Visual Basic Reference

Visual Studio 6.0

CacheSize Property

See Also Example Applies To

Returns or sets the number of records that are cached in local memory for the current **DECommand** object.

Syntax

object.**CacheSize** [=*value*]

The **CacheSize** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
value	Integer. A numeric expression that specifies the number of records that are cached in local memory for the current DECommand object. The default is 10 records.

Remarks

The *value* tells the provider how many records to keep in its fat buffer and how many records to fetch at one time to local memory. For example, when set to 10, after opening the first Recordset the provider fetches the first 10 records into local memory. Therefore, as you move forward from the current record, the provider returns the data values from the local memory buffer. As soon as you move past the last record, the next 10 records are fetched from the data source (or cursor library) into the cache.

While you can change *value* during the life of the cursor, the change only affects the number of records in the cache after the next fetch from the data source (or local cursor library).

This property corresponds to the ADO Recordset CacheSize property.

Visual Basic for Applications Reference

Visual Studio 6.0

Calendar Property

See Also Example Applies To Specifics

Returns or sets a value specifying the type of calendar to use with your project.

You can use one of two settings for **Calendar**:

Setting	Value	Description
vbCalGreg	0	Use Gregorian calendar (default).
vbCalHijri	1	Use Hijri calendar.

Remarks

You can only set the **Calendar** property programmatically. For example, to use the Hijri calendar, use:

```
Calendar = vbCalHijri
```

Visual Basic: Windows Controls

Visual Studio 6.0

CalendarBackColor, CalendarForeColor Properties

See Also Example Applies To

Return or set a value that specifies the background or foreground color of the dropdown calendar portion of the control.

Syntax

object.**CalendarBackColor** [= *color*]

object.**CalendarForeColor** [= *color*]

The **CalendarBackColor** and **CalendarForeColor** property syntaxes have these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
color	A value or constant that determines the color to be used.

Settings

The settings for *color* are:

Setting	Description
RGB colors	Colors specified by using the Color palette or by using the RGB or QBColor functions in code.
System default colors	Colors specified by system color constants listed in the Visual Basic (VBRUN) object library in the Object Browser. The Windows operating environment substitutes the user's choices as specified in the Control Panel settings.

Remarks

The **CalendarBackColor and CalendarForeColor** properties can be used with the **CalendarTitleBackColor**, **CalendarTitleForeColor**, and **CalendarTrailingForeColor** properties to customize the colors of the control.

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte isn't 0, Visual Basic uses the system colors, as defined in the user's Control Panel settings and by constants listed in the Visual Basic (VBRUN) object library in the Object Browser.

Visual Basic: Windows Controls

CalendarBackColor, CalendarTitleForeColor Properties Example

The example changes the appearance of the **DateTimePicker** control's calendar by resetting several color properties. To try the example, place a **DateTimePicker** and a **CommandButton** control on a form. Paste the code into the Declarations section of the code module. Start the project and click the **DateTimePicker** control to see the default colors. Click the **CommandButton**, and click the **DateTimePicker** control again to see the new colors.

```
Private Sub Command1_Click()
   With DTPicker1
      .CalendarBackColor = vbYellow
      .CalendarTitleBackColor = vbRed
      .CalendarTitleForeColor = vbWhite
      .CalendarTrailingForeColor = vbGreen
   End With
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

CalendarTitleBackColor, CalendarTitleForeColor Properties

See Also Example Applies To

Return or set a value that specifies the background or foreground color of the title in the dropdown calendar portion of the control.

Syntax

object.CalendarTitleBackColor [= *color*]

object.CalendarTitleForeColor [= *color*]

The **CalendarTitleBackColor** and **CalendarTitleForeColor** property syntaxes have these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
color	A value or constant that determines the color to be used.

Settings

The settings for *color* are:

Setting	Description
RGB colors	Colors specified by using the Color palette or by using the RGB or QBColor functions in code.
System default colors	Colors specified by system color constants listed in the Visual Basic (VBRUN) object library in the Object Browser. The Windows operating environment substitutes the user's choices as specified in the Control Panel settings.

Remarks

The **CalendarTitleBackColor and CalendarTitleForeColor** properties can be used with the **CalendarBackColor**, **CalendarForeColor**, and **CalendarTrailingForeColor** properties to customize the colors of the control.

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte isn't 0, Visual Basic uses the system colors, as defined in the user's Control Panel settings and by constants listed in the Visual Basic (VBRUN) object library in the Object Browser.

Visual Basic: Windows Controls

Visual Studio 6.0

CalendarTrailingForeColor Property

See Also Example Applies To

Returns or sets a value that specifies the foreground color of trailing dates that are displayed in the dropdown calendar.

Syntax

object.**CalendarTrailingForeColor** [= *color*]

The **CalendarTrailingForeColor** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
color	A value or constant that determines the color of trailing dates dates that belong to the previous or following months as described in Settings.

Settings

The settings for *color* are:

Setting	Description
RGB colors	Colors specified by using the Color palette or by using the RGB or QBColor functions in code.
System default colors	Colors specified by system color constants listed in the Visual Basic (VBRUN) object library in the Object Browser. The Windows operating environment substitutes the user's choices as specified in the Control Panel settings.

Remarks

Trailing dates are day numbers that are displayed which precede and follow day numbers of the currently selected month in the dropdown calendar. By default, trailing dates are displayed in **vbWhite**.

The **CalendarTrailingForeColor** property can be used with the **CalendarBackColor**, **CalendarForeColor**, **CalendarTitleBackColor** and **CalendarTitleForeColor** properties to customize the colors of the control.

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte isn't 0, Visual Basic uses the system colors, as defined in the user's Control Panel settings and by constants listed in the Visual Basic (VBRUN) object library in the Object Browser.

Visual Basic Reference

Visual Studio 6.0

CallSyntax Property

See Also Example Applies To

Specifies the call syntax used to send the **CommandText** property to the ADO Command object.

Syntax

object.**CallSyntax** [=*string*]

The **CallSyntax** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
string	A string expression that specifies the call syntax of the DECommand object, including both source and parameter information.

Remarks

This property is generally used when the source of the **DECommand** object is either a stored procedure or a parameterized query.

Visual Basic Reference

Visual Studio 6.0

Cancel Property

See Also Example Applies To

Returns or sets a value indicating whether a command button is the Cancel button on a form. This command button can be the **CommandButton** control or any object within an **OLE** container control that behaves as a command button.

Syntax

object.**Cancel** [= *boolean*]

The **Cancel** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression specifying whether the object is the Cancel button, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	The **CommandButton** control is the Cancel button.
False	(Default) The **CommandButton** control isn't the Cancel button.

Remarks

Use the **Cancel** property to give the user the option of canceling uncommitted changes and returning the form to its previous state.

Only one **CommandButton** control on a form can be the Cancel button. When the **Cancel** property is set to **True** for one **CommandButton**, it's automatically set to **False** for all other **CommandButton** controls on the form. When a **CommandButton** control's **Cancel** property setting is **True** and the form is the active form, the user can choose the **CommandButton** by clicking it, pressing the ESC key, or pressing ENTER when the button has the focus.

For **OLE** container controls, the **Cancel** property is provided only for those objects that specifically behave as command buttons.

Tip For a form that supports irreversible operations, such as deletions, it's a good idea to make the Cancel button the default button. To do this, set both the **Cancel** property and the **Default** property to **True**.

Visual Basic: CommonDialog Control

Visual Studio 6.0

CancelError Property

See Also Example Applies To

Returns or sets a value indicating whether an error is generated when the user chooses the Cancel button.

Syntax

object.**CancelError** [= *boolean*]

The **CancelError** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression indicating whether an error is generated, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	An error is generated.
False	(Default) No error is generated.

Remarks

When this property is set to **True**, error number 32755 (**cdlCancel**) occurs whenever the user chooses the Cancel button.

Data Type

Boolean

Visual Basic: Multimedia MCI Control

Visual Studio 6.0

CanEject Property (Multimedia MCI Control)

See Also Example Applies To

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

Syntax

[*form.*]*MMControl.***CanEject**

Remarks

The following table lists the **CanEject** property settings for the **Multimedia MCI** control.

Setting	Description
False	(Default) The device cannot eject its media.
True	The device can eject its media.

The value of **CanEject** is retrieved using MCI_GETDEVCAPS during the processing of an **Open** command.

Data Type

Integer (Boolean)

Visual Basic Reference

Visual Studio 6.0

CanGetFocus Property

See Also Example Applies To

Returns or sets a value determining if a control can receive focus. The **CanGetFocus** property is read/write at the controls authoring time, and not available at the controls run time.

Settings

The settings for **CanGetFocus** are:

Setting	Description
True	(Default) The control can receive focus. If the control contains constituent controls, the control itself will be unable to receive the focus unless none of its constituent controls can receive the focus. It is up to the author of the control to write the code that draws a focus rectangle on the control when it does receive focus.
False	The control cannot receive focus.

Remarks

As long as the control contains at least one constituent control that has been set to receive the focus, **CanGetFocus** cannot be set to **False**. If **CanGetFocus** is **False**, then no constituent control can be set to receive the focus.

Visual Basic Reference

Visual Studio 6.0

CanGrow Property

See Also Example Applies To

Returns or sets a value that determines if the control can grow vertically when the returned text exceeds the preset size of the control.

Syntax

object.**CanGrow** [= *boolean*]

The **CanGrow** property syntax has these parts:

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
boolean	Optional. A Boolean expression that specifies if the control can grow, as shown in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	The control can grow.
False	(Default) The control cannot grow.

Visual Basic Extensibility Reference

Visual Studio 6.0

CanPaste Property

See Also Example Applies To

Returns a boolean value indicating whether or not the Clipboard contains appropriate information (that is, controls) for pasting to the form.

Syntax

object.**CanPaste**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Visual Basic: Multimedia MCI Control

Visual Studio 6.0

CanPlay Property (Multimedia MCI Control)

See Also Example Applies To

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

Syntax

[*form.*]*MMControl.***CanPlay**

Remarks

The following table lists the **CanPlay** property settings for the **Multimedia MCI** control.

Setting	Description
False	(Default) The device cannot play.
True	The device can play.

The value of **CanPlay** is retrieved using MCI_GETDEVCAPS during the processing of an **Open** command.

Data Type

Integer (Boolean)

Visual Basic: Multimedia MCI Control

Visual Studio 6.0

CanRecord Property (Multimedia MCI Control)

See Also Example Applies To

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

Syntax

[*form.*]*MMControl*.**CanRecord**

Remarks

The following table lists the **CanRecord** property settings for the **Multimedia MCI** control.

Setting	Description
False	(Default) The device cannot record.
True	The device can record.

The value of **CanRecord** is retrieved using MCI_GETDEVCAPS during the processing of an **Open** command.

Data Type

Integer (Boolean)

Visual Basic: Multimedia MCI Control

Visual Studio 6.0

CanStep Property (Multimedia MCI Control)

See Also Example Applies To

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

Syntax

[*form.*]*MMControl.***CanStep**

Remarks

The following table lists the **CanStep** property settings for the **Multimedia MCI** control.

Setting	Description
False	(Default) The device cannot step a frame at a time.
True	The device can step a frame at a time.

Currently only MMMovie, Overlay, and VCR MCI devices can step a frame at a time. Because there is no way to check whether a device can step, programs set the value of this property by checking if the device type is MMMovie, Overlay, or VCR during the processing of an **Open** command.

Data Type

Integer (Boolean)

Visual Studio 6.0

Visual Basic: MSChart Control

Cap Property

See Also Example Applies To

Returns or sets a value that determines how line ends are capped.

Syntax

object.**Cap** [= *type*]

The **Cap** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
type	Integer. A VtPenCap constant that describes the line pen cap style.

Visual Basic Reference

Visual Studio 6.0

Caption Property

See Also Example Applies To

- Form determines the text displayed in the **Form** or **MDIForm** object's title bar. When the form is minimized, this text is displayed below the form's icon.
- Control determines the text displayed in or next to a control.
- **MenuLine** object determines the text displayed for a **Menu** control or an object in the **MenuItems** collection.

For a **Menu** control, **Caption** is normally read/write at run time. But **Caption** is read-only for menus that are exposed or supplied by Visual Basic to add-ins, such as the **MenuLine** object.

Syntax

object.**Caption** [= *string*]

The **Caption** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list. If *object* is omitted, the form associated with the active form module is assumed to be *object*.
string	A string expression that evaluates to the text displayed as the caption.

Remarks

When you create a new object, its default caption is the default **Name** property setting. This default caption includes the object name and an integer, such as Command1 or Form1. For a more descriptive label, set the **Caption** property.

You can use the **Caption** property to assign an access key to a control. In the caption, include an ampersand (&) immediately preceding the character you want to designate as an access key. The character is underlined. Press the ALT key plus the underlined character to move the focus to that control. To include an ampersand in a caption without creating an access key, include two ampersands (&&). A single ampersand is displayed in the caption and no characters are underlined.

A **Label** controls caption size is unlimited. For forms and all other controls that have captions, the limit is 255 characters.

To display the caption for a form, set the **BorderStyle** property to either Fixed Single (1 or **vbFixedSingle**), Sizable (2 or **vbSizable**), or Fixed Dialog (3 or **vbFixedDialog**). A caption too long for the form's title bar is clipped. When an MDI child form is maximized within an **MDIForm** object, the child form's caption is included in the parent form's caption.

Tip For a label, set the **AutoSize** property to **True** to automatically resize the control to fit its caption.

Visual Basic Reference

Caption Property Example

This example changes the **Caption** property of a **CommandButton** control each time the user clicks the button. To try this example, paste the code into the Declarations section of a form containing a **CommandButton** named Command1, and then press F5 and click the button.

