if the control is not in delayed-recognition mode (that is, the DelayRecog property is **False**). In delayed-recognition mode, ink cannot be drawn outside a control's boundary.

Data Type

Long

InflateLeft Property, Pen Edit Controls

Description

Sets or returns the area around a control's boundary onto which ink will be allowed.

Visual Basic

[form.]Penctrl.InflateLeft[= setting&]

Visual C++

pPenctrl->GetNumProperty("InflateLeft")
pPenctrl->SetNumProperty("InflateLeft", setting)

Remarks

This property defines a boundary around the control onto which ink can be placed after initially starting to draw ink within the control. This property must have a value greater than or equal to zero.

The settings are only applicable if the control is not in delayed-recognition mode (that is, the DelayRecog property is **False**). In delayed-recognition mode, ink cannot be drawn outside a control's boundary.

Data Type

Long

InflateRight Property, Pen Edit Controls

Description

Sets or returns the area around a control's boundary onto which ink will be allowed.

Visual Basic

[[form.]Penctrl.InflateRight[= setting&]

Visual C++

pPenctrl->GetNumProperty("InflateRight")
pPenctrl->SetNumProperty("InflateRight", setting)

Remarks

This property defines a boundary around the control onto which ink can be placed after initially starting to draw ink within the control. This property must have a value greater than or equal to zero.

The settings are only applicable if the control is not in delayed-recognition mode (that is, the DelayRecog property is **False**). In delayed-recognition mode, ink cannot be drawn outside a control's boundary.

Data Type

Long

InflateTop Property, Pen Edit Controls

Description

Sets or returns the area around a control's boundary onto which ink will be allowed.

Visual Basic

```
[form.]Penctrl.InflateTop[ = setting&]
```

Visual C++

```
pPenctrl->GetNumProperty("InflateTop")  
pPenctrl->SetNumProperty("InflateTop", setting)
```

Remarks

This property defines a boundary around the control onto which ink can be placed after initially starting to draw ink within the control. This property must have a value greater than or equal to zero.

The settings are only applicable if the control is not in delayed-recognition mode (that is, the DelayRecog property is **False**). In delayed-recognition mode, ink cannot be drawn outside a control's boundary.

Data Type

Long

InkColor Property, Pen Edit Controls

Description

Sets or returns the ink color.

Visual Basic

```
[form.]Penctrl.InkColor[ = color&]
```

Visual C++

```
pPenctrl->GetNumProperty("InkColor")  
pPenctrl->SetNumProperty("InkColor", color)
```

Remarks

The following table lists the InkColor property settings for the pen edit controls.

Setting	Description
&H8000000F& (color)	(Default) Pen ink color as set in the Control Panel for Microsoft Windows for Pen Computing. In Visual Basic, color specified by using the RGB scheme or the QBColor function in code. In Visual C++, use the MAKESYSCOLOR macro defined in the AFXEXT.H file to create system color constants. Standard RGB colors can be used in Visual C++ as well.

Data Type

Long

InkDataMode Property, Pen Edit Controls

Description

Sets or returns the mode that controls how the InkDataString data is used.

Visual Basic

```
[form.]Penctrl.InkDataMode[ = setting%]
```

Visual C++

```
pPenctrl->GetNumProperty("InkDataMode")  
pPenctrl->SetNumProperty("InkDataMode", setting)
```

Remarks

The following table lists the InkDataMode property settings for the pen edit controls.

Setting	Description
0	(Default) Replaces any ink that may be in the control when data is assigned to the InkDataString property.
1	Merges the currently displayed ink when ink data is assigned to the InkDataString property.

Data Type

Integer (Enumerated)

InkDataString Property, Pen Edit Controls

Description

Sets or returns a string containing the compressed ink data associated with the control. This property is only available at run time.

Visual Basic

```
[form.]Penctrl.InkDataString[ = inkdatastring$]
```

Visual C++

```
pPenctrl->GetStrProperty("InkDataString")  
pPenctrl->SetStrProperty("InkDataString", inkdatastring)
```

Remarks

When a string is assigned to a control with DelayRecog set to **True**, the InkDataMode is checked to determine whether the new ink data replaces the existing ink, or if the new ink is merged with the existing ink. Assigning invalid or uncompressed ink data to the InkDataString property generates a run-time error.

If the control has DelayRecog set to **False** and ink data is assigned to the control's InkDataString property, then the ink is immediately recognized as if it were written on the control. If InkDataMode is set to '0 - Replace' when this assignment is done, then any existing text in the control is replaced by the result from recognizing the ink. Otherwise, the new recognition result is appended to the text in the control.

When DelayRecog is set to **False** on a control, the InkDataString property is always a null string ("").

Data Type

String

InkWidth Property, Pen Edit Controls

Description

Sets or returns the width of the ink that is drawn.

Visual Basic

```
[form.]Penctrl.InkWidth[ = setting%]
```

Visual C++

```
pPenctrl->GetNumProperty("InkWidth")  
pPenctrl->SetNumProperty("InkWidth", setting)
```

Remarks

The InkWidth setting defaults to 1, which tells the control to use the system default ink width (as set using the Control Panel). The valid range for the InkWidth property is 0 to 15 pixels. No ink is displayed when InkWidth is set to 0.

Data Type

Integer

OnTap Property, Pen Edit Controls

Description

Determines whether the control recognizes the user's writing immediately upon changing the DelayRecog property from **True** to **False** or waits until the user taps the control.

Visual Basic

```
[form.]Penctrl.OnTap[ = {True | False}]
```

Visual C++

```
pPenctrl->GetNumProperty("OnTap")  
pPenctrl->SetNumProperty("OnTap", {TRUE | FALSE})
```

Remarks

The following table lists the OnTap property settings for the pen edit controls.

Setting	Description
False	(Default) Recognition is not delayed after DelayRecog is set to False .
True	Recognition is delayed until the user taps the control (after DelayRecog is set to False).

When DelayRecog is set to **False**, this property has no effect on the control. Its state is examined only when the DelayRecog property is changed from **True** to **False**.

Data Type

Integer (Boolean)

ScrollBars Property, Pen Edit Controls

Description

Determines if the control has horizontal or vertical scroll bars.

Visual Basic

[form.]Penctrl.ScrollBars [= *setting%*]

Visual C++

pPenctrl->GetNumProperty("ScrollBars")
pPenctrl->SetNumProperty("ScrollBars", setting)

Remarks

The following table lists the ScrollBars property settings for the pen edit controls.

Setting	Description
0	(Default) None
1	Horizontal (HEdit only)
2	Vertical
3	Both (HEdit only)

The settings for the ScrollBars property in a BEdit or HEdit control operate in the same manner as the settings for a standard Visual Basic text box control.

Horizontal scroll bars are not allowed on BEdit controls.

You can change the value of the ScrollBars property at run time.

Data Type

Integer (Enumerated)

RcResult Event, Pen Edit Controls

Description

Occurs whenever the control receives recognition results from the recognizer. The returned result can be used as a parameter to call the Pen API.