```
Private Sub Command1_Click ()
   ' Check caption, then change it.
   If Command1.Caption = "Clicked" Then
      Command1.Caption = "OK"
   Else
      Command1.Caption = "Clicked"
   End If
End Sub
```

Visual Basic Reference

Visual Studio 6.0

Caption Property (ActiveX Controls)

See Also Example Applies To

Returns or sets the caption of an object.

Syntax

object.**Caption** [= *string*]

The **Caption** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list. If *object* is omitted, the form associated with the active form moduleis assumed to be *object*.
string	A string expression that evaluates to the text displayed as the caption.

Remarks

You can use the **Caption** property to assign an access key to a control. In the caption, include an ampersand (&) immediately preceding the character you want to designate as an access key. The character is underlined. Press the ALT key plus the underlined character to move the focus to that control. To include an ampersand in a caption without creating an access key, include two ampersands (&&). A single ampersand is displayed in the caption and no characters are underlined.

Visual Basic: Windows Controls

Visual Studio 6.0

Caption Property (Band Object)

See Also Example Applies To

Returns or sets the caption that appears on a **Band** object in a **CoolBar** control.

Syntax

object.**Caption** [= *string*]

The **Caption** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a **Band** in the **Bands** collection of a **CoolBar** control.
string	A string expression that evaluates to the text displayed as the caption.

Remarks

The caption will be displayed left-aligned in between the move handle and the child control. When a Band is displayed vertically, the width of the band is dependent upon the width of the caption.

Visual Basic: DataGrid Control

Visual Studio 6.0

Caption Property (DataGrid Control, Column Object)

See Also Example Applies To

For a **DataGrid** control, this property determines the text displayed in the caption bar at the top of the grid.

For a **Column** object, this property determines the text displayed in the column's heading area.

Syntax

object.**Caption** [= *value*]

The **Caption** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A string expression that determines what is displayed, as described below.

Remarks

Setting the **Caption** property to an empty string for a **DataGrid** control hides its caption bar.

Setting the **Caption** property to an empty string for a **Column** object clears the text in the column's heading area but does not hide the heading. Column captions are only displayed if the **DataGrid** control's **ColumnHeaders** property is set to **True** and the **HeadLines** property is not set to 0.

Visual Basic Reference

Visual Studio 6.0

Caption Property (DEDesigner Extensibility)

See Also Example Applies To

Returns or sets the label, or caption control, that is set when a **DEAggregate** or **DEField** object is dragged from the Data Environment designer and dropped onto a Visual Basic form or report. To use the caption in a drag-and-drop operation, you must first specify that a caption will be created when the object is dropped.

- Form The text that displays in the title bar of the **Form** or **MDIForm** object. When the form is minimized, this text displays below the form's icon.
- Control The text that displays in or next to a control.
- **MenuLine** object The text that displays for a **Menu** control or an object in the **MenuItems** collection.
  Note For a **Menu** control, the **Caption** property is generally read/write at run time. However, the **Caption** property is read-only for menus that are exposed or supplied by Visual Basic to add-ins, such as the **MenuLine** object.

Syntax

object.**Caption** [=*string*]

The **Caption** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list. If *object* is omitted, the form associated with the active form module is assumed to be *object*.
string	A string expression that defines the textual caption that drops onto the form.

Remarks

When you create a new object, its default caption is the default **Name** property setting. This default caption includes the object name and an integer, such as Command1 or Form1. For a more descriptive label, set the **Caption** property.

You can use the **Caption** property to assign an access key to a control. In the caption, include an ampersand (&) immediately preceding the character you want to designate as an access key. The character is underlined. Press the ALT key plus the underlined character to move the focus to that control. To include an ampersand in a caption without creating an access key, include two ampersands (&&). A single ampersand is displayed in the caption and no characters are underlined.

The caption size of the **Label** control is unlimited. For forms and all other controls that have captions, the limit is 255 characters.

To display the caption for a form, set the **BorderStyle** property to either Fixed Single (1 or **vbFixedSingle**), Sizable (2 or **vbSizable**), or Fixed Double (3 or **vbFixedDouble**). If a caption is too long for the form's title bar, the caption is clipped. When an MDI child form is maximized within an **MDIForm** object, the caption of the child form is included in the caption of the parent form.

Tip For a label, set the **AutoSize** property to **True** to automatically resize the control to fit its caption.

Visual Basic: Windows Controls

Visual Studio 6.0

Caption Property (Tab Object)

See Also Example Applies To

Returns or sets the caption that appears on the tab or button of a **Tab** object in a **TabStrip** control.

Syntax

object.**Caption** [= *string*]

The **Caption** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a **Tab** object.
string	A string expression that evaluates to the text displayed as the caption.

Remarks

You can set the **Caption** property for a **Tab** object in the **TabStrip** control at design time or at run time.

- Design time On the Tab tab in the Properties Page of the **TabStrip** control, type the caption string in the Caption text box.
- Run time Set the caption as follows:

```
TabStrip1.Tabs(1).Caption = "First Tab"
```

-Or-

```
TabStrip1.Tabs.Add 2, , "Second Tab"
```

Visual Basic: Windows Controls

Caption Property (Tab Object) Example

This example sets the **Caption** property for each of three **Tab** objects it adds to a **TabStrip** control. The caption strings are "Time," "Date," and "Mail." Each **Tab** object also displays an image from an **ImageList** control. To try this example, place an **ImageList** and a **TabStrip** control on a form. Place three sample bitmaps in the **ImageList** control. The **ImageList** control supplies the images for the **Tab** objects. Paste the following code into the Load event of the Form object, and run the program.

```
Private Sub Form_Load()
   Dim X As Integer
   ' Associate an ImageList with the TabStrip control.
   Set TabStrip1.ImageList = ImageList1
   ' Set the captions.
   TabStrip1.Tabs(1).Caption = "Time"
   TabStrip1.Tabs.Add 2, , "Date"
   TabStrip1.Tabs.Add 3, , "Mail"
   For X = 1 To TabStrip1.Tabs.Count
      ' Associate an image with a tab.
      TabStrip1.Tabs(X).Image = X
   Next X
End Sub
```

Visual Basic Extensibility Reference

Visual Studio 6.0

Caption Property

See Also Example Applies To Specifics

Returns a String containing the title of the active window. Read-only.

Remarks

The title of the active window is the text displayed in the window's title bar.

Visual Basic Extensibility Reference

Caption Property Example

The following example uses the **Caption** property to display the caption of the active window.

```
Debug.Print Application.VBE.ActiveWindow.Caption
```

Visual Basic: DataRepeater Control

Visual Studio 6.0

CaptionStyle Property

See Also Example Applies To

Returns or sets a value that determines the style of the caption, including alignment and visibility.

Syntax

object.**CaptionStyle** [=*integer*]

The **CaptionStyle** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
integer	A numeric expression that determines the alignment of the caption, as shown in Settings.

Settings

Constant	Value	Description
drpNoCaption	0	The caption is hidden.
drpLeftAligned	1	(Default) The caption is left aligned.
drpRightAligned	2	The caption is right aligned.
drpCentered	3	The caption is centered.

Visual Basic Reference

Visual Studio 6.0

Category Property

See Also Example Applies To

Returns or sets the Category attribute associated with a **Member** object.

Syntax

object.**Category**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Visual Basic: MSChart Control

CategoryScale Property

See Also Example Applies To

Returns a reference to a **CategoryScale** object that describes the scale information for a category axis.

Syntax

object.**CategoryScale**

The object placeholder represents an object expression that evaluates to an object in the Applies To list.

Visual Basic Reference

Visual Studio 6.0

CausesValidation Property

See Also Example Applies To

Returns or sets a value that determines whether the Validate event will occur on a second control from which the second control's focus is being shifted.

Syntax

object.**CausesValidation** [= *boolean*]

The **CausesValidation** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression that specifies whether the control from which the focus is being shifted fires the Validate event.

Settings

The settings for *boolean* are:

Setting	Description
True	(Default) The control from which the focus has shifted fires its Validate event.
False	The control from which the focus has shifted does not fire its Validate event.

Remarks

The **CausesValidation** property works in tandem with the Validate event to limit when a control can lose focus.

Visual Basic Reference

Validate Event, CausesValidation Property Example

The example uses three controls to demonstrate the use of the Validate event and **CausesValidation** property. By default, the **CausesValidation** property of the two TextBox controls are set to **True**. Thus when you try to shift the focus from one TextBox to the other, the Validate event occurs. If Text1 doesn't contain a date, or if Text2 doesn't contain a number larger than 10, the shift of focus is prevented. Because the **CausesValidation** property of the Command1 control is set to **False**, however, you can always click the Help button.

To try the example, place one **CommandButton** and two **TextBox** controls on a form. Paste the code into the Declarations section of the form and run the project. Attempt to shift the focus by pressing the Tab key.

```
Private Sub Form_Load()
   ' Set the button's CausesValidation property to False. When the user
   ' clicks the button, the Validate event does not occur.
   ' Set the Caption property of the button to "Help".
   With Command1
      .CausesValidation = False
      .Caption = "Help"
   End With

   Show
   With Text1 ' Select text of Text1 and set focus to it.
      .SelLength = Len(Text1.Text)
      .SetFocus
   End With
End Sub

Private Sub Command1_Click()
   ' Give the user help when the button is clicked.
   MsgBox _
   "Text1 must be set to a date." & VbCrLF & _
   "Text2 must be a number less than 10."
End Sub

Private Sub Text1_Validate(KeepFocus As Boolean)
   ' If the value is not a date, keep the focus, unless the user
   ' clicks Help.
   If Not IsDate(Text1.Text) Then
      KeepFocus = True
      MsgBox "Please insert a date in this field.", ,   "Text1"
   End if
End Sub

Private Sub Text2_Validate(KeepFocus As Boolean)
   ' If the value is a number larger than 10, keep the focus.
   If Not IsNumeric(Text2.Text) Or Val(Text2.Text) > 10 Then
      KeepFocus = True
MsgBox _
"Please insert a number less than or equal to 10.", , "Text2"
```

```
    End If
End Sub
```

Visual Basic: MSComm Control

Visual Studio 6.0

CDHolding Property

See Also Example Applies To

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

Syntax

object.**CDHolding**

The **CDHolding** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Settings

The settings for the **CDHolding** property are:

Setting	Description
True	Carrier Detect line is high
False	Carrier Detect line is low

Remarks

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 (drop the carrier) at any time.

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

Data Type

Boolean

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellAlignment Property

See Also Example Applies To

Returns or sets a value that determines the horizontal and vertical alignment of data within the current cell. This property is not available at design time.

Syntax

object.**CellAlignment** [=*value*]

The **CellAlignment** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer or constant that specifies how text should be aligned within the current cell, as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
flexAlignLeftTop	0	The cell content is aligned left, top.
flexAlignLeftCenter	1	Default for strings. The cell content is aligned left, center.
flexAlignLeftBottom	2	The cell content is aligned left, bottom.
flexAlignCenterTop	3	The cell content is aligned center, top.
flexAlignCenterCenter	4	The cell content is aligned center, center.
flexAlignCenterBottom	5	The cell content is aligned center, bottom.
flexAlignRightTop	6	The cell content is aligned right, top.

flexAlignRightCenter	7	Default for numbers. The cell content is aligned right, center.
flexAlignRightBottom	8	The cell content is aligned right, bottom.
flexAlignGeneral	9	The cell content is of general alignment. This is "left, center" for strings and "right, center" for numbers.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

CellAlignment Property Example

The following example sets the text alignment for each cell to left center using the constant setting.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub Form1_Load ()
   MSHFlexGrid1.CellAlignment =flexAlignLeftCenter
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellBackColor, CellForeColor Properties

See Also Example Applies To

- **CellBackColor** Returns or sets the background colors of individual cells or cell ranges.
- **CellForeColor** Returns or sets the foreground colors of individual cells or cell ranges.

These properties are not available at design time.

Syntax

object.**CellBackColor** [=*color*]
object.**CellForeColor** [=*color*]

Syntax for the **CellBackColor** and **CellForeColor** properties has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
color	Integer (enumerated). A numeric expression that specifies the color for the current cell selection. Setting either of these properties to zero paints the cell using standard background and foreground colors.

Remarks

Changing this property affects the current cell or the current selection, depending on the setting of the **FillStyle** property.

Setting either of these properties to zero causes **MSHFlexGrid** to paint the cell using the standard background and foreground colors. If you want to set either of these properties to black, set them to one instead of zero.

To set the colors of various **MSHFlexGrid** elements, use the **BackColorBkg**, **BackColorFixed**, **BackColorSel**, **ForeColorFixed**, and **ForeColorSel** properties. To set all non-fixed cells to the same background color, use the **BackColor** property.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellFontBold Property

See Also Example Applies To

Returns or sets the bold style for the current cell text. This property is not available at design time.

Syntax

object.**CellFontBold** [=*Boolean*]

The **CellFontBold** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Boolean	A Boolean expression that specifies whether the current cell's text is bold.

Settings

The settings for *Boolean* are:

Setting	Description
True	The current cell text is bold.
False	Default. The current cell text is normal (not bold).

Remarks

Changing this property affects the current cell or the current selection, depending on the setting of the **FillStyle** property.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

CellFontBold Property Example

The following code sets the text of the current cell to bold whenever the **MSHFlexGrid** is in focus.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub MSHFlexGrid1_GotFocus()
   MSHFlexGrid1.CellFontBold =1
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellFontItalic Property

See Also Example Applies To

Returns or sets the italic style for the current cell text. This property is not available at design time.

Syntax

object.**CellFontItalic** [=*Boolean*]

The **CellFontItalic** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Boolean	A Boolean expression that specifies whether the text style in the cell is italic.

Settings

The settings for *Boolean* are:

Setting	Description
True	The current cell text is italic.
False	Default. The current cell text is normal (not italic).

Remarks

Changing this property affects the current cell or the current selection, depending on the setting of the **FillStyle** property.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

CellFontItalic Property Example

The following code sets the text of the current cell to Italic whenever the **MSHFlexGrid** is in focus.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub MSHFlexGrid1_GotFocus()
   MSHFlexGrid1.CellFontItalic =True
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellFontName Property

See Also Example Applies To

Returns or sets the name of the font for the current cell text. This property is not available at design time.

Syntax

object.**CellFontName** [=*string*]

The **CellFontName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
string	A string expression naming one of the available font faces.

Remarks

Changing this property affects the current cell or the current selection, depending on the setting of the **FillStyle** property.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

CellFontName Property Example

The following code sets the text of the current cell to a specific font type whenever the **MSHFlexGrid** is in focus.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub MSHFlexGrid1_GotFocus()
   MSHFlexGrid1.CellFontName =Screen.Fonts(3)
   MSHFlexGrid1.Text =Screen.Fonts(3)    ' Displays font
                                        ' name.
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellFontSize Property

See Also Example Applies To

Returns or sets the size, in points, for the current cell text. This property is not available at design time.

Syntax

object.**CellFontSize** [=*value*]

The **CellFontSize** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	Single. A numeric expression that specifies the size of the current cell text.

Remarks

Changing this property affects the current cell or the current selection, depending on the setting of the **FillStyle** property.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

CellFontSize Property Example

The following code sets the text of the current cell to 12 points whenever the **MSHFlexGrid** is in focus.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub MSHFlexGrid1_GotFocus()
   MSHFlexGrid1.CellFontSize =12
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellFontStrikeThrough Property

See Also Example Applies To

Returns or sets a value that determines if the Strikethrough style is applied to the current cell text.

Syntax

object.**CellFontStrikeThrough** =[*Boolean*]

The **CellFontStrikeThrough** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Boolean	A Boolean expression that specifies if the strike-through style is applied to the cell text.

Settings

The settings for *Boolean* are:

Setting	Description
True	The strikethrough style is applied to the cell text.
False	Default. The strikethrough style is not applied to the cell text.

Remarks

Changing the **CellFontStrikeThrough** property affects the current cell or the current selection, depending on the setting of the **FillStyle** property.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

CellFontStrikeThrough Property Example

The following code sets the text of the current cell to the strikethrough style whenever the **MSHFlexGrid** is in focus.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub MSHFlexGrid1_GotFocus
   MSHFlexGrid1.CellFontStrikeThrough =1
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellFontUnderline Property

SeeAlso Example Applies To

Returns or sets a value that specifies if the underline style is applied to the current cell text.

Syntax

object.**CellFontUnderline** [=*Boolean*]

The **CellFontUnderline** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Boolean	A Boolean expression that specifies if the underline style is applied to the cell text.

Settings

The settings for *Boolean* are:

Setting	Description
True	The underline style is applied to the cell text.
False	Default. The underline style is not applied to the cell text.

Remarks

Changing the **CellFontUnderline** property affects the current cell or the current selection, depending on the setting of the **FillStyle** property.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

CellFontUnderline Property Example

The following code sets the text of the current cell to Underline whenever the **MSHFlexGrid** is in focus.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub MSHFlexGrid1_GotFocus
   MSHFlexGrid1.CellFontUnderline =1
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellFontWidth Property

See Also Example Applies To

Returns or sets the width, in points, for the current cell text. This property is not available at design time.

Syntax

object.**CellFontWidth** [=*value*]

The **CellFontWidth** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	Single. A numeric expression that specifies the preferred point width for the current cells font.

Remarks

Changing the **CellFontWidth** property affects the current cell or the current selection, depending on the setting of the **FillStyle** property.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

CellFontWidth Property Example

The following code sets the width of the text of the current cell when the **MSHFlexGrid** is in focus.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub MSHFlexGrid1_GotFocus()
   MSHFlexGrid1.CellFontWidth =5
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellHeight, CellLeft, CellTop, CellWidth Properties (MSHFlexGrid)

See Also Example Applies To

Returns the position and size of the current cell, in twips. These properties are not available at design time.

Syntax

object.**CellHeight** [=*value*]
object.**CellLeft** [=*value*]
object.**CellTop** [=*value*]
object.**CellWidth** [=*value*]

Syntax for the **CellHeight**, **CellLeft**, **CellTop**, and **CellWidth** properties has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A numeric expression that specifies the return position or size of the current cell.

Remarks

These properties are useful if you want to emulate in-cell editing. By trapping the MSHFlexGrid's **KeyPress** event, you can place a text box or some other control over the current cell and let the user edit its contents.

The return values are always in twips, regardless of the forms **ScaleMode** setting.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellPicture Property

See Also Example Applies To

Returns or sets an image to be displayed in the current cell or range of cells. This property is not available at design time.

Syntax

object.**CellPicture** [=*picture*]

The **CellPicture** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
picture	A bitmap, icon, or metafile graphic. This can be assigned to another controls **Picture** property.

Remarks

You can set this property at run time using the **LoadPicture** function on a bitmap, icon, or metafile, or by assigning to it another controls **Picture** property.

Changing this property affects the current cell or the current selection, depending on the setting of the **FillStyle** property.

Each cell may contain text and a picture. The relative position of the text and picture is determined by the **CellAlignment** and **CellPictureAlignment** properties.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

CellPicture Property Example

The following example loads icons from the Visual Basic icon library into two cells within an **MSHFlexGrid**. You can use any two icons. Paste the code into the Declarations section of a form that contains the **MSHFlexGrid**. Press F5 to run the program, and then click the form.