Visual Basic

Sub *Penctrl_RcResult* (*RcResult As Long*)

Visual C++

Function Signature:

void *CMyDialog::OnRcResultPenctrl* (**UINT, int, CWnd*, LPVOID lpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (LONG, lpParams)

Remarks

The RcResult event occurs whenever the recognizer returns any recognition results to the control. The *RcResult* parameter of this event is a long pointer to the **RCRESULT** structure that is returned by the recognizer to the control. The pointer to the **RCRESULT** structure is valid only during the processing of this event.

The **RCRESULT** structure contains a symbol graph that describes the possible results from the ink, a handle to the ink, and the best guess at the symbols. If DelayRecog is **True**, the ink remains on the screen. In this case, RcResult is still valid but does not contain any information about recognized symbols. For further information, consult the Microsoft Windows for Pen Computing SDK documentation.

The PENCNTRL.VBX file provides two exported functions that you can call from your Visual Basic program: **CPointerToVbType** and **VbTypeToCPointer**.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

Update Event, Pen Edit Controls

Description

Occurs whenever the data in a control is changed.

Visual Basic

Sub *Penctrl_Update* ()

Visual C++

Function Signature:

void *CMyDialog::OnUpdatePenctrl* (**UINT, int, CWnd*, LPVOID**)

Remarks

The Update event occurs before the control redraws the data. This differs from the Change event, which redraws the data before the event. You can use Update to format the new data so that flashes do not appear. An Update event does not occur when an application changes the text in the control using the Text property of the control.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

CPointerToVBType Function, Pen Edit Controls

Example

Description

Copies bytes from a system memory location to a Visual Basic variable memory location.

Visual Basic Syntax

CPointerToVBType(**ByVal** *lpSrc* **As Long**, *vbDest* **As Any**, **ByVal** *cNum* **By Integer**)

Remarks

Copies *cNum* number of bytes from a memory location pointed to by *lpSrc* and places them in the *vbDest* memory location. This function is useful for manipulating the *RcResult* parameter that is passed by the *RcResult* event.

CPointertoVBType Example, Pen Edit Controls

Visual Basic Example

The pen application in the \SAMPLES\PEN subdirectory uses the **CPointerToVBType** function to copy the returned *RcResult* parameter into a Visual Basic variable in order to modify the value:

```
Sub HEdit_RcResult (RcResult As Long)
    Dim VBRC As RcResult      ' RCRESULT Structure
    .
    .
    .
    Rem --- Get a copy of the RcResult Structure
    CPointerToVBType ByVal RcResult, VBRC, 80
```

VbTypeToCPointer Function, Pen Edit Controls

Example

Description

Copies bytes from a Visual Basic variable memory location to a system memory location.

Visual Basic Syntax

VbTypeToCPointer(*vbSrc* **As Any**, **ByVal** *lpDest* **As Long**, **ByVal** *cNum* **By Integer**)

Remarks

Copies *cNum* number of bytes from a memory location pointed to by *lpSrc* and places them in the *vbDest* memory location. This function is useful for manipulating the *RcResult* parameter that is passed by the *RcResult* event.

VBTypeToCPointer Example, Pen Edit Controls

Visual Basic Example

The pen application in the \SAMPLES\PEN subdirectory uses the **VBTypeToCPointer** function to copy the value of the modified Visual Basic variable into the memory location specified by the *RcResult* parameter:

```
Sub HEdit1_RcResult (RcResult As Long)
    Dim VBrc As RcResult      ' VB Copy of the RcResult Structure
    ... // Process RcResult value

    Rem --- Copy the Possibly modified RcResult back
    VBTypeToCPointer VBRC, ByVal RcResult, 80
```

Error Messages, Pen Edit Controls

The following table lists the trappable errors for the pen edit controls.

Error number	Message explanation
32001	PENERR_INKWIDTH InkWidth must be in range 0 = 15 or =1 for default. This error is caused by setting the InkWidth property to an invalid value.
32002	PENERR_NEGCELLWIDTH CellWidth has to be greater than 0. This error occurs if the CellWidth property is set to a value less than one.
32003	PENERR_CELLWIDTH CellWidth has to be greater than or equal to (CombSpacing * 2). This error occurs if the CellWidth property is set to a value that is less than two times the value of the CombSpacing property. Try setting CombSpacing to 0 before setting the CellWidth property.
32004	PENERR_NEGCELLHEIGHT CellHeight has to be greater than 0. This error occurs if the CellHeight property is set to a value less than zero.
32005	PENERR_CELLHEIGHT CellHeight has to be greater than or equal to CombBaseLine. This error occurs if the CellHeight property is set to a value less than the CombBaseLine property. Change the CombBaseLine property before changing the CellHeight property.
32006	PENERR_COMBSPACING CombSpacing out of range (0 = CellWidth / 2). This error occurs if the CombSpacing property is set to a value that is either less than zero or more than half the cell width.
32007	PENERR_COMBBASELINE CombBaseLine out of range (0 - CellHeight). This error occurs if the CombBaseLine property is set to a value that is either less than 0 or greater than the CellHeight property.
32009	PENERR_COMBHEIGHT CombHeight out of range (0 - CombBaseLine). This error occurs if the CombHeight property is set to a value that is either less than 0 or greater than the value of the CombBaseLine property.
32010	PENERR_COMBENDHEIGHT CombEndHeight out of range (0 - CombBaseLine). This error occurs if the CombEndHeight property is set to a value less than 0 or greater than the value of the CombBaseLine property.
32011	PENERR_INFLATE Inflate value has to be greater than or equal to 0. This error occurs if either InflateTop, InflateLeft, InflateRight, or InflateBottom property is set to a value less than zero.
32015	PENERR_MERGEFAILED Unable to merge ink data. The control was unable to merge new ink data with its existing ink data. This error can occur when trying to merge a large amount of ink data.
32016	PENERR_INVALIDINKDATA Invalid InkDataString format. This error can occur when trying to assign invalid or uncompressed ink data to the InkDataString property.

▪ Pen Ink-On-Bitmap Control

[Properties](#)

[Methods](#)

[Events](#)

[Error Messages](#)

Description

The pen ink-on-bitmap control is an enhanced picture box control that allows the user to draw and erase the ink on a bitmap. The *ink* refers to what you draw on top of the bitmap. You can also save the ink in three ways: as a bitmap, as compressed ink, or as a combined bitmap of the background image and the ink.

Note This control requires Microsoft Windows for Pen Computing.

In Visual Basic, the values of the error constants are in the PENAPI.TXT file in the \SAMPLES\ PEN subdirectory. In Visual C++, these values are in the CONSTANT.H file.

File Name

PENCNTRL.VBX

Object Type

InkOnBitmap

Distribution Note When you create and distribute applications that use the pen ink-on-bitmap control, you should install the file PENCNTRL.VBX 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 (*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*AutoSize</u>	<u>*hDC</u>	<u>*InkDataString</u>	TabIndex
BackColor	HelpContextID	<u>*InkingMode</u>	TabStop
BorderStyle	<u>*hInk</u>	<u>*InkPicture</u>	Tag
DragIcon	hWnd	<u>*InkWidth</u>	Top
DragMode	<u>*Image</u>	Left	Visible
Enabled	Index	Name	Width
<u>*EraseInk</u>	<u>*InkColor</u>	Parent	
Height	<u>*InkDataMode</u>	<u>*Picture</u>	

Picture 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 equivalent of the CtlName property in Visual Basic 1.0 and Visual C++.

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 (*). For documentation of the remaining events, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

DragDrop	GotFocus	<u>*IOBChange</u>	LostFocus
DragOver			

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	Move	Refresh	ZOrder
-------------	-------------	----------------	---------------

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

AutoSize Property, Pen Ink-On-Bitmap Control

Description

Determines the appearance of the background image in the pen ink-on-bitmap control.

Visual Basic

[*form.*]JOB.AutoSize[= *setting%*]

Visual C++

pIOB->GetNumProperty("AutoSize")
pIOB->SetNumProperty("AutoSize", *setting*)

Remarks

The following table lists the AutoSize property settings for the pen ink-on-bitmap control.

Setting	Description
0	(Default) No automatic sizing takes place, and the image is displayed in the upper-left corner of the control.
1	Automatically stretches the image to the size of the control.
2	Automatically adjusts the size of the control to exactly fit the bitmap.
3	Tiles the background image on the control.

Data Type

Integer (Enumerated)

EraseInk Property, Pen Ink-On-Bitmap Control

Description

Setting the EraseInk property to **True** erases any ink in the control. This property is not available at design time.

Visual Basic

```
[form.]JOB.EraseInk[ = {True | False}]
```

Visual C++

```
pJOB->GetNumProperty("EraseInk")  
pJOB->SetNumProperty("EraseInk", {TRUE | FALSE})
```

Remarks

The property setting reverts to **False** immediately after being set to **True**.

Data Type

Integer (Boolean)

hDC Property, Pen Ink-On-Bitmap Control

Description

Returns a handle provided by the operating environment to the device context of the pen ink-on-bitmap control. This property is not available at design time and is read-only at run time.

Visual Basic

*[form.]*IOB.hDC

Visual C++

*pIOB->*GetNumProperty("hDC")

Remarks

This property is a Microsoft Windows device-context handle. Use this handle when you need to pass an hDC value to Windows API function calls.

The device context is that of the combined bitmap image of the InkPicture picture and the Picture property.

Data Type

Integer

hInk Property, Pen Ink-On-Bitmap Control

Description

Returns a handle to an ink structure (**HPENDATA**) used by Microsoft Windows for Pen Computing. This property is not available at design time and is read-only at run time.

Visual Basic

[*form.*]JOB.hInk

Visual C++

pJOB->GetNumProperty("hInk")

Remarks

Use this handle when you need to make calls to Windows for Pen Computing API functions that require a handle to a **PENDATA** structure. Refer to the *Microsoft Windows for Pen Computing Programmer's Reference* for details on handles to pen data.

Data Type

Integer

Image Property, Pen Ink-On-Bitmap Control

Description

Returns a handle to a bitmap containing the combined Picture and InkPicture bitmaps. This property is not available at design time and is read-only at run time.

Visual Basic

[form.]JOB.Image

Visual C++

pJOB->GetNumProperty("Image")

Data Type

Integer

InkColor Property, Pen Ink-On-Bitmap Control

Description

Sets or returns the ink color.

Visual Basic

```
[form.]IOB.InkColor[ = color&]
```

Visual C++

```
pIOB->GetNumProperty("InkColor")  
pIOB->SetNumProperty("InkColor", color)
```

Remarks

The following table lists the InkColor property settings for the pen ink-on-bitmap control.

Setting	Description
&H8000000F(Default)	Pen ink color as set in the Control Panel.
(color)	In Visual Basic, color specified by using the RGB scheme or the QBColor function in code. In Visual C++, use the MAKESYSCOLOR macro defined in the AFXEXT.H file to create system color constants. Standard RGB colors can be used in Visual C++ as well.

The color of the *erase ink* that is displayed while erasing is the nearest solid color to the BackColor property of the control.

Data Type

Long

InkDataMode Property, Pen Ink-On-Bitmap Control

Description

Sets or returns the mode that controls how the InkDataString data is used.

Visual Basic

[*form.*]IOB.**InkDataMode**[= *setting%*]

Visual C++

pIOB->**GetNumProperty**("InkDataMode")
pIOB->**SetNumProperty**("InkDataMode", *setting*)

Remarks

The following table lists the InkDataMode property settings for the pen ink-on-bitmap control.

Setting	Description
0	(Default) Replaces any ink that may be in the control when data is assigned to the InkDataString property.
1	Merges the currently displayed ink when ink data is assigned to the InkDataString property.

Data Type

Integer (Enumerated)

InkDataString Property, Pen Ink-On-Bitmap Control

Description

Sets or returns a string containing the compressed ink data associated with the control. This property is only available at run time.

Visual Basic

```
[form.]JOB.InkDataString[ = inkdatastring]
```

Visual C++

```
pJOB->GetStrProperty("InkDataString")  
pJOB->SetStrProperty("InkDataString", inkdatastring)
```

Remarks

When a string is assigned to a control, the InkDataMode is checked to determine whether the new ink data replaces the existing ink, or if the new ink is merged with the existing ink. Assigning invalid or uncompressed ink data to the InkDataString property generates a run-time error.

Data Type

String

InkingMode Property, Pen Ink-On-Bitmap Control

Description

Sets or returns the currently selected inking mode of the pen ink-on-bitmap control.

Visual Basic

[form.]IOB.InkingMode[= setting%]

Visual C++

pIOB->GetNumProperty("InkingMode")
pIOB->SetNumProperty("InkingMode", setting)

Remarks

The following table lists the InkingMode property settings for the pen ink-on-bitmap control.

Setting	Description
0	Pen tip and barrel button disabled. No drawing or erasure takes place.
1	Pen tip draws ink. Barrel button is disabled.
2	Pen tip erases. Barrel button is disabled.
3	(Default) Pen tip draws ink. Barrel button, if pressed, sets pen tip to erase.

The color of the erase ink that is displayed while erasing is the nearest solid color to the BackColor property of the control.

Note This property is hardware-dependent ■ not all pen systems have a barrel button (or its equivalent).

Data Type

Integer (Enumerated)

InkPicture Property, Pen Ink-On-Bitmap Control

Description

Returns a handle to a bitmap containing the ink in the color specified by the InkColor property. This property is not available at design time and is read-only at run time.

Visual Basic

*[form.]*IOB.**InkPicture**

Visual C++

*pIOB->***GetProperty("InkPicture")**

Data Type

Integer

InkWidth Property, Pen Ink-On-Bitmap Control

Description

Sets or returns the width of the ink.