```
Private Sub Form_Click ()
   ' Load the icons.
   MSHFlexGrid1.Row =1
   MSHFlexGrid1.Col =1
   Set MSHFlexGrid1.CellPicture =_
   LoadPicture("Icons\Computer\Trash02a.ico")
   MSHFlexGrid1.Row =1
   MSHFlexGrid1.Col =2
   Set MSHFlexGrid1.CellPicture =_
   LoadPicture("Icons\Computer\Trash02b.ico")
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellPictureAlignment Property

See Also Example Applies To

Returns or sets the alignment of pictures in a cell or range of selected cells. This property is not available at design time.

Syntax

object.**CellPictureAlignment** [=*value*]

The **CellPictureAlignment** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer or constant that specifies how pictures should be aligned within cells, as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
flexAlignLeftTop	0	The picture is aligned left, top.
flexAlignLeftCenter	1	The picture is aligned left, center.
flexAlignLeftBottom	2	The picture is aligned left, bottom.
flexAlignCenterTop	3	The picture is aligned center, top.
flexAlignCenterCenter	4	The picture is aligned center, center.
flexAlignCenterBottom	5	The picture is aligned center, bottom.
flexAlignRightTop	6	The picture is aligned right, top.
flexAlignRightCenter	7	The picture is aligned right, center.

flexAlignRightBottom	8	The picture is aligned right, bottom.

Remarks

Changing this property affects the current cell or the current selection, depending on the setting of the **FillStyle** property.

The **FillStyle** property must be set to 1 (Repeat) for CellAlignment to align a range of selected cells within the **MSHFlexGrid**.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

CellPictureAlignment Property Example

The following code sets the picture alignment of the current cell to right center using the constant value.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub Form1_Load
   MSHFlexGrid1.CellPictureAlignment =_
   flexAlignRightCenter
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellTextStyle Property

See Also Example Applies To

Returns or sets the three-dimensional style for text in a specific cell or range of cells. This property is not available at design time.

Syntax

object.**CellTextStyle** [=*value*]

The **CellTextStyle** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer or constant that specifies one of the constants of the **CellTextStyle** property, as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
flexTextFlat	0	Default. The text is normal, flat text.
flexTextRaised	1	The text appears raised.
flexTextInset	2	The text appears inset.
flexTextRaisedLight	3	The text appears slightly raised.
flexTextInsetLight	4	The text appears slightly inset.

Remarks

Settings 1 and 2 work best for large and bold fonts. Settings 3 and 4 work best for small regular fonts. The cells appearance is also affected by the **BackColor** settings; some **BackColor** settings do not show the raised or inset feature.

Changing this property affects the current cell or the current selection, depending on the setting of the **FillStyle** property.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

CellTextStyle Property Example

The following code sets the text style of the current cell or current selection to inset using the constant value.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub MSHFlexGrid1_GotFocus
   MSHFlexGrid1.CellTextStyle =flexTextInset
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

CellType Property (MSHFlexGrid)

SeeAlso Example Applies To

Returns the type of the current, active, cell of the **MSHFlexGrid**.

Syntax

object.**CellType** [=*value*]

The **CellType** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A Long value that specifies the position of the active cell, as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
flexCellTypeStandard	0	Default. The cell is a standard cell.
flexCellTypeFixed	1	The cell is contained in a fixed row or column.
flexCellTypeHeader	2	The cell is a header cell for a band of data.
flexCellTypeIndent	3	The cell is used in a column that indents a band of data.
flexCellTypeUnpopulated	4	The cell is an unpopulated cell.

Visual Basic: Windows Controls

Visual Studio 6.0

Center Property

See Also Example Applies To

Determines whether the .avi file is centered within the **Animation** control. When set to **True** (the default), the .avi is displayed in the center of the control, based on the size of the image. When set to **False**, the .avi is positioned at 0,0 within the control.

Syntax

object.**Center** [= *boolean*]

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
boolean	Required. A Boolean expression specifying whether the AVI file is centered within the control.

Data Type

Integer (Boolean)

Remarks

If the **Center** property is set to True, and the .avi frame size is larger than the control, the edges of the .avi frames will not be shown as the file plays.

Visual Basic Reference

Visual Studio 6.0

Changed Property

See Also Example Applies To

Returns or sets a value indicating that a value of a property on a property page has changed. The **Changed** property is not available at the property pages authoring time, and read/write at the property pages run time.

Syntax

object.**Changed** [= *boolean*]

The **Changed** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A boolean value that determines if a property on the property page has been changed, making the property page dirty.

Settings

The settings for *boolean* are:

Setting	Description
True	The property page is now dirty, since the value of a property on the page has been changed.
False	The property page is not dirty, and no properties on the page have had their value changed.

Remarks

When the user changes the value of properties on a property page, these changes should not take effect immediately; instead, the changes should be applied only if the user presses the **Apply** button, the **OK** button, or changes property pages by selecting tabs. This allows the user to easily back out of any changes that have been made to a property page.

The **Changed** property should be set to **True**, for example, when a user changes a property value on a property page. Setting the **Changed** property to **True** would notify the property page to make available the **Apply** button.

Visual Basic Reference

Visual Studio 6.0

Charset Property

See Also Example Applies To

Sets or returns the character set used in the font.

Syntax

object.**Charset** [= *value*]

The **Charset** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A integer value that specifies the character set used by the font, as described in Settings.

Settings

The following are some common settings for *value*:

Value	Description
0	Standard Windows characters
2	The symbol character set.
128	Double-byte character set (DBCS) unique to the Japanese version of Windows
255	Extended characters normally displayed by DOS applications.

Remarks

Setting the **Charset** property to one of its available values selects the character set only if it is available in the current font.

Visual Studio 6.0

Visual Basic: MSChart Control

Chart3d Property

See Also Example Applies To

Returns a value that specifies whether or not a chart is three dimensional.

Syntax

object.**Chart3D**

The object placeholder represents an object expression that evaluates to an object in the Applies To list.

Return Values

Setting	Description
True	The chart is a three-dimensional chart.
False	The chart is not a three-dimensional chart.

Remarks

Whether or not a chart is three-dimensional depends on the **ChartType** property's setting.

Visual Basic: MSChart Control

ChartData Property

See Also Example Applies To

Returns or sets an array that contains the values that will be displayed by the chart.

Syntax

object.**ChartData** [= *data*]

The **ChartData** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
data	Variant. A two-dimensional array that holds the data used to draw the chart.

Remarks

If the first series of a multi-dimensional array contains strings, those strings will become the labels of the chart.

ChartData is the default property for the MSChart control.

Visual Basic: MSChart Control

ChartData Property Example

The following example uses a Visual Basic array to load the chart data grid directly. To try the example, draw a **MSChart** control on a form, paste the code into the **Form** object's code module, and run the project.

```
Option Explicit
Option Base 1

Private Sub Form_Load()
    Dim arrData(3, 1 To 3)
    arrData(1, 1) = "Jan"   ' Set the labels in the first series.
    arrData(2, 1) = "Feb"
    arrData(3, 1) = "Mar"

    arrData(1, 2) = 8
    arrData(2, 2) = 4
    arrData(3, 2) = 0.3

    arrData(1, 3) = 0.2
    arrData(2, 3) = 3
    arrData(3, 3) = 6.3
    MSChart1.ChartData = arrData
End Sub
```

In this example, the lower subscript bound was declared as 1 using the **Option Base 1** statement, rather than the default of 0. We use a **Variant** array where the first series values are set to string variables and the second and third series' values are set to numeric values. This allows both the chart's labels and data to be set simultaneously. Note that declaring the array as type **String** works too, as long as the second and third series contain numeric values. If you wish only to set the chart's data, the array can be of the numeric types **Integer**, **Long**, **Single** or **Double**. Note that doing this will replace the existing chart labels with default row/column labels. Also, a one-dimensional array will work as well as a two-dimensional one as long as the last values are either numeric or text representations of numeric values.

The following example returns data from the chart. The example contains a loop to print out the array returned from the chart. Note the use of the **Lbound** and **Ubound** functions to determine the array bounds from the chart. To try the example, draw a **CommandButton** control onto a form with a **MSChart** control. Paste the code into the code module, run the project, and click the button.

```
Option Explicit
Private Sub Command1_Click()
   Dim y() As Variant
   Dim i, j As Integer
   y = MSChart1.ChartData
   For i = LBound(y, 2) To UBound(y, 2)
      Debug.Print
      For j = LBound(y, 1) To UBound(y, 1)
         Debug.Print y(j, i)
      Next j
   Next i
End Sub
```

The returned array lower bound values are equal to 0. The returned array will always be a two-dimensional array of type **Variant**. Since **ChartData** is the default property for the chart, the object name alone, such as MSChart1, can be substituted

for MSChart1.ChartData. So you could use MSChart1 = data or data = MSChart1.

Visual Studio 6.0

Visual Basic: MSChart Control

ChartType Property

See Also Example Applies To

Returns or sets the chart type being used to display a chart.

Syntax

object.**ChartType** [= *type*]

The **ChartType** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
type	Integer. A VtChChartType constant that describes the chart type.

Visual Basic: Windows Controls

Visual Studio 6.0

CheckBox Property

See Also Example Applies To

Returns or sets a value that determines if a check box is displayed to the left of the selected date.

Syntax

object.**CheckBox** [= *boolean*]

The **CheckBox** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression specifying whether or not a check box is displayed.

Settings

The settings for *boolean* are:

Setting	Description
True	A check box is displayed to the left of the selected date.
False	(Default) A check box is not displayed to the left of the selected date.

Remarks

The check box allows the user to determine whether or not to set the date using the control. When the check box is empty, no date is selected.

Visual Basic: Windows Controls

Visual Studio 6.0

Checkboxes Property

See Also Example Applies To

Returns or sets a value that determines if checkboxes appear.

Syntax

object.**Checkboxes** [= *boolean*]

The **Checkboxes** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression specifying if the checkboxes appear, as described in Settings.

Settings

The settings for *boolean* are:

Constant	Description
False	(Default) Checkboxes do not appear.
True	Checkboxes appear.

Visual Basic Reference

Visual Studio 6.0

Checked Property

See Also Example Applies To

Returns or sets a value that determines whether a check mark is displayed next to a menu item.

Syntax

object.**Checked** [= *boolean*]

The **Checked** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression that specifies whether a check mark is displayed next to a menu item.

Settings

The settings for *boolean* are:

Setting	Description
True	Places a check mark next to a menu item.
False	(Default) Doesn't place a check mark next to a menu item.

Remarks

At design time, you can use the Menu Editor to set **Checked** to **True**. At run time, you can toggle **Checked** on and off as part of a Click event procedure attached to a **Menu** control. You can also set the value of **Checked** in a startup procedure or in a form's Load event procedure.

For a **Menu** control, **Checked** is normally read/write at run time. But **Checked** is read-only for menu items that are exposed or supplied by Visual Basic to add-ins, such as the Add-In Manager command on the Add-Ins menu.

Visual Basic Reference

Checked Property Example

This example displays and removes a check mark next to a menu item. To try this example, create a form with a **Menu** control that has one menu item (set both the **Caption** and **Name** properties to MyMenuItem), and then press F5 and choose the menu item.

```
Private Sub MyMenuItem_Click ()
   ' Turn check mark on menu item on and off.
   MyMenuItem.Checked = Not MyMenuItem.Checked
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

Checked Property (Windows Common Controls)

See Also Example Applies To

Returns or sets a value that determines if an item is checked (has a checkmark next to it).

Syntax

object.**Checked** [= *boolean*]

The **Checked** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression specifying if an item is checked, as described in Settings.

Settings

The settings for *boolean* are:

Constant	Description
False	(Default) The item is not checked.
True	The item is checked.

Visual Basic: Windows Controls

Visual Studio 6.0

Child Property (Band Object)

See Also Example Applies To

Sets or returns a reference to a child control in a **Band** object on a **CoolBar** control.

Syntax

Set *object*.**Child** = *control*

object.**Child**

The **Child** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a **Band** in the **Bands** collection of a **CoolBar** control.
control	An expression that evaluates to a control in the **Band** object of a **CoolBar** control.

Remarks

You can set the Child property of a band at design time by selecting a control from the Child list on the Bands tab in the Properties Page of the **CoolBar** control. At run time you can retrieve the name of the control on a **Band** object by reading the **Child** property.

To assign a new child control to a band at run time, use the **Set** keyword. This will replace the existing control on the **Band** object.

Visual Basic: Windows Controls

Visual Studio 6.0

Child Property (Node Object)

See Also Example Applies To

Returns a reference to the first child of a **Node** object in a **TreeView** control.

Syntax

object.**Child**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

The **Child**, **FirstSibling**, **LastSibling**, **Previous**, **Parent**, **Next**, and **Root** properties all return a reference to another **Node** object. Therefore, you can simultaneously reference and perform operations on a **Node**, as follows:

```
With TreeView1.Nodes(TreeView1.SelectedItem.Index).Child
   .Text = "New text"
   .Key = "New key"
   .SelectedImage = 3
End With
```

You can also set an object variable to the referenced **Node**, as follows:

```
Dim NodChild As Node
' Get a reference to the child of the selected node.
Set NodChild = TreeView1.Nodes(TreeView1.SelectedItem.Index).Child
' Use this reference to perform operations on the child Node.
With nodChild
   .Text = "New text"    '  Change the text.
   .Key = "New key"    ' Change key.
   .SelectedImage = 3    ' Change SelectedImage.
End With
```

Visual Basic: Windows Controls

Child Property Example

This example creates several **Node** objects. When you click on a **Node** object, the code first uses the **Children** property to determine if the **Node** has children nodes. If so, the caption of the form displays the text of the **Child** node.

```
Option Explicit

Private Sub Form_Load()
' This code creates a tree with 3 Node objects.
   TreeView1.Style = tvwTreelinesPlusMinusText ' Style 6.
   TreeView1.LineStyle = tvwRootLines  'Linestyle 1.

   ' Add several Node objects.
   Dim nodX As Node  ' Create variable.