Visual Basic

[*form.*]JOB.InkWidth[= *setting%*]

Visual C++

pIOB->GetNumProperty("InkWidth")

pIOB->SetNumProperty("InkWidth", *setting*)

Remarks

The following table lists the InkWidth property settings for the pen ink-on-bitmap control.

Setting	Description
1	(Default) Sets the ink width to the default system ink width (as defined in the Control Panel).
0	No ink is displayed.
1 to 15	The range of valid, visible ink widths, in pixels.

Data Type

Integer

Picture Property, Pen Ink-On-Bitmap Control

Description

Specifies the graphic to be displayed as the background image on the pen ink-on-bitmap control. This property is write-only at design time.

Visual Basic

```
[form.]JOB.Picture[ = picture%]
```

Visual C++

```
pJOB->GetPictureProperty("Picture")  
pJOB->SetPictureProperty("Picture", picture)
```

Remarks

The following table lists the Picture property settings for the pen ink-on-bitmap control.

Setting	Description
(none)	(Default) No image is used in the background.
(Bitmap)	Designates that a bitmap is displayed in the background.
(Icon)	Designates that an icon is displayed in the background.

Data Type

Integer

IOBChange Event, Pen Ink-On-Bitmap Control

Description

Indicates that the contents of the pen ink-on-bitmap control have changed. This occurs if the Picture property is reassigned, or if inking or erasure occurs on the control.

Visual Basic

Sub *IOB_IOBChange* ()

Visual C++

Function Signature:

void *CMyDialog::OnChangeIOB* (UINT, int, CWnd*, LPVOID)

Remarks

The IOBChange event is generated after bitmaps are updated in the control.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

Error Messages, Pen Ink-On-Bitmap Control

The following table lists the trappable run-time errors for the pen ink-on-bitmap control.

Error number	Message explanation
32001	PENERR_INKWIDTH InkWidth must be in range 0 ■ 15 or

■ **1 for default.**

This error is caused by setting the InkWidth property to an invalid value.

32012 PENERR_INVALIDPICTURE

Picture format not supported.

This error is caused by setting the InkPicture property to an invalid value. The control can display only bitmap (.BMP) and icon (.ICO) format files.

32013 PENERR_DISPLAYFAILED

Unable to display bitmap.

The control is unable to display the bitmap. This error can be caused by low memory.

32014 PENERR_AUTOSIZEHW

Cannot change Height or Width while AutoSize equals 2.

When AutoSize is set to 2 (Adjust Window Size to Picture) on a control, its Height and Width properties cannot be modified.

32015 PENERR_MERGEFAILED

Unable to merge ink data.

The control was unable to merge new ink data with its existing ink data. This error can occur when trying to merge a large amount of ink data.

32016 PENERR_INVALIDINKDATA

Invalid InkDataString format.

This error can occur when trying to assign invalid or uncompressed ink data to the InkDataString property.

▪ Pen On-Screen Keyboard Control

[Properties](#)

[Methods](#)

[Events](#)

[Error Messages](#)

Description

The pen on-screen keyboard control allows you to directly access the Windows for Pen Computing on-screen keyboard without having to use the Windows for Pen Application Program Interface (API).

Note This control requires Microsoft Windows for Pen Computing.

This control provides a special command button that allows the user to display the on-screen keyboard. The user can make the on-screen keyboard appear and disappear by clicking the keyboard command button or by altering the value of the SKBVisible property of the control.

When the on-screen keyboard is activated, its output is sent to the control on the Visual Basic form that currently has the focus. The on-screen keyboard never gets the focus.

Certain keys on the keyboard are *sticky*; that is, when the button is pressed once, the button remains down until the user presses the button again. The SHIFT key is a sticky key.

In Visual Basic, the values of the error constants are in the PENAPI.TXT file in the \SAMPLES\ PEN subdirectory. In Visual C++, these values are in the CONSTANT.H file.

File Name

PENCNTRL.VBX

Object Type

SKBButton

Distribution Note When you create and distribute applications that use the pen on-screen keyboard control, you should install the file PENCNTRL.VBX 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 (*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*AutoSize</u>	Name	<u>*SKBType</u>
DragIcon	Parent	<u>*SKBTypeStatus</u>
DragMode	<u>*Picture</u>	<u>*SKBVisible</u>
Enabled	<u>*SKBLeft</u>	<u>*SKBVisibleStatus</u>
Height	<u>*SKBLeftStatus</u>	Tag
HelpContextID	<u>*SKBMin</u>	Top
hWnd	<u>*SKBMinStatus</u>	Visible
Index	<u>*SKBTop</u>	Width
Left	<u>*SKBTopStatus</u>	

Visible 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 equivalent of the CtlName property in Visual Basic 1.0 and Visual C++.

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 (*). For documentation of the remaining events, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

Click	DragDrop	DragOver	<u>*SKBChange</u>
-------	----------	----------	-------------------

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	Move	Refresh	ZOrder
-------------	-------------	----------------	---------------

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

AutoSize Property, Pen On-Screen Keyboard Control

Description

Determines the appearance of the bitmap on the keyboard command button.

Visual Basic

[*form*.]SKB.AutoSize[= *setting*%]

Visual C++

pSKB->GetNumProperty("AutoSize")

pSKB->SetNumProperty("AutoSize", *setting*)

Remarks

The following table lists the AutoSize property settings for the pen on-screen keyboard control.

Setting	Description
0	(Default) No automatic sizing takes place, and the bitmap is centered within the keyboard command button.
1	Automatically stretches the picture to the size of the keyboard command button.
2	Automatically adjusts the size of the keyboard command button to exactly fit the bitmap.

Data Type

Integer (Enumerated)

Picture Property, Pen On-Screen Keyboard Control

Description

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

Visual Basic

```
[form.]SKB.Picture[ = picture%]
```

Visual C++

```
pSKB->GetPictureProperty("Picture")  
pSKB->SetPictureProperty("Picture", picture)
```

Remarks

The following table lists the Picture property settings for the pen on-screen keyboard control.

Setting	Description
(none)	No image is displayed on the button.
(Bitmap)	(Default) A bitmap image is displayed on the button.
(Icon)	An icon image is displayed on the button.

If the Picture property is set to (Bitmap) and you do not specify a bitmap image, a bitmap of the keyboard is displayed on the button as the default image.

The keyboard command button automatically draws the button in the up, down, and disabled states.

Data Type

Integer

SKBLeft, SKBTop Properties, Pen On-Screen Keyboard Control

Description

SKBLeft determines the distance between the left edge of the on-screen keyboard and the left edge of the keyboard command button's container when the pen on-screen keyboard control is active.

SKBTop determines the distance between the top edge of the on-screen keyboard and the top edge of the keyboard command button's container when the pen on-screen keyboard control is active.

Visual Basic

```
[form.]SKB.SKBLeft[ = x&]  
[form.]SKB.SKBTop[ = y&]
```

Visual C++

```
pSKB->GetNumProperty("SKBLeft")  
pSKB->SetNumProperty("SKBLeft", x)  
pSKB->GetNumProperty("SKBTop")  
pSKB->SetNumProperty("SKBTop", y)
```

Remarks

Use SKBLeft and SKBTop to set the position of the on-screen keyboard's upper-left corner. The default value for the (x, y) position is (0, 0).