   Set nodX = TreeView1.Nodes.Add(, , "r", "Root")
   Set nodX = TreeView1.Nodes.Add("r", tvwChild, "c1", "Child 1")

   nodX.EnsureVisible ' Show all nodes.
   Set nodX = TreeView1.Nodes.Add("c1", tvwChild, "c2", "Child 2")
   Set nodX = TreeView1.Nodes.Add("c1", tvwChild, "c3", "Child 3")
   nodX.EnsureVisible ' Show all nodes.
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
   ' If the Node does have children, then display the text of
   ' the child Node.
   If Node.Children Then
      Caption = Node.Child.Text
   End If
End Sub
```

Visual Basic Reference

Visual Studio 6.0

ChildComparison Property

See Also Example Applies To

Specifies whether the relation condition is against a field or parameter in the child DECommand object.

Note When you create a relation hierarchy, a **DERelationCondition** object is created for each relation condition that you define. Thus, each **DERelationCondition** object is equivalent to one relation condition.

Syntax

object.**ChildComparison** [=*value*]

The **ChildComparison** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
value	A constant or value that specifies the type of comparison, as described in Settings.

Settings

The settings for *value* are:

Constant	Description
deField	Field. (Default) The relation condition is against a Field object.
deParameter	Parameter. The relation condition is against a Parameter object.

Visual Basic Reference

Visual Studio 6.0

ChildCondition Property

See Also Example Applies To

Returns or sets the item in the child Command object that is used in the comparison condition.

Syntax

object.**ChildCondition** [=*string*]

The **ChildCondition** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
string	A string expression that specifies the name of a field or parameter in the child Command object.

Visual Basic: Windows Controls

Visual Studio 6.0

Children Property

See Also Example Applies To

Returns the number of child **Node** objects contained in a **Node** object.

Syntax

object.**Children**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

The **Children** property can be used to check if a **Node** object has any children before performing an operation that affects the children. For example, the following code checks for the presence of child nodes before retrieving the **Text** property of the first **Node**, using the **Child** property.

```
Private Sub TreeView1_NodeClick(ByVal Node As Node)
   If Node.Children > 0 Then
      MsgBox Node.Child.Text
   End If
End Sub
```

Visual Basic: Windows Controls

Children Property Example

This example puts several **Node** objects in a **TreeView** control. The code checks to see if a **Node** has children nodes. If so, then it displays the text of the children nodes. To try the example, place a **TreeView** control on a form and paste the code into the form's Declarations section. Run the example, click a **Node** object to select it, then click the form to see the text of the **Node** object's children.

```
Option Explicit
Private Sub Form_Click()
   Dim strC As String
   Dim N As Integer
   If TreeView1.SelectedItem.Children > 0 Then ' There are children.

      ' Get first child's text, and set N to its index value.
      strC = TreeView1.SelectedItem.Child.Text & vbLF
      N = TreeView1.SelectedItem.Child.Index

      ' While N is not the index of the child node's
      ' last sibling, get next sibling's text.
      While N <> TreeView1.SelectedItem.Child.LastSibling.Index
         strC = strC & TreeView1.Nodes(N).Next.Text & vbLF
         ' Reset N to next sibling's index.
         N = TreeView1.Nodes(N).Next.Index
      Wend
      ' Show results.
      MsgBox "Children of " & TreeView1.SelectedItem.Text & _
      " are: " & vbLF & strC
   Else ' There are no children.
      MsgBox TreeView1.SelectedItem.Text & " has no children"
   End If
End Sub

Private Sub Form_Load()
   TreeView1.BorderStyle = 1  ' Ensure border is visible
   Dim nodX As Node
   Set nodX = TreeView1.Nodes.Add(,,"d","Dates")
   Set nodX = TreeView1.Nodes.Add("d",tvwChild,"d89","1989")
   Set nodX = TreeView1.Nodes.Add("d",tvwChild,"d90","1990")

   ' Create children of 1989 node.
   Set nodX = TreeView1.Nodes.Add("d89",tvwChild, ,"John")
   Set nodX = TreeView1.Nodes.Add("d89",tvwChild, ,"Brent")
   Set nodX = TreeView1.Nodes.Add("d89",tvwChild, ,"Eric")
   Set nodX = TreeView1.Nodes.Add("d89",tvwChild, ,"Ian")
   nodX.EnsureVisible ' Show all nodes.

   ' Create children of 1990 node.
   Set nodX = TreeView1.Nodes.Add("d90",tvwChild, ,"Randy")
   Set nodX = TreeView1.Nodes.Add("d90",tvwChild, ,"Ron")
   nodX.EnsureVisible ' Show all nodes.
End Sub
```

Visual Basic: RDO Data Control

Visual Studio 6.0

ChunkRequired Property (Remote Data)

See Also Example Applies To

Returns a Boolean value that indicates if data must be accessed using the **GetChunk** method.

Syntax

object.**ChunkRequired**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Return Values

The **ChunkRequired** property return values are:

Value	Description
True	The data should be accessed using the **GetChunk** method.
False	The data need not be accessed using the **GetChunk** method.

Remarks

Use the **ChunkRequired** property to determine if the column in question should be manipulated using the **AppendChunk** and **GetChunk** methods. Accessing the **Value** property of a column whose **ChunkRequired** property is **True**, will *only* result in a trappable error when RDO is unable to fetch the data without use of the **AppendChunk** or **GetChunk** methods. In other words, when the data column does not contain more data than can be handled by conventional string handling, you are *not* required to use the **GetChunk** and **AppendChunk** methods.

By setting the **BindThreshold** property, you can adjust the number of bytes that will force the use of the **AppendChunk** and **GetChunk** methods. You can also determine the length of a *chunk* column by using the **ColumnSize** method.

Visual Basic Reference

Visual Studio 6.0

Class Property

See Also Example Applies To

Returns or sets the class name of an embedded object.

Syntax

object.**Class** [= *string*]

The **Class** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
string	A string expression specifying the class name.

Remarks

A class name defines the type of an object. Applications that support ActiveX components fully qualify the class names of their objects using either of the following syntaxes:

application.objecttype.version

objecttype.version

The syntax for ActiveX component class names has the following parts:

Part	Description
application	The name of the application that supplies the object.
objecttype	The object's name as defined in the object library.
version	The version number of the object or application that supplies the object.

For example, Microsoft Excel version 5.0 supports a number of objects, including worksheets and charts. Their class names are **Excel.Sheet.5** and **Excel.Chart.5**. Microsoft WordArt version 2.0 supports a single object with the class name **MSWordArt.2**.

Note Some ActiveX component programming documentation refers to the class name syntax as a programmatic ID.

To view a list of class names available on your system, select the **OLE** container control, select the **Class** property in the Properties window, and then click the builder button.

Copying an object from the system Clipboard updates the controls **Class** property. For example, if you paste a Microsoft Excel chart from the system Clipboard into an **OLE** container control that previously contained a Microsoft Excel worksheet, its **Class** property setting changes from **Excel.Sheet.5** to **Excel.Chart.5**. You can paste an object from the system Clipboard at run time with the **Paste** method or the **PasteSpecialDlg** method.

Visual Basic: RichTextBox Control

Visual Studio 6.0

Class Property (OLEObject Object)

See Also Example Applies To

Returns the class name of the embedded object.

Syntax

object.**Class**

The *object* argument is an object expression that evaluates to an object in the Applies To list.

Remarks

A class name defines the type of an object. Applications that support OLE or Automation fully qualify the class names of their objects using either of the following syntaxes:

```
application.objecttype.version
objecttype.version
```

The syntax for OLE class names has the following parts:

Part	Description
application	The name of the application that supplies the object.
objecttype	The object's name as defined in the object library.
version	The version number of the object or application that supplies the object.

For example, Microsoft Excel version 5.0 supports a number of objects, including worksheets and charts. Their class names are Excel.Sheet.5 and Excel.Chart.5. Microsoft WordArt version 2.0 supports a single object with the class name MSWordArt.2.

Note Some OLE programming documentation refers to the class name syntax as a programmatic ID (ProgID).

Creating an object by using the **Add** method of the **OLEObjects** collection, automatically sets the **Class** property of the **OLEObject** object.

Visual Basic Extensibility Reference

Visual Studio 6.0

ClassName Property

See Also Example Applies To

Returns a **VBControl** object.

Syntax

object.**ClassName**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Visual Basic: Windows Controls

Visual Studio 6.0

ClientHeight, ClientWidth, ClientLeft, ClientTop Properties

See Also Example Applies To

Return the coordinates of the internal area (display area) of the **TabStrip** control. Read-only at run time; not available at design time.

Syntax

object.**ClientHeight**

object.**ClientWidth**

object.**ClientLeft**

object.**ClientTop**

The *object* placeholder represents an object expression that evaluates to a **TabStrip** control.

Remarks

At run time, the client-coordinate properties **ClientLeft**, **ClientTop**, **ClientHeight**, and **ClientWidth** automatically store the coordinates of the **TabStrip** control's internal area, which is shared by all **Tab** objects in the control. So that the controls associated with a specific **Tab** appear when that **Tab** object is selected, place the **Tab** object's controls inside a container, such as a **Frame** control, whose size and position match the client-coordinate properties. To associate a container (and its controls) with a **Tab** object, create a control array, such as a **Frame** control array.

All client-coordinate properties use the scale mode of the parent form. To place a **Frame** control so it fits perfectly in the internal area, use the following code:

```
Frame1.Left = TabStrip1.ClientLeft
Frame1.Top = TabStrip1.ClientTop
Frame1.Width = TabStrip1.ClientWidth
Frame1.Height = TabStrip1.ClientHeight
```

To create the effect of placing a new tab and its associated container on top when the tab is selected:

- Set the size and location of the container in the **TabStrip** control's internal area to the client-coordinate properties; and
- Use the **ZOrder** method to place the selected tab's container control at the front or back of the z-order.

Visual Basic: Windows Controls

ClientHeight, ClientWidth, ClientLeft, ClientTop Properties Example

The following example demonstrates using the Client-coordinate properties **ClientLeft**, **ClientTop**, **ClientWidth**, and **ClientHeight** along with a **Frame** control array to display tab specific objects in the internal area of the **TabStrip** control when switching tabs. The example uses the **ZOrder** method to display the appropriate **Frame** control and the objects it contains.

To try this example, place a **TabStrip** control and a three-element **Frame** control array on the form. In one **Frame** control, place a **CheckBox** control, in another, place a **CommandButton** control, and in the third, place a **TextBox** control. Paste the following code into the Load event of the Form object, and run the program. Click the various tabs to select them and their contents.

```
Private Sub Form_Load()
Dim Tabx As Object
Dim i As Integer
   ' Sets the caption of the first tab to "Check."
   TabStrip1.Tabs(1).Caption = "Check"
   ' Adds a second tab with "Command" as its caption.
   Set Tabx = TabStrip1.Tabs.Add(2, , "Command")
   ' Adds a third tab with "Text" as its caption.
   Set Tabx = TabStrip1.Tabs.Add(3, , "Text")

   ' Aligns the frame containers with the internal
    ' area of the TabStrip control.
   For i = 0 To 2
      With TabStrip1
        Frame1(i).Move .ClientLeft, .ClientTop, _
      .ClientWidth, .ClientHeight
       End With
   Next
   ' Puts the first tab's picture box container on top
   ' at startup.
   Frame1(0).ZOrder 0
End Sub

Private Sub TabStrip1_Click()
   Frame1(TabStrip1.SelectedItem.Index - 1).ZOrder 0
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

Clip Property (MSHFlexGrid)

See Also Example Applies To

Returns or sets the contents of the cells in the **MSHFlexGrid's** selected region. This property is not available at design time.

Syntax

object.**Clip** [=*string*]

The **Clip** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
string	A string expression with the contents of the selected area.

Remarks

The *string* may hold the contents of multiple rows and columns. In *string*, a tab character, **Chr** (9), or the constant **vbTab** indicates a new cell in a row. Also in *string*, a carriage return, **Chr** (13), or the constant **vbCR** indicates the beginning of a new row. Use the **Chr** function or the vb constants to embed these characters in strings.

When placing data into an **MSHFlexGrid**, only the selected cells are affected. If there are more cells in the selected region than are described in the *string*, the remaining cells are left alone. If there are more cells described in the *string* than there are in the selected region, the unused portion of *string* is ignored.

Visual Basic: PictureClip Control

Visual Studio 6.0

Clip Property (PictureClip Control)

See Also Example Applies To

Returns a bitmap of the area in the **PictureClip** control specified by the **ClipX**, **ClipY**, **ClipWidth**, and **ClipHeight** properties. This property is read-only at run time.

Syntax

[*form.*]*PictureClip*.**Clip**

Remarks

Use this property to specify a random clipping region from the selected bitmap.

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 use pixels because the **ClipHeight** and **ClipWidth** properties that define the clipping region are measured in pixels.

Data Type

Integer

Visual Basic: PictureClip Control

Clip Example (PictureClip 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 **PictureBox**, a **PictureClip** control, and two TextBox controls. At design time, use the Properties sheet to load a valid bitmap into the **PictureClip** control.

```
Private Sub Form_Click ()
   Dim SaveMode As Integer
   ' Save the current ScaleMode for the picture box.
   SaveMode = Picture1.ScaleMode
   ' Get X and Y coordinates of the clipping region.
   PicClip1.ClipX = Val(Text1.Text)
   PicClip1.ClipY = Val(Text2.Text)
   ' Set the area of the clipping region (in pixels).
   PicClip1.ClipHeight = 100
   PicClip1.ClipWidth = 100
   ' Set the picture box ScaleMode to pixels.
   Picture1.ScaleMode = 3
   ' Set the destination area to fill the picture box.
   PicClip1.StretchX = Picture1.ScaleWidth
   PicClip1.StretchY = Picture1.ScaleHeight
   ' Assign the clipped bitmap to the picture box.
   Picture1.Picture = PicClip1.Clip
   ' Reset the ScaleMode of the picture box.
   Picture1.ScaleMode = SaveMode
End Sub
```

Visual Basic Reference

Visual Studio 6.0

ClipBehavior Property

See Also Example Applies To

Returns or sets a value that defines the clipping behavior of the HitTest event on a Windowless **UserControl** object.

Syntax

object.**ClipBehavior** [= *number*]

The **ClipBehavior** property syntax has these parts:

Part	Description
object	A UserControl object.
number	An integer that specifies clipping behavior, as described in Settings.

Settings

The settings for *number* are:

Constant	Setting	Description
None	0	The output of graphics methods will appear anywhere within the rectangular bounds of the control.
UseRegion	1	(Default) The output of graphics methods will be clipped to the MaskRegion of your control.

Remarks

You can use the **ClipBehavior** property to determine where the output of graphics methods will appear on your UserControl. By default, any drawing done using graphics methods will only be visible over the MaskRegion of the control. The MaskRegion consists of any subcontrols plus any mask defined by the **MaskPicture** and **MaskColor** properties. Any drawing done outside of the MaskRegion will be invisible.

By setting **ClipBehavior** to None, drawing will be visible over both the MaskRegion and the TransparentRegion of the control.

When used in combination with the **HitBehavior** property, this property helps to determine the HitResult argument of the HitTest event.

Note This property is ignored if the **Windowless** property of the **UserControl** object is set to False or if the **BackStyle** property is set to Opaque.

Important Not all control containers support the Windowless property. The **HitBehavior** property should only be changed if you know it will be used in containers that support Windowless activation.

Visual Basic Reference

Visual Studio 6.0

Clipboard Property

See Also Example Applies To

Returns a **Clipboard** object, which provides access to the system Clipboard.

Syntax

Clipboard

Remarks

The **Clipboard** object is used to manipulate text and graphics on the Clipboard. You can use this object to enable a user to copy, cut, and paste text or graphics in your application. Before copying any material to the **Clipboard** object, you should clear its contents by performing a **Clear** method, such as `Clipboard.Clear`.

Note that the **Clipboard** object is shared by all Windows applications, and thus, the contents are subject to change whenever you switch to another application.

The **Clipboard** object can contain several pieces of data as long as each piece is in a different format. For example, you can use the **SetData** method to put a bitmap on the Clipboard with the **vbCFDIB** format, and then use the **SetText** method with the **vbCFText** format to put text on the Clipboard. You can then use the **GetText** method to retrieve the text or the **GetData** method to retrieve the graphic. Data on the Clipboard is lost when another set of data of the same format is placed on the Clipboard either through code or a menu command.

Visual Basic Reference

Visual Studio 6.0

ClipControls Property

See Also Example Applies To

Returns or sets a value that determines whether graphics methods in Paint events repaint the entire object or only newly exposed areas. Also determines whether the Microsoft Windows operating environment creates a clipping region that excludes nongraphical controls contained by the object. Read-only at run time.

Syntax

object.**ClipControls**

The **ClipControls** property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
Boolean	A Boolean expression that specifies how objects are repainted, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	(Default) Graphics methods in Paint events repaint the entire object. A clipping region is created around nongraphical controls on the form before a Paint event.
False	Graphics methods in Paint events repaint only newly exposed areas. A clipping region isnt created around nongraphical controls before a Paint event. Complex forms usually load and repaint faster when **ClipControls** is set to **False**.

Remarks

Clipping is the process of determining which parts of a form or container, such as a **Frame** or **PictureBox** control, are painted when the form is displayed. An outline of the form and controls is created in memory. The Windows operating environment uses this outline to paint some parts, such as the background, without affecting other parts, such as the contents of a **TextBox** control. Because the clipping region is created in memory, setting this property to **False** can reduce the time needed to paint or repaint a form.

The clipping region includes most controls, but doesn't clip around the **Image**, **Label**, **Line**, or **Shape** controls.

Avoid nesting intrinsic controls with **ClipControls** set to **True** inside a control with **ClipControls** set to **False** (for instance, a command button inside a picture box). This kind of control nesting causes the controls to repaint incorrectly. To fix this problem, set the **ClipControls** property for both the container control and the nested controls to **True**.

Visual Basic Reference

ClipControls Property Example

This example shows how the **ClipControls** property affects the repainting of a form. To try this example, paste the code into the Declarations section of a form, and then press F5. Notice that the color of the entire form changes each time you resize it or cover part of it with another form or application. End the program and set **ClipControls** to **False**, and then run the program again. Notice that only newly exposed parts of the form are repainted.

```
Private Sub Form_Paint ()
   ' Select a random color for the background.
   BackColor = &HFFFFFF * Rnd
End Sub
```

Visual Basic: PictureClip Control

Visual Studio 6.0

ClipHeight Property (PictureClip Control)

See Also Example Applies To

Specifies the height of the bitmap section to be copied by the **Clip** property. This property is not available at design time.

Syntax

[*form.*]*PictureClip*.**ClipHeight**[= *Height%*]

Remarks

This property is measured in pixels.

Data Type

Integer

Visual Basic: PictureClip Control

Clip Example (PictureClip 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 **PictureBox**, a **PictureClip** control, and two TextBox controls. At design time, use the Properties sheet to load a valid bitmap into the **PictureClip** control.

```
Private Sub Form_Click ()
   Dim SaveMode As Integer
   ' Save the current ScaleMode for the picture box.
   SaveMode = Picture1.ScaleMode
   ' Get X and Y coordinates of the clipping region.
   PicClip1.ClipX = Val(Text1.Text)
   PicClip1.ClipY = Val(Text2.Text)
   ' Set the area of the clipping region (in pixels).
   PicClip1.ClipHeight = 100
   PicClip1.ClipWidth = 100
   ' Set the picture box ScaleMode to pixels.
   Picture1.ScaleMode = 3
   ' Set the destination area to fill the picture box.
   PicClip1.StretchX = Picture1.ScaleWidth
   PicClip1.StretchY = Picture1.ScaleHeight
   ' Assign the clipped bitmap to the picture box.
   Picture1.Picture = PicClip1.Clip
   ' Reset the ScaleMode of the picture box.
   Picture1.ScaleMode = SaveMode
End Sub
```

Visual Basic: MaskedEdit Control

Visual Studio 6.0

ClipMode Property

See Also Example Applies To

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

Syntax

[*form.*]**MaskedEdit.ClipMode** [= *setting%*]

Remarks

The following table lists the **ClipMode** property settings for the **Masked Edit** control.

Setting	Description
mskIncludeLiterals (0)	(Default) Include literals on a cut or copy command.
mskExcludeLiterals (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)

Visual Basic: MaskedEdit Control

Visual Studio 6.0

ClipText Property

See Also Example Applies To

Returns the text in the **Masked Edit** control, excluding literal characters of the input mask. This property is not available at design time and is read-only at run time.

Syntax

[*form.*]**MaskedEdit.ClipText**

Remarks

This property acts the same as the **SelText** property when the **Mask** property is set to the empty string ("").

Data Type

String

Visual Basic: PictureClip Control

Visual Studio 6.0

ClipWidth Property (PictureClip Control)

See Also Example Applies To

Specifies the width of the bitmap section to be copied by the **Clip** property. This property is not available at design time.

Syntax

[*form.*]*PictureClip*.**ClipWidth**[= *Width%*]

Remarks

This property is measured in pixels.

Data Type

Integer

Visual Basic: PictureClip Control

Clip Example (PictureClip 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 **PictureBox**, a **PictureClip** control, and two TextBox controls. At design time, use the Properties sheet to load a valid bitmap into the **PictureClip** control.

```
Private Sub Form_Click ()
   Dim SaveMode As Integer
   ' Save the current ScaleMode for the picture box.
   SaveMode = Picture1.ScaleMode
   ' Get X and Y coordinates of the clipping region.
   PicClip1.ClipX = Val(Text1.Text)
   PicClip1.ClipY = Val(Text2.Text)
   ' Set the area of the clipping region (in pixels).
   PicClip1.ClipHeight = 100
   PicClip1.ClipWidth = 100
   ' Set the picture box ScaleMode to pixels.
   Picture1.ScaleMode = 3
   ' Set the destination area to fill the picture box.
   PicClip1.StretchX = Picture1.ScaleWidth
   PicClip1.StretchY = Picture1.ScaleHeight
   ' Assign the clipped bitmap to the picture box.
   Picture1.Picture = PicClip1.Clip
   ' Reset the ScaleMode of the picture box.
   Picture1.ScaleMode = SaveMode
End Sub
```

Visual Basic: PictureClip Control

Visual Studio 6.0

ClipX Property (PictureClip Control)

See Also Example Applies To

Specifies the x-coordinate of the upper-left corner of the bitmap section to be copied by the **Clip** property. This property is not available at design time.

Syntax

[*form.*]*PictureClip*.**ClipX**[= *X%*]

Remarks

This property is measured in pixels.

Data Type

Integer

Visual Basic: PictureClip Control

Clip Example (PictureClip 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 **PictureBox**, a **PictureClip** control, and two TextBox controls. At design time, use the Properties sheet to load a valid bitmap into the **PictureClip** control.

```
Private Sub Form_Click ()
   Dim SaveMode As Integer
   ' Save the current ScaleMode for the picture box.
   SaveMode = Picture1.ScaleMode
   ' Get X and Y coordinates of the clipping region.
   PicClip1.ClipX = Val(Text1.Text)
   PicClip1.ClipY = Val(Text2.Text)
   ' Set the area of the clipping region (in pixels).
   PicClip1.ClipHeight = 100
   PicClip1.ClipWidth = 100
   ' Set the picture box ScaleMode to pixels.
   Picture1.ScaleMode = 3
   ' Set the destination area to fill the picture box.
   PicClip1.StretchX = Picture1.ScaleWidth
   PicClip1.StretchY = Picture1.ScaleHeight
   ' Assign the clipped bitmap to the picture box.
   Picture1.Picture = PicClip1.Clip
   ' Reset the ScaleMode of the picture box.
   Picture1.ScaleMode = SaveMode
End Sub
```

Visual Basic: PictureClip Control

Visual Studio 6.0

ClipY Property (PictureClip Control)

See Also Example Applies To

Specifies the y-coordinate of the upper-left corner of the bitmap section to be copied by the **Clip** property. This property is not available at design time.

Syntax

[*form.*]*PictureClip*.**ClipY**[= *Y%*]

Remarks

This property is measured in pixels.

Data Type

Integer

Visual Basic: PictureClip Control

Clip Example (PictureClip 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 **PictureBox**, a **PictureClip** control, and two TextBox controls. At design time, use the Properties sheet to load a valid bitmap into the **PictureClip** control.

```
Private Sub Form_Click ()
   Dim SaveMode As Integer
   ' Save the current ScaleMode for the picture box.
   SaveMode = Picture1.ScaleMode
   ' Get X and Y coordinates of the clipping region.
   PicClip1.ClipX = Val(Text1.Text)
   PicClip1.ClipY = Val(Text2.Text)
   ' Set the area of the clipping region (in pixels).
   PicClip1.ClipHeight = 100
   PicClip1.ClipWidth = 100
   ' Set the picture box ScaleMode to pixels.
   Picture1.ScaleMode = 3
   ' Set the destination area to fill the picture box.
   PicClip1.StretchX = Picture1.ScaleWidth
   PicClip1.StretchY = Picture1.ScaleHeight
   ' Assign the clipped bitmap to the picture box.
   Picture1.Picture = PicClip1.Clip
   ' Reset the ScaleMode of the picture box.
   Picture1.ScaleMode = SaveMode
End Sub
```

Visual Basic: MSChart Control

Clockwise Property

See Also Example Applies To

Returns or sets a value that specifies whether pie charts are drawn in a clockwise direction.

Syntax

object.**Clockwise** [= *boolean*]

The **Clockwise** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression that controls the direction used to draw pie charts, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	(Default) Pie charts are drawn in a clockwise direction.
False	The charts are drawn in a counterclockwise direction.

Visual Basic Extensibility Reference

Visual Studio 6.0

CodeLocation Property

See Also Example Applies To

Returns the line location in the code module where the member is defined.

Syntax

object.**CodeLocation** [= *propkind*]

The **CodeLocation** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
propkind	An enumerated value of **vbext_PropertyKind**, as described in Settings.

Settings

The settings for **vbext_PropertyKind** are:

Constant	Value	Description
vbext_PropertyGet	1	(Default) Returns the code location for the Get element of the property.
vbext_PropertyLet	2	Returns the code location for the Let element of the property.
vbext_PropertySet	3	Returns the code location for the Set element of the property.

Visual Basic Extensibility Reference

Visual Studio 6.0

CodeModule Property

See Also Example Applies To Specifics

Returns an object representing the code behind the component. Read-only.

Remarks

The **CodeModule** property returns **Nothing** if the component doesn't have a code module associated with it.

Note The **CodePane** object represents a visible code window. A given component can have several **CodePane** objects. The **CodeModule** object represents the code within a component. A component can only have one **CodeModule** object.

Visual Basic Extensibility Reference

CodeModule Property Example

The following example uses the **CodeModule** and **CountOfLines** properties to return the number of lines in a particular code module.

```
Debug.Print Application.VBE.ActiveVBProject.VBComponents(6).CodeModule.CountOfLines
```

Visual Basic Extensibility Reference

Visual Studio 6.0

CodePane Property

See Also Example Applies To Specifics

Returns a **CodePane** object. Read-only.

Remarks

If a code pane exists, it becomes the active code pane, and the window that contains it becomes the active window. If a code pane doesn't exist for the module, the **CodePane** property creates one.

Visual Basic Extensibility Reference

Visual Studio 6.0

CodePanes Property

See Also Example Applies To Specifics

Returns the collection of active **CodePane** objects. Read-only.

Visual Basic Extensibility Reference

CodePanes Property Example

The following example uses the **CodePanes** and **TopLine** properties to display the line number of the top line in the specified code pane.

```
Debug.Print Application.VBE.CodePanes(3).TopLine
```

Visual Basic Extensibility Reference

Visual Studio 6.0

CodePaneView Property

See Also Example Applies To Specifics

Returns a value indicating whether the code pane is in Procedure view or Full Module view. Read-only.

Return Values

The **CodePaneView** property return values are:

Constant	Description
vbext_cv_ProcedureView	The specified code pane is in Procedure view.
Vbext_cv_FullModuleView	The specified project is in Full Module view.