In Visual Basic, the units of measure for both properties are those of the underlying container (that is, the form, picture, or frame), and are measured relative to the upper-left corner of the container's window. In Visual C++, all dimensions are in pixels.

Data Type

Long

SKBLeftStatus, SKBTopStatus Properties, Pen On-Screen Keyboard Control

Description

SKBLeftStatus returns the current distance between the left edge of the on-screen keyboard and the left edge of the keyboard command button's container. This property is not available at design time and is read-only at run time.

SKBTopStatus returns the current distance between the top edge of the on-screen keyboard and the top edge of the keyboard command button's container. This property is not available at design time and is read-only at run time.

Visual Basic

[*form*].SKB.SKBLeftStatus

[*form*].SKB.SKBTopStatus

Visual C++

pSKB->GetNumProperty("SKBLeftStatus")

pSKB->GetNumProperty("SKBTopStatus")

Remarks

The SKBLeftStatus and SKBTopStatus properties reflect the current state of the on-screen keyboard. If the user moves the on-screen keyboard, these values are updated to reflect the keyboard's actual position.

Note If the on-screen keyboard is not visible, the values of these properties may not be valid.

In Visual Basic, the units of measure for both properties are those of the underlying container (that is, the form, picture, or frame), and are measured relative to the upper-left corner of the container's window. In Visual C++, all dimensions are in pixels.

Data Type

Long

SKBMin Property, Pen On-Screen Keyboard Control

Description

Determines whether the on-screen keyboard is minimized when the keyboard command button is activated.

Visual Basic

```
[form.]SKB.SKBMin[ = {True | False}]
```

Visual C++

```
pSKB->GetNumProperty("SKBMin")
```

```
pSKB->SetNumProperty("SKBMin", {TRUE | FALSE})
```

Remarks

The following table lists the SKBMin property settings for the pen on-screen keyboard control.

Setting	Description
True	The on-screen keyboard is minimized when the keyboard command button is activated.
False	(Default) The on-screen keyboard is not minimized when the keyboard command button is activated.

Data Type

Integer (Boolean)

SKBMinStatus Property, Pen On-Screen Keyboard Control

Description

Returns the current minimized state of the on-screen keyboard. This property is not available at design time and is read-only at run time.

Visual Basic

*[form.]*SKB.SKBMinStatus

Visual C++

*pSKB->*GetNumProperty("SKBMinStatus")

Remarks

The following table lists the SKBMinStatus property settings for the pen on-screen keyboard control.

Setting	Description
True	The on-screen keyboard is currently minimized.
False	The on-screen keyboard is not currently minimized.

Data Type

Integer (Boolean)

SKBType Property, Pen On-Screen Keyboard Control

Description

Determines the type of on-screen keyboard that is displayed when the keyboard command button is activated.

Visual Basic

```
[form.]SKB.SKBType[ = setting%]
```

Visual C++

```
pSKB->GetNumProperty("SKBType")  
pSKB->SetNumProperty("SKBType", setting)
```

Remarks

The following table lists the SKBType property settings for the pen on-screen keyboard control.

Setting	Description
0	(Default) A full keyboard is displayed.
1	A basic keyboard, without the numeric keypad, is displayed.
2	The numeric keypad is displayed.

Data Type

Integer (Enumerated)

SKBTypeStatus Property, Pen On-Screen Keyboard Control

Description

Returns the type of screen keyboard currently being displayed. This property is not available at design time and is read-only at run time.

Visual Basic

[*form*.]SKB.SKBTypeStatus

Visual C++

pSKB->GetNumProperty("SKBTypeStatus")

Remarks

The following table lists the SKBTypeStatus property settings for the pen on-screen keyboard control.

Setting	Description
0	(Default) A full keyboard is displayed.
1	A basic keyboard, without the numeric keypad, is displayed.
2	The numeric keypad is displayed.

Data Type

Integer (Enumerated)

SKBVisible Property, Pen On-Screen Keyboard Control

Description

Sets or returns the visual state of the on-screen keyboard that occurs when the keyboard command button is activated. This property is not available at design time.

Visual Basic

```
[form.]SKB.SKBVisible[ = {True | False}]
```

Visual C++

```
pSKB->GetNumProperty("SKBVisible")  
pSKB->SetNumProperty("SKBVisible", {TRUE | FALSE})
```

Remarks

The SKBVisible property returns or sets the state of the on-screen keyboard when the keyboard command button is activated. A **True** value indicates that the current keyboard command button is in control of the on-screen keyboard. If a **False** value is returned, the keyboard command button is currently not in control of the keyboard. By setting this property, control of the keyboard can be switched from one keyboard command button to another.

The following table lists the SKBVisible property settings for the pen on-screen keyboard control.

Setting	Description
True	The keyboard command button controls the on-screen keyboard, and it is visible.
False	(Default) The keyboard command button is not in control of the on-screen keyboard.

If all keyboard command buttons have SKBVisible = **False**, then the on-screen keyboard is not visible unless the keyboard is started by another application.

Note The pen on-screen keyboard control also supports the standard Visible property. However, this property only affects the keyboard command button's state of visibility.

Data Type

Integer (Boolean)

SKBVisibleStatus Property, Pen On-Screen Keyboard Control

Description

Returns the on-screen keyboard's current state of visibility. This property is not available at design time and is read-only at run time.

Visual Basic

[*form.*]SKB.SKBVisibleStatus

Visual C++

pSKB->GetNumProperty("SKBVisibleStatus")

Remarks

The following table lists the SKBVisibleStatus property settings for the pen on-screen keyboard control.

Setting	Description
---------	-------------

True	The on-screen keyboard is currently visible.
-------------	--

False	The on-screen keyboard is not currently visible.
--------------	--

If the on-screen keyboard is visible, then the SKBVisibleStatus property is **True** for all keyboard command buttons, regardless of which button controls the on-screen keyboard. In fact, if the on-screen keyboard is visible, the SKBVisibleStatus property is **True**, even if an entirely different application controls the on-screen keyboard.

Note If you hide the form that contains the keyboard command button currently in control of the on-screen keyboard, you do not affect the visibility state of the keyboard. However, if you terminate the application or unload the form that contains the keyboard command button currently in control of the on-screen keyboard, the keyboard disappears.

Data Type

Integer (Boolean)

SKBChange Event, Pen On-Screen Keyboard Control

Description

The SKBChange event occurs when you change the on-screen keyboard directly.

Visual Basic

Sub *SKB_SKBChange* (*ChangeCode* **As Integer**)

Visual C++

Function Signature:

void *CMyDialog::OnChangeSKB* (**UINT, int, CWnd*, LPVOID lpParams**)

Parameter Usage:

AFX_NUM_EVENTPARAM (**short, lpParams**)

Remarks

This event is generated when the user moves, minimizes, or closes the on-screen keyboard, or changes the type of keyboard being displayed. This event is only generated for the keyboard command button that currently controls the on-screen keyboard.

The *ChangeCode* parameter identifies which features have changed. If more than one change occurred, use the **OR** operator to combine values.

ChangeCode	Description
SKN_MINCHANGED	The on-screen keyboard has changed the minimized state.
SKN_PADCHANGED	The on-screen keyboard has changed keyboard types.
SKN_POSCHANGED	The on-screen keyboard has changed position.
SKN_TERMINATED	The on-screen keyboard has been closed.
SKN_VISCHANGED	The on-screen keyboard has changed its visible state.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

Error Messages, Pen On-Screen Keyboard Control

The following table lists the trappable run-time errors for the pen on-screen keyboard control.

Error number	Message explanation
32012	PENERR_INVALIDPICTURE Picture format not supported. This error is caused by setting the Picture property to an invalid value. The control can display only bitmap (.BMP) and icon (.ICO) format files.
32013	PENERR_DISPLAYFAILED Unable to display bitmap. The control is unable to display the bitmap. This error can be caused by low memory.
32014	PENERR_AUTOSIZEHW Cannot change Height or Width while AutoSize equals 2. When AutoSize is set to 2 (Adjust Window Size to Picture) on a control, its Height and Width properties cannot be modified.

Picture Clip Control

[Properties](#) [Error Messages](#)

Description

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

PICCLIP.VBX

Object Type

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 file PICCLIP.VBX 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 (*). For documentation on the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

<u>*Clip</u>	<u>*Cols</u>	Name	<u>*StretchY</u>
<u>*ClipHeight</u>	<u>*GraphicCell</u>	Parent	Tag
<u>*ClipWidth</u>	<u>*Height</u>	<u>*Picture</u>	<u>*Width</u>
<u>*ClipX</u>	hWnd	<u>*Rows</u>	
<u>*ClipY</u>	Index	<u>*StretchX</u>	

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 and Visual C++.

Clip Property, Picture Clip Control

Example

Description

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.

Visual Basic

[form.]PictureClip.Clip

Visual C++

pPictureClip->GetPictureProperty("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. In Visual C++, all dimensions are 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.

```
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

Description

Specifies the area of the picture clip control to be copied by the Clip property. This property is not available at design time.

Visual Basic

```
[form.]PictureClip.ClipHeight[ = Height%]  
[form.]PictureClip.ClipWidth[ = Width%]  
[form.]PictureClip.ClipX[ = X%]  
[form.]PictureClip.ClipY[ = Y%]
```

Visual C++

```
pPictureClip->GetNumProperty("ClipHeight")  
pPictureClip->SetNumProperty("ClipHeight", Height)
```

Remarks

This property is measured in pixels.

Data Type

Integer

ClipWidth Property, Picture Clip Control

Description

Specifies the area of the picture clip control to be copied by the Clip property. This property is not available at design time.

Visual Basic

```
[form.]PictureClip.ClipWidth[ = Width%]
```

Visual C++

```
pPictureClip->GetNumProperty("ClipWidth")  
pPictureClip->SetNumProperty("ClipWidth", Width)
```

Remarks

This property is measured in pixels.

Data Type

Integer

ClipX Property, Picture Clip Control

Description

Specifies the area of the picture clip control to be copied by the Clip property. This property is not available at design time.

Visual Basic

```
[form.]PictureClip.ClipX[ = X%]
```

Visual C++

```
pPictureClip->GetNumProperty("ClipX")  
pPictureClip->SetNumProperty("ClipX", X)
```

Remarks

This property is measured in pixels.

Data Type

Integer

ClipY Property, Picture Clip Control

Description

Specifies the area of the picture clip control to be copied by the Clip property. This property is not available at design time.

Visual Basic

```
[form.]PictureClip.ClipY[ = Y%]
```

Visual C++

```
pPictureClip->GetNumProperty("ClipY")  
pPictureClip->SetNumProperty("ClipY", Y)
```

Remarks

This property is measured in pixels.

Data Type

Integer

Cols, Rows Properties, Picture Clip Control

Description

Set or return the total number of columns or rows in the picture.

Visual Basic

```
[form.]PictureClip.Cols[ = cols%]  
[form.]PictureClip.Rows[ = rows%]
```

Visual C++

```
pPictureClip->GetNumProperty("Cols")  
pPictureClip->SetNumProperty("Cols", cols)  
pPictureClip->GetNumProperty("Rows")  
pPictureClip->SetNumProperty("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

Description

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.

Visual Basic

[form.]PictureClip.**GraphicCell** (*Index%*)

Visual C++

pPictureClip->**GetPictureProperty**("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

Description

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.

Visual Basic

```
[form.]PictureClip.Height  
[form.]PictureClip.Width
```

Visual C++

```
pPictureClip->GetNumProperty("Height")  
pPictureClip->GetNumProperty("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

Description

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

Description

Specify the target size for the bitmap created with the Clip property. These properties are not available at design time.

Visual Basic

```
[form.]PictureClip.StretchX[ = X%]  
[form.]PictureClip.StretchY[ = Y%]
```

Visual C++

```
pPictureClip->GetNumProperty("StretchX")  
pPictureClip->SetNumProperty("StretchX", X)  
pPictureClip->GetNumProperty("StretchY")  
pPictureClip->SetNumProperty("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. In Visual C++, all dimensions are in pixels.

Data Type

Integer

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 <i>index</i> 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 size or stretch property request.
32011	Clipboard already open.
32012	GetObject () Windows function failure. A call to the Windows function GetObject () failed.
32013	CreateCompatibleDC () Windows function failure. A call to the Windows function CreateCompatibleDC () 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 pixels).
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.

▪ Spin Button Control

[Properties](#)

[Methods](#)

[Events](#)

[Error Messages](#)

Description

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

SPIN.VBX

Object Type

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 file SPIN.VBX 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 (*). For documentation of the remaining properties, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

BackColor	ForeColor	MousePointer	Tag
<u>*BorderColor</u>	Height	Name	<u>*TdThickness</u>
<u>*BorderThickness</u>	HelpContextID	<u>*ShadeColor</u>	Top
<u>*Delay</u>	hWnd	<u>*ShadowBackColor</u>	Visible
DragIcon	Index	<u>*ShadowForeColor</u>	Width
DragMode	Left	<u>*ShadowThickness</u>	
Enabled	<u>*LightColor</u>	<u>*SpinOrientation</u>	

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 and Visual C++.

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 (*). For documentation of the remaining events, see Appendix A, "Standard Properties, Events, and Methods," in the *Custom Control Reference*.

DragDrop	DragOver	<u>*SpinDown</u>	<u>*SpinUp</u>
----------	----------	------------------	----------------

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	Move	Refresh	ZOrder
-------------	-------------	----------------	---------------

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

BorderColor Property, Spin Button Control

Description

Determines the color of the border drawn around the control.