Visual Basic Extensibility Reference

CodePaneView Property Example

The following example uses the **CodePaneView** property to return a value indicating whether the specified code pane is in procedure view or full module view.

```
Debug.Print Application.VBE.CodePanes(3).CodePaneView
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

Col, Row Properties (MSHFlexGrid)

See Also Example Applies To

Returns or sets the coordinates of the active cell in an **MSHFlexGrid**. These properties are not available at design time.

Syntax

object.**Col** [=*number*]
object.**Row** [=*number*]

Syntax for the **Col** and **Row** properties has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
number	A Long value that specifies the position of the active cell.

Remarks

Use these properties to specify a cell in an **MSHFlexGrid** or to determine which row or column contains the current cell. Columns and rows are numbered from zero rows are numbered from top to bottom, and columns are numbered from left to right.

Setting these properties automatically resets **RowSel** and **ColSel**, the selection becoming the current cell. Therefore, to specify a block selection, you must set **Row** and **Col** first, and then set **RowSel** and **ColSel**.

The value of the current cell, defined by the **Col** and **Row** settings, is the text contained in that cell. To modify a cells value without changing the selected **Row** and **Col** properties, use the **TextMatrix** property.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

ColAlignment, ColAlignmentBand, ColAlignmentHeader Properties (MSHFlexGrid)

See Also Example Applies To

Returns or sets the alignment of data in a column. This can be a standard column, a column within a band, or a column within a header. This property is not available at design time (except indirectly through the **FormatString** property).

Syntax

object.**ColAlignment(***Index***)** [=*value*]
object.**ColAlignmentBand(***BandNumber*, *BandColIndex***)** [=*value*]
object.**ColAlignmentHeader(***BandNumber*, *BandColIndex***)** [=*value*]

Syntax for the **ColAlignment**, **ColAlignmentBand**, and **ColAlignmentHeader** properties has:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Index *BandNumber*	Required. A Long value that specifies the number of the band in the **MSHFlexGrid**.
BandColIndex	Required. A Long value that specifies the number of the column in the **MSHFlexGrid**.
value	An integer or constant that specifies the alignment of data in a column, as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
flexAlignLeftTop	0	The column content is aligned left, top.
flexAlignLeftCenter	1	Default for strings. The column content is aligned left, center.

flexAlignLeftBottom	2	The column content is aligned left, bottom.
flexAlignCenterTop	3	The column content is aligned center, top.
flexAlignCenterCenter	4	The column content is aligned center, center.
flexAlignCenterBottom	5	The column content is aligned center, bottom.
flexAlignRightTop	6	The column content is aligned right, top.
flexAlignRightCenter	7	Default for numbers. The column content is aligned right, center.
flexAlignRightBottom	8	The column content is aligned right, bottom.
flexAlignGeneral	9	The column content is of general alignment. This is "left, center" for strings and "right, center" for numbers.

Remarks

Any column can have an alignment that is different from other columns. The **ColAlignment** property affects all cells in the specified column, including those in fixed rows.

To set individual cell alignments, use the **CellAlignment** property. To set column alignments at design time, use the **FormatString** property.

If the **MSHFlexGrid** is in vertical mode, the setting **ColAlignment(**3**)** may affect columns in multiple bands.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

ColAlignment Property (MSHFlexGrid) Example

The following code sets the alignment of the third column to "right, center" using the constant value.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub Form1_Load ()
   MSHFlexGrid1.ColAlignment(3) =flexAlignRightCenter
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

ColAlignmentFixed Property

See Also Example Applies To

Returns or sets the alignment of data in the fixed cells of a column in an **MSHFlexGrid**.

Syntax

object.**ColAlignmentFixed(***index***)** [=*value*]

The **ColAlignmentFixed** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	A Long value that specifies the column.
value	An integer that determines the alignment of the data in the fixed cells, as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
flexAlignLeftTop	0	The cell content is aligned left, top.
flexAlignLeftCenter	1	The cell content is aligned left, center.
flexAlignLeftBottom	2	The cell content is aligned left, bottom.
flexAlignCenterTop	3	The cell content is aligned center, top.
flexAlignCenterCenter	4	The cell content is aligned center, center.
flexAlignCenterBottom	5	The cell content is aligned center, bottom.
flexAlignRightTop	6	The cell content is aligned right, top.

flexAlignRightCenter	7	The cell content is aligned right, center.
flexAlignRightBottom	8	The cell content is aligned right, bottom.
flexAlignGeneral	9	The column content is of general alignment. This is left center for strings and right center for numbers.

Remarks

You can use the **FixedCols** and **FixedRows** properties to define the cells in a column.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

ColData, RowData, BandData Properties (MSHFlexGrid)

SeeAlso Example Applies To

Returns or sets an arbitrary long value associated with each row, column, or band. These properties are not available at design time.

Syntax

object.**ColData(***number***)** [=*value*]
object.**RowData(***number***)** [=*value*]
object.**BandData(***number***)** [=*value*]

Syntax for the **ColData**, **RowData**, and **BandData** properties has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
number	A Long value that specifies the column, row, or band in the **MSHFlexGrid**. This number indicates where to save or retrieve the data.
value	A Long value that specifies the contents of the **ColData**, **RowData**, or **BandData** arrays.

Remarks

Use the **RowData**, **ColData**, and **BandData** properties to associate a specific number with each row, column, or band on an **MSHFlexGrid**. Once a number is associated with each row, column, or band, you can use the numbers programmatically to identify the items.

You can add rows containing totals to an **MSHFlexGrid** and identify those rows by setting their **RowData** property to a non-zero value. To update the totals, you can delete the old totals by scanning the **RowData** array and removing the appropriate rows.

Another typical use of the **RowData** property is to keep an index in an array of data structures associated with the items described in each row.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

ColHeader Property (MSHFlexGrid)

SeeAlso Example Applies To

Specifies whether headers should display for the specified band.

Syntax

object.**ColHeader(***BandNumber***)** [=*value*]

The **ColHeader** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
BandNumber	Required. A Long value that specifies the band to be affected.
value	Optional. A Long value that specifies whether the band that contains the headers will show or be hidden, as described in Settings.

Settings

The settings for *value* are:

Settings	Value	Description
flexColHeaderOff	0	No headers display for the band.
flexColHeaderOn	1	The headers display for each band.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

ColHeaderCaption Property (MSHFlexGrid)

SeeAlso Example Applies To

Specifies the caption to display in the header of the specified column and band.

Syntax

object.**ColHeaderCaption(***number, index***)** [=*string*]

The **ColHeaderCaption** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
number	A Long value that indicates the band that contains the specified column to set the caption.
index	A Long value that indicates the specified column.
string	A string expression that specifies the caption.

Visual Basic: DataGrid Control

Visual Studio 6.0

ColIndex Property

See Also Example Applies To

Returns a value indicating the position of the column in the **Columns** collection of the **DataGrid** control and the visible position (left-to-right) of the column in the **DataGrid** control. This property is read-only at run time and not available at design time.

Syntax

object.**ColIndex**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

Note that the value of the position given is the same as if all columns were visible and doesn't change if you hide one or more columns.

This property returns the zero-based index of a column within the **Columns** collection. Note, however, that **For Each** is not guaranteed to return items in any particular order.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

ColIsVisible Property

SeeAlso Example Applies To

Returns a value indicating whether a specified column is currently visible.

Syntax

object.**ColIsVisible(***index***)** = *Boolean*

The **ColIsVisible** property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
Index	A Long value that specifies the column.
Boolean	A Boolean expression that specifies whether the specified column is visible.

Settings

The settings for *Boolean* are:

Setting	Description
True	Default. The specified column is visible.
False	The specified column is not visible.

Remarks

A column is not visible when it's outside the client area of the control's window. In other words, when it's scrolled out of view.

Visual Basic Extensibility Reference

Visual Studio 6.0

Collection Property

See Also Example Applies To Specifics

Returns the collection that contains the object you are working with. Read-only.

Remarks

Most objects in this object model have either a **Parent** property or a **Collection** property that points to the object's parent object.

Use the **Collection** property to access the properties, methods, and controls of the collection to which the object belongs.

Visual Basic Extensibility Reference

Collection Property Example

The following example uses the **Collection** and **Count** properties to return the number of objects the active project contains, when viewed as a collection of objects.

```
Debug.Print Application.VBE.ActiveVBProject.Collection.Count
```

Visual Basic: CommonDialog Control

Visual Studio 6.0

Color Property

See Also Example Applies To

Returns or sets the selected color.

Syntax

object.**Color** [= *number*]

The **Color** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
number	A numeric expression that specifies the color.

Settings

The settings for *number* are:

Setting	Description
Normal RGB colors	Colors set with the **RGB** or **QBColor** functions in code.
System default colors	Colors specified with the system color constants in the Visual Basic (VB) object library in the Object Browser. The Microsoft Windows operating environment substitutes the user's choices, as specified by the user's Control Panel settings.

Remarks

For this property to return a color in a Color dialog box, the **cdlCCRGBInit** flag must be set. In the Font dialog box, the **cdlCFEffects** flag must be set.

Data Type

Long

Visual Basic Reference

Visual Studio 6.0

ColorMode Property

See Also Example Applies To

Returns or sets a value that determines whether a color printer prints output in color or monochrome. Not available at design time.

Syntax

object.**ColorMode** [= *value*]

The **ColorMode** property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
Value	A constant or integer that specifies the print mode, as described in Settings.

Settings

The settings for *value* are:

Constant	Value	Description
VbPRCMMonochrome	1	Print output in monochrome (usually shades of black and white).
VbPRCMColor	2	Print output in color.

Remarks

The default value depends on the printer driver and the current printer settings. Monochrome printers ignore this property.

Note The effect of the properties of the **Printer** object depends on the driver supplied by the printer manufacturer. Some property settings may have no effect, or several different property settings may all have the same effect. If you set the **ColorMode** property for a printer which doesnt support color, the setting is ignored. If you attempt to reference the **ColorMode** property, however, you will get an error message. Settings outside the accepted range may also produce an error. For more information, see the manufacturer's documentation for the specific driver.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

ColPos Property

SeeAlso Example Applies To

Returns the distance, in twips, between the upper-left corner of the control and the upper-left corner of a specified column.

Syntax

object.**ColPos(***index***)**

The **ColPos** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	A Long value that specifies the column.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

ColPosition, RowPosition Properties

See Also Example Applies To

- **ColPosition** Sets the position of an **MSHFlexGrid** column, allowing you to move columns to specific positions.
- **RowPosition** Sets the position of an **MSHFlexGrid** row, allowing you to move rows to specific positions.

Syntax

object.**ColPosition(***index*, *number***)** [= *value*]
object.**RowPosition(***number***)** [= *value*]

Note The above **ColPosition** syntax cannot be used in an ***MSFlexGrid*** due to the limitations of this control. If you are using **MSFlexGrid**, use the syntax: *object*.**ColPosition(***number***)** [= *value*]

Syntax for the **ColPosition** and **RowPosition** properties has these parts:

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
index	A Long value that specifies the column to be moved. **Note** This is not applicable to the **MSFlexGrid**.
number	A Long value that specifies the band containing the column to be moved. Optional. **Note** In the **MSFlexGrid**, this is the number of the column or row to be moved.
value	Integer. A numeric expression that specifies the new position of the column or row.

Remarks

In the **MSHFlexGrid**, when **BandNumber** is not specified, it defaults to 0. Hence, when the grid is not bound to a hierarchical Recordset, using 0 for **BandNumber** and not specifying **BandNumber** both have the same result. Note that **BandNumber** is an optional parameter for backward compatibility with the **MSFlexGrid**.

The index and setting must correspond to valid row or column numbers (in the range 0 to **Rows** 1 or **Cols** 1), otherwise an error is generated.

When a row or column is moved using the **RowPosition** and **ColPosition** properties, all formatting information moves with it. This includes width, height, alignment, colors, and font properties. To move text only, use the **Clip** property.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

ColPosition Property Example

When implemented, the following code causes a column to move to the first position (the left-most column) when the user clicks on the column.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub MSHFlexGrid1_Click ()
   MSHFlexGrid1.ColPosition(MSFlexGrid1.MouseCol) = 0
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

Cols, Rows Properties (MSHFlexGrid)

See Also Example Applies To

- **Cols** Returns or sets the total number of columns in an **MSHFlexGrid**.
- **Rows** Returns or sets the total number of rows in an **MSHFlexGrid**.
  Note The **MSFlexGrid** contains no bands. If using the **Rows** property programmatically in the **MSFlexGrid**, you can only return or set the total number of columns or rows. Also, the **Rows** syntax, below, cannot be used in an **MSFlexGrid** due to the limitations of this control. If you are using **MSFlexGrid**, use the syntax: *object*.**Rows** [= *value*].

Syntax

object.**Cols(***BandNumber***)** [= *value*]
object.**Rows** [= *value*]

Syntax for the **Cols** and **Rows** properties has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
BandNumber	A Long value that either specifies the band to get or sets the total number of columns. Optional. **Note** This is not applicable to the **MSFlexGrid**.
value	A Long value that specifies the number of columns or rows.

Remarks

You can use these properties to expand and shrink **MSHFlexGrid** dynamically at run time.

The minimum number of rows and columns is 0. The maximum number is limited by the memory available on your computer.

The value of **Cols** must be at least one greater than the value of **FixedCols**, unless they are both set to zero. The value of **Rows** must be at least one greater than the value of **FixedRows**, unless they are both set to zero.

When *number* is not specified, it defaults to 0. Hence, when the **MSHFlexGrid** is not bound to a hierarchical Recordset, using 0 or not specifying *number* both have the same result. Note that *number* is an optional parameter for backward compatibility with the **MSFlexGrid**.

Visual Basic: PictureClip Control

Visual Studio 6.0

Cols, Rows Properties (PictureClip Control)

See Also Example Applies To

Set or return the total number of columns or rows in the picture.

Syntax

[*form.*]*PictureClip*.**Cols**[= *cols%*]

[*form.*]*PictureClip*.**Rows**[= *rows%*]

Remarks

Use these properties to divide the source bitmap into a uniform matrix of picture cells. Use the **GraphicCell** property to specify individual cells.

A **PictureClip** 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

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

ColSel Property Example

The following code returns the *value* of the **ColSel** property into the first cell of MSHFlexGrid1. This value changes as the user clicks on various selections of cell groups.

Note If you are using the **MSFlexGrid**, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Private Sub MSHFlexGrid1_MouseUp _
(Button As Integer, Shift As Integer, x As Single, _
y As Single)
   MSHFlexGrid1.Text = MSHFlexGrid1.ColSel
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

ColSel, RowSel Properties

See Also Example Applies To

- **ColSel** Returns or sets the start or end column for a range of cells.
- **RowSel** Returns or sets the start or end row for a range of cells.

These properties are not available at design time.

Syntax

object.**ColSel** [= *value*]
object.**RowSel** [= *value*]

Syntax for the **ColSel** and **RowSel** properties has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A Long value that specifies the start or end row, or column, for a range of cells.

Remarks

You can use these properties to select a specific region of the **MSHFlexGrid** programmatically, or to read the dimensions of an area that the user selects into code.

The **MSHFlexGrid** cursor is in the cell at **Row**, **Col**. The **MSHFlexGrid** selection is the region between rows **Row** and **RowSel** and columns **Col** and **ColSel**. Note that **RowSel** may be above or below **Row**, and **ColSel** may be to the left or to the right of **Col**.

Whenever you set the **Row** and **Col** properties, **RowSel** and **ColSel** are automatically reset, so the cursor becomes the current selection. To select a block of cells from code, you must first set the **Row** and **Col** properties, and then set **RowSel** and **ColSel**.

Visual Basic for Applications Reference

Visual Studio 6.0

Column Property

See Also Example Applies To Specifics

Description

Read-only property that returns the column number of the current character position in a **TextStream** file.

Syntax

object.**Column**

The *object* is always the name of a **TextStream** object.

Remarks

After a newline character has been written, but before any other character is written, **Column** is equal to 1.

Visual Studio 6.0

Visual Basic: MSChart Control

Column Property (MSChart Control)

See Also Example Applies To

Returns or sets the current data column in the data grid.

Syntax

object.**Column** [= *col*]

The **Column** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
col	Integer. The current data column.

Remarks

You must select a column before you can use other properties to change the column's corresponding chart series or any data point within the series.

Visual Studio 6.0

Visual Basic: MSChart Control

ColumnCount Property

See Also Example Applies To

Returns or sets the number of columns in the current data grid associated with a chart.

Syntax

object.**ColumnCount** [= *count*]

The **ColumnCount** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
count	The number of data columns.

Visual Basic: Windows Controls

Visual Studio 6.0

ColumnHeaderIcons Property

See Also Example Applies To

Returns or sets the **Imagelist** control that supplies images for the **ColumnHeaders** collection.

Syntax

object.**ColumnHeaderIcons** [= *imagelist*]

The **ColumnHeaderIcons** syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
imagelist	An **ImageList** control or an object reference to an **Imagelist** control.

Remarks

To set an icon for a **ColumnHeader** object, set its **Icon** property to an index, key, or object reference to a **ListImage** object in the **ImageList** control specified by the **ColumnHeaderIcons** property.

Visual Basic: Windows Controls

ColumnHeaderIcons, Icon Property Example

The example sets the **ColumnHeaderIcons** property an **ImageList** control, then sets the **Icon** property to the **Key** of a **ListImage** object.

```
Option Explicit
Private Sub Form_Load()
   ' Assumes an ImageList control populated with at least one image.
   Dim c As ColumnHeader
   Dim i As Integer

   For i = 1 To 4 ' Create four ColumnHeader objects.
      ListView1.ColumnHeaders.Add , , "Col " & i
   Next I

   ListView1.View = lvwReport

   ImageList1.ListImages(1).Key = "Key1" ' Set Key property of ListImage.
   ListView1.ColumnHeaderIcons = ImageList1
   For Each c In ListView1.ColumnHeaders
      c.Icon = "Key1"
   Next
End Sub
```

Visual Basic: DataGrid Control

Visual Studio 6.0

ColumnHeaders Property

See Also Example Applies To

Returns or sets a value indicating whether the column headers are displayed in a **DataGrid** control.

Syntax

object.**ColumnHeaders** [= *value*]

The **ColumnHeaders** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A Boolean expression that determines whether column headers are displayed, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
True	**DataGrid** control's column headers are displayed
False	**DataGrid** control's column headers aren't displayed

Visual Basic: Windows Controls

Visual Studio 6.0

ColumnHeaders Property (ListView Control)

See Also Example Applies To

Returns a reference to a collection of **ColumnHeader** objects.

Syntax

object.**ColumnHeaders**

The *object* placeholder represents an object expression that evaluates to a **ListView** control.

Remarks

You can manipulate **ColumnHeader** objects using standard collection methods (for example, the **Remove** method). Each **ColumnHeader** in the collection can be accessed either by its index or by a unique key, stored in the **Key** property.

Visual Studio 6.0

Visual Basic: MSChart Control

ColumnLabel Property (DataGrid Object)

See Also Example Applies To

Returns or sets the label on a data column in the grid associated with a chart.

Syntax

object.**ColumnLabel**(*column, labelIndex*) [= *text*]

The **ColumnLabel** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
column	Integer. Identifies a specific data column. Columns are numbered from left to right beginning with 1. Any columns containing labels are not counted as data columns.
labelIndex	Integer. Identifies a specific label. If more than one level of column labels exist for the column, you must identify one of them. Column labels are numbered from bottom to top beginning at 1.
text	String. The column label text.

Visual Studio 6.0

Visual Basic: MSChart Control

ColumnLabel Property (MSChart)

See Also Example Applies To

Returns or sets the label text associated with a column in the data grid of a chart.

Syntax

object.**ColumnLabel** [= *text*]

The **ColumnLabel** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
text	String. Label text associated with a column in the data grid.

Remarks

This property sets the label for the column currently identified by the **Column** property.

Visual Basic: MSChart Control

ColumnLabel Property Example

The example adds three column labels to the z axis of a chart. To try the example, draw an **MSChart** control on a form. Paste the code into the **Form** object's code module, and run the project. Click the form to see the labels.