Visual Basic

[*form.*]SpinButton.BorderColor [= *color*&]

Visual C++

pSpinButton->GetNumProperty("BorderColor")

pSpinButton->SetNumProperty("BorderColor", *color*)

Remarks

The following table lists the BorderColor property settings for the spin button control.

Setting	Description
&H00000000& (Default)	Black.
(<i>color</i>)	In Visual Basic, <i>color</i> specified by using the RGB scheme or the QBColor function in code. In Visual C++, use the MAKESYSCOLOR macro defined in the AFXEXT.H file to create system color constants. Standard RGB colors can be used in Visual C++ as well.

Data Type

Long

BorderThickness Property, Spin Button Control

Description

Sets the width of the border.

Visual Basic

[*form.*]SpinButton.**BorderThickness**[= *setting*%]

Visual C++

pSpinButton->**GetNumProperty**("BorderThickness")
pSpinButton->**SetNumProperty**("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

Description

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.

Visual Basic

[*form.*]SpinButton.**Delay** [= *setting%*]

Visual C++

pSpinButton->**GetNumProperty**("Delay")

pSpinButton->**SetNumProperty**("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

Description

Sets the color of a narrow margin located along the left and upper edges of the control.

Visual Basic

```
[form.]SpinButton.LightColor[ = color&]
```

Visual C++

```
pSpinButton->GetNumProperty("LightColor")  
pSpinButton->SetNumProperty("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. In Visual C++, use the MAKESYSCOLOR macro defined in the AFXEXT.H file to create system color constants. Standard RGB colors can be used in Visual C++ as well.

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

Description

Sets the color of a narrow margin that is located along the right and lower edges of the control.

Visual Basic

```
[form.]SpinButton.ShadeColor[ = color&]
```

Visual C++

```
pSpinButton->GetNumProperty("ShadeColor")  
pSpinButton->SetNumProperty("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. In Visual C++, use the MAKESYSCOLOR macro defined in the AFXEXT.H file to create system color constants. Standard RGB colors can be used in Visual C++ as well.

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

Description

Sets the background color for the shadow effect.

Visual Basic

```
[form.]SpinButton.ShadowBackColor[ = color&]
```

Visual C++

```
pSpinButton->GetNumProperty("ShadowBackColor")  
pSpinButton->SetNumProperty("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. In Visual C++, use the MAKESYSCOLOR macro defined in the AFXEXT.H file to create system color constants. Standard RGB colors can be used in Visual C++ as well.

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

Description

Sets the color of the shadow effect.

Visual Basic

```
[form.]SpinButton.ShadowForeColor[ = color&]
```

Visual C++

```
pSpinButton->GetNumProperty("ShadowForeColor")  
pSpinButton->SetNumProperty("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. In Visual C++, use the MAKESYSOLOR macro defined in the AFXEXT.H file to create system color constants. Standard RGB colors can be used in Visual C++ as well.

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

Description

Sets the width of the shadow effect.

Visual Basic

```
[form.]SpinButton.ShadowThickness[ = setting%]
```

Visual C++

```
pSpinButton->GetNumProperty("ShadowThickness")  
pSpinButton->SetNumProperty("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

Description

Sets the direction of the spin control arrows.

Visual Basic

[form.]SpinButton.SpinOrientation[= *setting%*]

Visual C++

pSpinButton->GetNumProperty("SpinOrientation")

pSpinButton->SetNumProperty("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

Description

Sets the width of the LightColor and the ShadeColor borders.

Visual Basic

```
[form.]SpinButton.TdThickness[ = setting%]
```

Visual C++

```
pSpinButton->GetNumProperty("TdThickness")  
pSpinButton->SetNumProperty("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

Description

Occur when the user clicks one of the arrows of a spin button.

Visual Basic

Sub *SpinButton_SpinUp* ()

Sub *SpinButton_SpinDown* ()

Visual C++

Function Signatures:

void *CMyDialog::OnSpinUpSpinButton* (UINT, int, CWnd*, LPVOID)

void *CMyDialog::OnSpinDownSpinButton* (UINT, int, CWnd*, LPVOID)

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.

For more information on using events in Visual C++, see Appendix B, "Using Custom Controls with Visual C++," in the *Custom Control Reference*.

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.

```
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

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
```

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. The Delay, BorderThickness, ShadowThickness, and TdThickness properties cannot be set to a negative value

Using Custom Controls with Visual C++

This appendix contains useful information for the Visual C++ programmer who is using custom controls. This appendix supplements the information in Chapter 17, "Programming with VBX Controls," of the Visual C++ *Class Library User's Guide*.

[CVBControl Class](#)

[Getting and Setting Properties](#)

[Using Events](#)

[Using Methods](#)

The CVBControl Class, Visual C++ Custom Controls

The **CVBControl** class is a special MFC 2.0 class that maps directly to custom control functionality. This class allows you to load controls, get control properties, and set control properties through public member functions. The **CVBControl** class also provides support for custom control events and methods. Refer to the *Visual C++ Class Library Reference* for more information.

Creating Controls

In both Visual Basic and Visual C++, you can create controls at design time by selecting a control from the Toolbox and placing it on the form or dialog. However, there is a major difference in how you dynamically create controls at run time.

In Visual Basic, you dynamically create controls at run time by *cloning* a control from a master copy of a control. The master copy of the control must be created at design time. In addition, all controls created in this way are part of a control array, meaning that every control has the same control name but a different control index. You reference a specific control in a control array as an array reference.

In Visual C++, you dynamically create controls at run time by using the **CVBControl::Create** member function. You do not need to have a master copy of a control. In addition, **CVBControl::Create** allows you to determine the container of a control by providing an argument that specifies the handle of the parent window.

Note Visual C++ does not support control arrays, as implemented in Visual Basic. You can, however, create your own array of controls and manipulate them yourself.

Getting and Setting Properties, Visual C++ Custom Controls

See Also

Use the property access member functions of the **CVBControl** class to get and set properties for custom controls. The data type of the control's property determines which one of the member functions you use. Refer to the Visual C++ *Class Library Reference* for more information on the property access member functions.

Getting Properties

To get a property from a custom control, use one of the following **CVBControl** member functions.

Member function	Description
------------------------	--------------------

GetFloatProperty	Gets the floating point value assigned to a floating point property.
-------------------------	--

GetNumProperty	Gets the integer value assigned to an integer-valued property.
-----------------------	--

GetPictureProperty	Gets a handle to the picture (HPIC) assigned to a picture property.
---------------------------	---

GetStrProperty	Gets the string assigned to a string property.
-----------------------	--

For example, if you wanted to return the Value property of a gauge control, you could write the following code:

```
int nValue = (int)pGauge->GetNumProperty("Value");
```

Setting Properties

To set a property for a custom control, use one of the following **CVBControl** member functions:

Member function	Description
------------------------	--------------------

SetFloatProperty	Sets a floating point property to the specified value.
-------------------------	--

SetNumProperty	Sets an integer-valued property to the specified value.
-----------------------	---

SetPictureProperty	Sets a picture property to the specified picture.
---------------------------	---

SetStrProperty	Sets a string property to the specified string.
-----------------------	---

For example, if you wanted to set the GraphTitle property of a graph control, you could write the following code:

```
pGraph->SetNumProperty("GraphTitle", lpszTitle);
```

See Also

[Accessing Picture Data](#)

[Getting Custom Control Error Values](#)

Accessing Picture Data

Some VBX controls have properties of the Picture type. Visual C++ provides three functions that allow you to manipulate Picture property values of custom controls. These functions are **AfxSetPict**, **AfxGetPict**, and **AfxReferencePict**. Refer to Technical Note 27: *Emulation Support for Visual Basic Custom Controls*, which you can access by selecting TN027 in the Visual C++ online Help file, MFCNOTES.HLP.