```
Option Explicit
Option Base 1

Private Sub Form_Click()
   With MSChart1
      .chartType = VtChChartType3dLine

      .Plot.Axis(VtChAxisIdZ).Labels(1).VtFont.Size = 24
      .DataGrid.ColumnLabel(1, 1) = "Label Z1"
      .DataGrid.ColumnLabel(2, 1) = "Label Z2"
      .DataGrid.ColumnLabel(3, 1) = "Label Z3"
   End With
End Sub

Private Sub Form_Load()
   ' The first series contains labels for the X axis.
   Dim arrData(4, 1 To 4)
   arrData(1, 1) = "Jan"
   arrData(2, 1) = "Feb"
   arrData(3, 1) = "Mar"
   arrData(4, 1) = "Apr"

   arrData(1, 2) = 8
   arrData(2, 2) = 4
   arrData(3, 2) = 0.3
   arrData(4, 2) = 3

   arrData(1, 3) = 0.2
   arrData(2, 3) = 3
   arrData(3, 3) = 6.3
   arrData(4, 3) = 2

   arrData(1, 4) = 5
   arrData(2, 4) = 7
   arrData(3, 4) = 2
   arrData(4, 4) = 9
   MSChart1.ChartData = arrData
End Sub
```

Visual Studio 6.0

Visual Basic: MSChart Control

ColumnLabelCount Property

See Also Example Applies To

Returns or sets the number of levels of labels on the columns in the data grid associated with a chart.

Syntax

object.**ColumnLabelCount** [= *count*]

The **ColumnLabelCount** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
count	Integer. The number of column label levels. Set this property to add or delete levels of labels on data grid columns.

Remarks

Column label levels are numbered from bottom to top, beginning at 1. Levels are added or subtracted from the top.

Visual Studio 6.0

Visual Basic: MSChart Control

ColumnLabelIndex Property

See Also Example Applies To

Returns or sets a specific level of column labels associated with a chart.

Syntax

object.**ColumnLabelIndex** [= *index*]

The **ColumnLabelIndex** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	Integer. Identifies a column label level.

Remarks

To set a label on a column with more than one level of labels, or to return the current value for a label, you must first identify which level you want to affect. Column label levels are numbered from bottom to top, beginning at 1.

Visual Basic: DataGrid Control

Visual Studio 6.0

Columns Property (DataGrid)

See Also Example Applies To

Returns a collection of **Column** objects.

Syntax

object.**Columns**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

The **Columns** property returns a collection of **Column** objects in a **Variant**.

You can manipulate most of a **DataGrid** control's attributes by changing the properties of **Column** objects. Choose a specific **Column** object with the **Col** property.

Visual Basic Reference

Visual Studio 6.0

Columns Property (ListBox)

See Also Example Applies To

Returns or sets a value that determines whether a **ListBox** control scrolls vertically or horizontally and how the items in the columns are displayed. If it scrolls horizontally, the **Columns** property determines how many columns are displayed.

Syntax

object.**Columns** [= *number*]

The **Columns** property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
Number	An integer that specifies how a control scrolls and how items are arranged in columns, as described in Settings.

Settings

The settings for *number* are:

Setting	Description
0	(Default) Items are arranged in a single column and the **ListBox** scrolls vertically.
1 to *n*	Items are arranged in snaking columns, filling the first column, then the second column, and so on. The **ListBox** scrolls horizontally and displays the specified number of columns.

Remarks

For horizontal-scrolling **ListBox** controls, the column width is equal to the width of the **ListBox** divided by the number of columns.

This property can't be set to 0 or changed from 0 at run time that is, you can't change a multiple-column **ListBox** to a single-column **ListBox** or a single-column **ListBox** to a multiple-column **ListBox** at run time. However, you can change the number of columns in a multiple-column **ListBox** at run time.

Visual Basic Reference

Columns Property Example

This example illustrates how the two different kinds of **ListBox** controls work when they contain the same data. To try this example, paste the code into the Declarations section of a form that contains two **ListBox** controls. Set the **Columns** property to 2 for List2, and then press F5 and click the form.

```
Private Sub Form_Load ()
   Dim I   ' Declare variable.
   List1.Move 50, 50, 2000, 1750   ' Arrange list boxes.
   List2.Move 2500, 50, 3000, 1750
   For I = 0 To Screen.FontCount -1   ' Fill both boxes with
      List1.AddItem Screen.Fonts(I)   ' names of screen fonts.
      List2.AddItem Screen.Fonts(I)
   Next I
End Sub
```

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

Visual Studio 6.0

ColWidth Property (MSHFlexGrid)

See Also Example Applies To

Returns or sets the width of the column in the specified band, in logical twips. This property is not available at design time.

Note When using the **MSFlexGrid**, this property returns or sets the width of the specified column, in twips. Also, the **ColWidth** syntax, below, cannot be used in an **MSFlexGrid** due to the limitations of this control. If you are using **MSFlexGrid**, use the syntax: *object*.**ColWidth(***number***)** [= *value*]

Syntax

object.**ColWidth(***index, number***)** [= *value*]

The **ColWidth** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	A Long value that specifies which column's width to change. **Note** This is not applicable to the **MSFlexGrid**.
number	A Long value that specifies the band that contains the column. Optional. **Note** In the **MSFlexGrid**, this is a numeric expression that specifies the column.
value	A numeric expression that specifies the width of the specified column, in twips.

Remarks

You can use this property to set the width of any column at run time. For instructions on setting column widths at design time, see the **FormatString** property.

You can set **ColWidth** to zero to create invisible columns, or to 1 to reset the column width to its default value, which depends on the size of the current font.

When *number* is not specified, it defaults to 0. Hence, when the **MSHFlexGrid** is not bound to a hierarchical Recordset, using 0 and not specifying *number* both have the same result. Note that *number* is an optional parameter for backward compatibility with the **MSFlexGrid**.

Visual Basic: MSFlexGrid/MSHFlexGrid Controls

ColWidth Property Example

The following code sets the *value* of the size of the first column. If the **AllowUserResizing** property is **True**, the value changes as the user resizes Column 1.

Note If you are using the MSFlexGrid control, substitute "MSHFlexGrid1" with "MSFlexGrid1."

```
Sub Form1_Load()
   MSHFlexGrid1.AllowUserResizing = True
End Sub

Sub MSHFlexGrid1_MouseUp (Button As Integer, Shift As _
Integer, X As Single, Y As Single)
   MSHFlexGrid1.Text = MSHFlexGrid1.ColWidth(0)
End Sub
```

Visual Basic: Windows Controls

Visual Studio 6.0

ComboItems Property

See Also Example Applies To

Returns a reference to the **ComboItems** collection of **ComboItem** objects.

Syntax

object.**ComboItems**(*index*)

The **ComboItems** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	The index or key of a member in the collection.

Visual Basic: Multimedia MCI Control

Visual Studio 6.0

Command Property (Multimedia MCI Control)

See Also Example Applies To

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

Syntax

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

Remarks

The *cmdstring$* argument gives the name of the **MCI** command to execute: Open, Close, Play, Pause, Stop, Back, Step, Prev, Next, Seek, Record, Eject, Sound, or Save. The command is executed immediately, and the error code is stored in the **Error** property.

The following table describes each command and lists the properties it uses. If a property is not set, either a default value is used (shown in parentheses following the property name), or the property is not used at all (if no default value is shown).

Command	**Description/Properties used**
Open	Opens a device using the MCI_OPEN command.
	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_PAUSE 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 backward using the MCI_STEP command.
	Notify (**False**)
	Wait (**True**)
	Frames
Step	Steps forward 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

Visual Basic: Multimedia MCI Control

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 file Gong.wav. To try this example, first create a form with a **Multimedia MCI** control.

```
Private Sub Form_Load ()
   ' Set properties needed by MCI to open.
   MMControl1.Notify = FALSE
   MMControl1.Wait = TRUE
   MMControl1.Shareable = FALSE
   MMControl1.DeviceType = "WaveAudio"
   MMControl1.FileName = "C:\WINDOWS\MMDATA\GONG.WAV"

   ' Open the MCI WaveAudio device.
   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** control.

```
Private Sub Form_Unload (Cancel As Integer)
   MMControl1.Command = "Close"
End Sub
```

Visual Basic Extensibility Reference

Visual Studio 6.0

CommandBarEvents Property

See Also Example Applies To Specifics

Returns the **CommandBarEvents** object. Read-only.

Settings

The setting for the argument you pass to the **CommandBarEvents** property is:

Argument	Description
vbcontrol	Must be an object of type **CommandBarControl**.

Remarks

Use the **CommandBarEvents** property to return an event source object that triggers an event when a command bar button is clicked. The argument passed to the **CommandBarEvents** property is the command bar control for which the Click event will be triggered.

Visual Basic Extensibility Reference

CommandBarEvents Property Example

The following example uses code including the **CommandBarEvents** property to support any code to handle a mouse click on a command bar.

```
Private WithEvents ce As CommandBarEvents

Sub Test()
    Dim c As CommandBarControl
    Set c = Application.VBE.CommandBars("Tools").Controls(1)
    Set ce = Application.VBE.Events.CommandBarEvents(c)
End Sub

Private Sub ce_Click(ByVal CommandBarControl As Object, Handled As Boolean, CancelDefault As Boolean)
    ' Put event-handling code here
End Sub
```

Visual Basic Extensibility Reference

Visual Studio 6.0

CommandBars Property

See Also Example Applies To Specifics

Contains all of the command bars in a project, including command bars that support shortcut menus.

Visual Basic Reference

Visual Studio 6.0

CommandText Property

See Also Example Applies To

Returns or sets the source of the Command object.

Syntax

object.**CommandText** [=*string*]

Parameters

The **CommandText** event syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
string	A string expression that represents a Command object.

Remarks

This property corresponds to the ADO Command CommandText property.

Visual Basic Reference

Visual Studio 6.0

CommandTimeout Property

See Also Example Applies To

Returns or sets the duration, in seconds, that the provider waits for a command to return from the server.

Syntax

object.**CommandTimeout** [=*number*]

Parameters

The **CommandTimeout** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
number	Integer. A numeric expression that specifies the number of seconds to wait when establishing a connection. The default is 30 seconds.

Remarks

Once this period of time has elapsed without completing the command, the provider raises an exception to the calling application and cancels the command.

This property corresponds to the ADO Command and Connection CommandTimeout property.

Visual Basic Reference

Visual Studio 6.0

CommandType Property

See Also Example Applies To

Informs the provider what the **CommandText** property holds, which may include the source type of the Command object. Setting this property optimizes the execution of the command.

Syntax

object.**CommandType** [=*value*]

The **CommandType** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
value	A constant or value that specifies what the **CommandText** holds, which may include the source type of the Command object.

Remarks

The **CommandText** property can be anything the provider recognizes, hence, you can pass a variety of command types. For example, some providers may allow the **CommandText** property to be set to a base-table name or to the name of an existing stored procedure in the data source. In these cases, the provider must know whether the **CommandText** property contains text or a data source item. Rather than having an order of preference, you can set the type using the **CommandType** property.

When set to **adCmdUnknown**, the command type is not explicitly known, and the provider attempts to execute the command text first as plain text, then as a stored procedure, and finally as a base table name. An error only occurs if all three of these attempts fail.

This property corresponds to the ADO CommandType property.

Visual Basic Reference

Visual Studio 6.0

Comments Property

See Also Example Applies To

Returns or sets a string containing comments about the running application. Read only at run time.

Syntax

object.**Comments**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

You can set this property at design time in the Type box in the Make tab of the Project Properties dialog box.

Visual Basic: MSComm Control

Visual Studio 6.0

CommEvent Property

See Also Example Applies To

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

Syntax

object.**CommEvent**

The **CommEvent** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

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 **CommEvent** property returns one of the following values for communication errors or events. These constants can also be found in the Object Library for this control.

Communication errors include the following settings:

Constant	Value	Description
comEventBreak	1001	A Break signal was received.
comEventFrame	1004	Framing Error. The hardware detected a framing error.
comEventOverrun	1006	Port Overrun. A character was not read from the hardware before the next character arrived and was lost.
comEventRxOver	1008	Receive Buffer Overflow. There is no room in the receive buffer.
comEventRxParity	1009	Parity Error. The hardware detected a parity error.
comEventTxFull	1010	Transmit Buffer Full. The transmit buffer was full while trying to queue a character.

comEventDCB	1011	Unexpected error retrieving Device Control Block (DCB) for the port.

Communications events include the following settings:

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

Data Type

Integer

Visual Basic: MSComm Control

Visual Studio 6.0

CommID Property

See Also Example Applies To

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

Syntax

object.**CommID**

The **CommID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Remarks

This is the same value that's returned by the Windows API **CreateFile** function. Use this value when calling any communications routines in the Windows API.

Data Type

Long

Visual Basic: MSComm Control

Visual Studio 6.0

CommPort Property

See Also Example Applies To

Sets and returns the communications port number.

Syntax

object.**CommPort**[= *value*]

The **CommPort** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A integer value specifying the port number.

Remarks

You can set *value* to any number between 1 and 16 at design time (the default is 1). However, the **MSComm** 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

Visual Basic Reference

Visual Studio 6.0

CompanyName Property

See Also Example Applies To

Returns or sets a string value containing the name of the company or creator of the running application. Read only at run time.

Syntax

object.**CompanyName**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

You can set this property at design time in the Type box in the Make tab of the Project Properties dialog box.

Visual Basic for Applications Reference

Visual Studio 6.0

CompareMode Property

See Also Example Applies To Specifics

Description

Sets and returns the comparison mode for comparing string keys in a **Dictionary** object.

Syntax

object.**CompareMode**[= *compare*]

The **CompareMode** property has the following parts:

Part	Description
object	Required. Always the name of a **Dictionary** object.
compare	Optional. If provided, *compare* is a value representing the comparison mode used by functions such as **StrComp**.

Settings

The *compare* argument can have the following values:

Constant	Value	Description
vbUseCompareOption	1	Performs a comparison using the setting of the **Option Compare** statement.
vbBinaryCompare	0	Performs a binary comparison.
vbTextCompare	1	Performs a textual comparison.
vbDatabaseCompare	2	Microsoft Access only. Performs a comparison based on information in your database.

Remarks

An error occurs if you try to change the comparison mode of a **Dictionary** object that already contains data.

The **CompareMode** property uses the same values as the *compare* argument for the **StrComp** function. Values greater than 2 can be used to refer to comparisons using specific Locale IDs (LCID).

Visual Basic Extensibility Reference

Visual Studio 6.0

CompatibleOLEServer Property

See Also Example Applies To

Returns or sets the compatible ActiveX component of this project.

Syntax

object.**CompatibleOLEServer**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Visual Studio 6.0

Visual Basic: MSChart Control

Component Property

See Also Example Applies To

Returns or sets the type of label to be used to identify the data point.

Syntax

object.**Component** [= *type*]

The **Component** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
type	Integer. A VtChLabelComponent constant that identifies the label type.

Visual Studio 6.0

Visual Basic: MSChart Control

CompositeColumnLabel Property

See Also Example Applies To

Returns the multilevel label string that identifies a column in the data grid associated with a chart.

Syntax

object.**CompositeColumnLabel**(*column*)

The **CompositeColumnLabel** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
column	Integer. Identifies a specific data column. Columns are numbered from left to right beginning with 1. Any columns containing labels are not counted as data columns.