Getting Custom Control Error Values

The **CVBControl::m_nError** public data member variable holds the custom control error value when a property access member function generates an error. (The value of **m_nError** is equivalent to the Visual Basic run-time error code.)

The following table lists the general error code values that **m_nError** can have. Refer to CONSTANT.H for the error constants.

Error number	Message
---------------------	----------------

7	AFX_VBX_ERR_OUT_OF_MEMORY Out of memory.
61	AFX_VBX_ERR_DISK_FULL Disk full.
62	AFX_VBX_ERR_INPUT_PAST_EOF Input past end of file.
380	AFX_VBX_ERR_INVALID_PROP_VALUE Invalid property value.
420	AFX_VBX_ERR_INVALID_OBJ_REF Invalid object reference.
422	AFX_VBX_ERR_PROP_NOT_FOUND Property item not found.

Custom controls may also generate error codes that are specific to the control. Refer to the individual custom control description in the *Custom Control Reference* for any specific error codes.

Using Events, Visual C++ Custom Controls

See Also

In order to use custom control events, you need to provide a linkage between the control's events and message-handling functions in your application. Visual C++ provides the ClassWizard tool to help you link events to functions. You can use ClassWizard to:

- Create a member variable that points to the control. This pointer is placed in the declaration of the class (usually derived from **CDialog**) that contains the VBX control.
- Register events (messages) for the control.
- Create a message map that associates the control's events to the member functions that handle the events.
- Create message-handling functions for VBX control notification in your implementation file.

For more information on how to use ClassWizard with custom controls, refer to Chapter 17, "Programming with VBX Controls," of the Visual C++ *Class Library User's Guide*.

See Also

[Message Handling Functions](#)

[Accessing Event Parameters](#)

[Accessing the Control Pointer](#)

[Accessing the hWnd of a Custom Control](#)

Message-Handling Functions

When the ClassWizard creates a message-handling function template, the function is called with four arguments. For example, the DIALOG.CPP file in the VBCIRCLE sample application contains the following line:

```
void CCircleDialog::OnClickinCircle(UINT, int, CWnd*, LPVOID)
```

The four arguments that are passed to **OnClickinCircle** are not named since they are not used in the function itself. To use a function argument you need to name it. Normally, the first two arguments are not used. The following table lists the naming convention and the description for each argument:

Parameter	Type	Description
<i>uCode</i>	UINT	Notification code (normally not used).
<i>nIndex</i>	int	Index of event in event table (normally not used).
<i>pWnd</i>	CWnd*	Pointer to control.
<i>lpParams</i>	LPVOID	Event parameter structure.

Accessing Event Parameters

Some custom controls pass event parameters when an event is generated. For example, the multimedia MCI control passes a notification code when the Done event is generated. This means that the message-handling function needs to extract the parameters from the *lpParams* argument.

The Visual C++ AFXEXT.H file provides two macro functions that extract a data value from the *lpParams* argument. The first macro function, **AFX_NUM_EVENTPARAM**, requires the data type of the value you want to extract. The second macro function, **AFX_HLSTR_EVENTPARAM**, should only be used for extracting string data values.

```
#define AFX_NUM_EVENTPARAM(type, lpParams) \  
    (type FAR&) (**(type FAR* FAR*) lpParams)
```

```
#define AFX_HLSTR_EVENTPARAM(lpParams) \  
    (HLSTR FAR&) (*(HLSTR FAR*) lpParams)
```

The following table summarizes the usage of the two macro functions according to the data type of *lpParams*:

Data type of <i>lpParams</i>	Usage
-------------------------------------	--------------

ENUM	AFX_NUM_EVENTPARAM(BYTE, <i>lpParams</i>)
Boolean, Integer	AFX_NUM_EVENTPARAM(short, <i>lpParams</i>)
Long	AFX_NUM_EVENTPARAM(LONG, <i>lpParams</i>)
Single	AFX_NUM_EVENTPARAM(float, <i>lpParams</i>)
Picture	AFX_NUM_EVENTPARAM(HPIC, <i>lpParams</i>)
String	AFX_HLSTR_EVENTPARAM(<i>lpParams</i>)

For example, if you wanted to extract the notification code of the Done event that the multimedia control passes to the message-handling function, you could write:

```
void CVideoDialog::OnDoneMmcontrol1(UINT, int, CWnd*, LPVOID lpParams)  
{  
    LONG lCode = AFX_NUM_EVENTPARAM(LONG, lpParams);
```

Accessing the Control Pointer

The third argument passed to message-handling functions is the pointer to the VBX custom control that generated the event. If you give the argument a name (for example, *pWnd*), you can use any of the standard **CWnd** class functionality. However, since *pWnd* is actually a **CVBControl** class, you can typecast *pWnd* to be a **CVBControl** pointer.

```
void CCircleDialog::OnClickinCircle(UINT, int, CWnd* pWnd, LPVOID)
{
    CVBControl* pCircle = (CVBControl *)pWnd;
    LPCSTR lpClass = pCircle->GetVBXClass();
    ...
}
```

Accessing the hWnd of a Custom Control

You can access the window handle of the custom control several ways. The easiest way to access the window handle is through the public data member, **CWnd::m_hWnd**. This is possible because the **CVBControl** class is derived from the **CWnd** class.

For example, if you created a pointer to a VBX control and defined it as a data member of your class, the window handle of the custom control could be retrieved by writing:

```
HWND hWnd = m_control->m_hwnd;
```

You can also access the window handle of a custom control by using the **CWnd::GetSafeHwnd** function. The following example uses the *pWnd* argument, which points to the control that generated the event:

```
void CCircleDialog::OnClickinCircle(UINT, int, CWnd* pWnd, LPVOID)
{
    HWND hWnd = pWnd->GetSafeHwnd();
    .
    .
    .
}
```

Using Methods, Visual C++ Custom Controls

To use custom control methods in Visual C++, use the methods set of public member functions for the **CVBControl** class. This set of member functions consists of:

Method	Description
AddItem	Adds items to a list managed by a list box or combo box control.
Move	Moves a control to a specified location and resizes the control at the same time.
Refresh	Updates a control to reflect changes that have been made to the control or to the environment.
RemoveItem	Removes an item from a list managed by a control.

Refer to the Visual C++ *Class Library Reference* for more information on using these member functions.