Visual Studio 6.0

Visual Basic: MSChart Control

CompositeRowLabel Property

See Also Example Applies To

Returns the multilevel label string that identifies a row in the data grid associated with a chart.

Syntax

object.**CompositeRowLabel** (*row*)

The **CompositeRowLabel** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
row	Integer. Identifies a specific data row. Rows are numbered from top to bottom beginning with 1. Any rows containing labels are not counted as data rows.

Visual Basic Reference

Visual Studio 6.0

Connect Property

See Also Example Applies To

Returns or sets the connected state of an add-in.

Syntax

object.**Connect**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

Returns **True** if the add-in is registered and currently connected (active).

Returns **False** if the add-in is registered, but not connected (inactive).

Visual Basic: RDO Data Control

Visual Studio 6.0

Connect Property (Remote Data)

See Also Example Applies To

Returns or sets a value that provides information about the source of an open **rdoConnection**. The **Connect** property contains the ODBC connect string. This property is always readable, but cannot be changed after the connection is established.

Syntax

object.**Connect** [= *value*]

The **Connect** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A string expression as described in Remarks. (Data type is String)

Settings

The **Connect** property return value is a **String** expression composed of zero or more parameters separated by semicolons, as described in Remarks.

Remarks

When used with the **rdoQuery** or **rdoConnection** objects, this property is read-only unless created as a stand-alone object when it is read-write until the connection is established. The **Connect** property becomes read-write when the **rdoConnection** object is closed. When used with the **RemoteData control**, this property is read-write.

The **Connect** property is used to pass additional information to and from the ODBC driver manager to establish a connection with a data source. The **Connect** property holds the ODBC connect string which is also used as an argument to the **OpenConnection** method. When used with a stand-alone **rdoConnection** or **rdoQuery** objects, the **Connect** property is used by the **EstablishConnection** method.

Except when associated with the **RemoteData** control, once a connection is made, the **Connect** property is completed with the values optionally supplied by the user and the ODBC driver manager. The **Connect** property of the **rdoQuery** contains this amended connect string.

The **RemoteData** control's **Connect** property is not changed after the connection is established. However, the completed connect string can be extracted from the **RemoteData** control's **Connection** property. For example:

```
FullConnect = MSRDC1.Connection.Connect
```

The following table details valid ODBC connect string arguments and typical usage. Note that each parameter is delineated with a semi-colon (;).

ODBC Connect String Arguments

Parameter	Specifies	Example
DSN	Registered ODBC data source by name.	`DSN=MyDataSource;` (If specified when establishing a DSN-less connection, DSN must be the *last* argument)
UID	User name of a recognized user of the database	`UID=Victoria;`
PWD	Password associated with user name	`PWD=ChemMajor;`
DRIVER	Description of driver. (Note brackets for driver names that include spaces.)	`DRIVER={SQL Server};`
DATABASE	Default database to use once connected	`DATABASE=Pubs;`
SERVER	Name of remote server	`SERVER=SEQUEL;`
WSID	Workstation ID (your system's Net name)	`WSID=MYP5`
APP	Application name. At design time this is set to your project name. At runtime this is your .exe name.	`APP=Accounting`

Note Some ODBC drivers require different parameters not shown in this list.

For example, to set the **Connect** property of a **RemoteData** control you could use code like the following:

```
Dim Cnct As String
Cnct = "DSN=WorkData;UID=Chrissy;"  _
      & "PWD=MIDFLD;DATABASE=WorkDB;"
RemoteData1.Connect = Cnct
RemoteData1.SQL = "Select Name, City " _
   & " From Teams Where Type = 12"
RemoteData1.Refresh
```

You can use this same connect string to establish a new connection:

```
Dim Cn As rdoConnection
Set Cn = rdoEnvironments(0).OpenConnection("", _
rdDriverNoPrompt,True,Cnct$)
```

Note Valid parameters are determined by the ODBC driver. The parameters shown in the preceding example are supported by the Microsoft SQL Server ODBC driver. ODBC, LOGINTIMEOUT and DBQ are not valid parameters of the **RemoteData** control or the **rdoConnection** object's **Connect** property. These parameters are supported by the Microsoft Jet database engine, and not by the ODBC driver. To set login timeout delay, you must use the **LoginTimeout** property of the **rdoEnvironment** object.

Capturing Missing Arguments

If the connect string is null, the information provided by the DSN is incomplete, or invalid arguments are provided, the connection cannot be established. If your code sets the *prompt* argument of the **OpenConnection** method or the **RemoteData** control's **Prompt** property to prohibit user completion of missing ODBC connect arguments, a trappable error is triggered. Otherwise the ODBC driver manager displays a dialog box to gather missing information from the user. Depending on the setting of the **Prompt** argument of the **OpenConnection** or **EstablishConnection** methods, these dialogs capture the DSN from a list of registered ODBC data sources.Names presented to the user, and optionally, the user ID and password. If the connection fails to succeed using these user-provided values, the dialogs are presented again until the connection succeeds or the user cancels the operation. In some cases, the user can create their own DSN using these dialogs.

If a password is required, but not provided in the **Connect** property setting, a login dialog box is displayed the first time a table is accessed by the ODBC driver and each time the connection is closed and reopened.

Connecting with Domain-Managed Security

When connecting to ODBC data sources that support domain-managed security, set the UID and PWD parameters to "". In this case, the Windows NT user name and password are passed to the data source for validation. This strategy permits access to the data source by users with access to the NT domain through authenticated workstation logons.

You can set the **Connect** property for an **rdoConnection** object by providing a *connect* argument to the **OpenConnection** method. Once the connection is established, you can check the **Connect** property setting to determine the DSN, database, user name, password, or ODBC data source of the database.

Registering Data Source Names

Before you can establish a connection using a Data Source Name (DSN), you must either manually register the DSN using the Windows control panel application or use the **rdoRegisterDataSource** method. This process establishes the server name, driver name and other options used when referencing this data source.

Establishing DSN-Less Connections

Under the right circumstances you might not need to pre-register a DSN before connecting. If the following conditions are met, RDO can establish a DSN-less connection using the **RemoteData** control, or the **OpenConnection** or **EstablishConnection** methods with a fully-populated **Connect** property or connect string:

- The connection uses the default named-pipes networking protocol.
- The connection does not set the OEMTOANSI option.
- You specify the name of the server using the SERVER argument in the connect string.
- You specify the name of the ODBC driver using the DRIVER argument in the connect string.
- You set the DSN argument in the connect string (or wherever it appears as in the **DataSourceName** property of the **RemoteData** control) to an empty string. The empty DSN argument must be specified as the *last* parameter of the connect string.

Visual Basic: RDO Data Control

Visual Studio 6.0

Connection Property (Remote Data)

See Also Example Applies To

Returns a reference to a **RemoteData control**'s underlying **rdoConnection** object.

Syntax

object.**Connection**

Set *connection* = *object*.**Connection**

The **Connection** property syntax has these parts:

Part	Description
connection	An object expression that evaluates to a valid **rdoConnection** object.
object	An object expression that evaluates to an object in the Applies To list.

Remarks

When a **RemoteData** control is initialized, **RemoteData** opens a connection to the data source specified in the control's **Connect** property. The **rdoConnection** object created by RDO is exposed by the **Connection** property.

rdoConnection objects have properties and methods you can use to manage data. You can use any method of an **rdoConnection** object, such as **Close** and **Execute**, with the **Connection** property of a **RemoteData** control.

Except when associated with the **RemoteData** control, once a connection is made, the **Connect** property is completed with the values optionally supplied by the user and the ODBC driver manager. The **Connect** property of the **rdoQuery** contains this amended connect string.

The **RemoteData** control's **Connect** property is not changed after the connection is established. However, the completed connect string can be extracted from the **RemoteData** control's **Connection** property. For example:

```
FullConnect = MSRDC1.Connection.Connect
```

Visual Basic Reference

Visual Studio 6.0

ConnectionName Property

See Also Example Applies To

Specifies the name of the **DEConnection** object associated with the Command object. This property can be set to the name of any Connection object in the current DataEnvironment object.

Syntax

object.**ConnectionName** [=*string*]

The **Connection Name** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
string	A string expression that specifies the name of the associated DEConnection object in the current DataEnvironment object.

Visual Basic Reference

Visual Studio 6.0

ConnectionSource Property

See Also Example Applies To

Returns or sets the source of the **DEConnection** object.

Syntax

object.**ConnectionSource** [=*string*]

Parameters

The **ConnectionSource** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
string	A string expression that defines the source of the DEConnection object, which may be a connection string, a Data Link file, or a data source name (DSN).

Remarks

The type of source contained in this property is defined by the setting of the SourceOfData property.

Visual Basic Reference

Visual Studio 6.0

ConnectionTimeout Property

See Also Example Applies To

Returns or sets the duration of time, in seconds, for which the provider attempts to connect to the server specified in the DEConnection object.

Syntax

object.**ConnectionTimeout** [=*integer*]

Parameters

The **ConnectionTimeout** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
integer	A numeric expression that specifies the number of seconds to wait when establishing a connection. The default is 15 seconds.

Remarks

This property corresponds to the ADO ConnectionTimeout property.

Visual Basic Reference

Visual Studio 6.0

ContainedControls Property

See Also Example Applies To

Returns a collection of the controls that were added to the control by the developer or the end user at the controls run-time. The **ContainedControls** property is not available at the controls authoring time, and read-only at the controls run time.

Syntax

object.**ContainedControls**

The **ContainedControls** property syntax has this part:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Remarks

The ContainedControls collection is filled with all the controls that were added to the control by the developer or the end-user. The control can use the ContainedControls collection to perform operations on any of these contained controls.

This collection functions in a similar manner to the Controls collection on a form.

In order to allow contained controls to be placed on the control, the **ControlContainer** property must be **True**.

Contained controls cannot be added or removed through this ContainedControls collection; the contained controls must be changed in whatever manner the container allows.

The **ContainedControls** property may not be supported by all containers, even though the container may support the control having contained controls; Visual Basic forms do support this property. If this property is not supported, then calls to the ContainedControls collection will cause errors; use error handling when accessing the collection. Note, however, that if error handling is done while in an event procedure such as the InitProperties event procedure or the ReadProperties event procedure, the error handler should not raise an error event; doing this may be fatal to the container.

The ContainedControls collection is not available when the Initialize event is raised; but is available when the InitProperties event or ReadProperties event is raised.

Once the ContainedControls collection is present, it may not immediately contain references to the controls a developer has placed on the control. For example, if the control is on a Visual Basic form, the **Count** property of the ContainedControls collection will be zero until after the ReadProperties event procedure has executed.

Visual Basic Extensibility Reference

Visual Studio 6.0

ContainedVBControls Property

See Also Example Applies To

Returns the collection of contained controls.

Syntax

object.**ContainedVBControls**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Visual Basic Reference

Visual Studio 6.0

Container Property

See Also Example Applies To

Returns or sets the container of a control on a **Form**. Not available at design time.

Syntax

Set *object*.**Container** [= *container*]

The **Container** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
container	An object expression that evaluates to an object that can serve as a container for other controls, as described in Remarks.

Remarks

The following controls can contain other controls:

- **Frame** control
- **PictureBox** control
- **SSTab** control

Visual Basic Reference

Container Property Example

This example demonstrates moving a **CommandButton** control from container to container on a **Form** object. To try this example, paste the code into the Declarations section of a form that contains a **Frame** control, a **PictureBox** control and a **CommandButton**, and then press F5.

```
Private Sub Form_Click()
   Static intX As Integer
   Select Case intX
      Case 0
         Set Command1.Container = Picture1
         Command1.Top= 0
         Command1.Left= 0
      Case 1
         Set Command1.Container = Frame1
         Command1.Top= 0
         Command1.Left= 0
      Case 2
         Set Command1.Container = Form1
         Command1.Top= 0
         Command1.Left= 0
   End Select
   intX = intX + 1
End Sub
```

Visual Basic Extensibility Reference

Visual Studio 6.0

Container Property

See Also Example Applies To

Returns the containing control or form.

Syntax

object.**Container**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Visual Basic Reference

Visual Studio 6.0

ContainerHWnd Property

See Also Example Applies To

Returns the window handle (hWnd) of a UserControls container.

Syntax

object.**ContainerHWnd**

The object placeholder represents an object expression that evaluates to a UserControl object.

Remarks

The **ContainerHWnd** property returns the handle of the container on which a UserControl object is sited. The **ContainerHWnd** can be passed as a parameter to Windows API calls that require an hWnd parameter.

When the **Windowless** property of a UserControl is set to True, the UserControl doesnt have a handle of its own. In some cases, the **ContainerHWnd** property can be used in place of the **hWnd** property for UserControls to get information about a window. The **ContainerHWnd** should not be used for Windows API calls that modify the **hWnd** value.

Visual Basic Reference

Visual Studio 6.0

Contents Property

See Also Example Applies To

Returns or sets a byte array representing the contents of a PropertyBag object.

Remarks

The **Contents** property is used to store or retrieve the contents of a **PropertyBag** object to or from an external storage such as a text file, a cell in a spreadsheet, a message field, or a structured storage. An object stored in this manner can be retrieved and reused without using the New keyword. The **Contents** property must be passed as a byte array; attempting to assign to or from any other data type will result in a run time error.

This property only applies to a **PropertyBag** declared As PropertyBag; it does not apply to a **PropertyBag** object contained in a UserControl or UserDocument.

Visual Basic Reference

Visual Studio 6.0

ContinuousScroll Property

See Also Example Applies To

Returns or sets a value that determines if scrolling is continuous, or if the **UserDocument** only redraws when the scroll thumb is released.

Syntax

object.**ContinuousScroll** = *boolean*

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression that specifies whether scrolling is continuous or not.

Settings

The settings for boolean are:

Setting	Description
True	Default. Scrolling is continuous.
False	The UserDocument redraws only when the thumb is released.

Visual Basic Reference

Visual Studio 6.0

Control Property

See Also Example Applies To

Returns or sets which control to use when a **DEAggregate** or **DEField** object is dragged from a Data Environment designer and dropped onto a Visual Basic form or report.

Syntax

object.**Control** [=*string*]

The **Control** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
string	A string expression that specifies the ClassID of the control.

Remarks

The *string* is a globally unique identifier (GUID) for the control. For example, the value "{0ECD9B64-23AA-11D0-B351-00A0C9055D8E}" is the string for the **MSHFlexGrid** control.

Visual Basic Reference

Visual Studio 6.0

ControlBox Property

See Also Example Applies To

Returns or sets a value indicating whether a Control-menu box is displayed on a form at run time. Read-only at run time.

Syntax

object.**ControlBox**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Settings

The **ControlBox** property settings are:

Setting	Description
True	(Default) Displays the Control-menu box.
False	Removes the Control-menu box.

Remarks

To display a Control-menu box, you must also set the form's **BorderStyle** property to 1 (Fixed Single), 2 (Sizable), or 3 (Fixed Dialog).

Both modal and modeless windows can include a Control-menu box.

The commands available at run time depend on the settings for related properties for example, setting **MaxButton** and **MinButton** to **False** disables the Maximize and Minimize commands on the Control menu, but the Move and Close commands remain available.

Note Settings you specify for the **ControlBox**, **BorderStyle**, **MaxButton**, and **MinButton** properties aren't reflected in the form's appearance until run time.

Visual Basic Reference

Visual Studio 6.0

ControlContainer Property

See Also Example Applies To

Returns or sets a value determining if a control can contain controls placed on it by the developer or the end user at the controls run time; in the same way the **PictureBox** control can contain other controls. The **ControlContainer** property is read/write at the controls authoring time, and not available at the controls run time.

Settings

The settings for **ControlContainer** are:

Setting	Description
True	The control can contain controls placed on it. If an instance of this control is placed on a container that is not aware of ISimpleFrame, support of contained controls will be disabled. The control will continue to work correctly in all other ways, but developers or end users will be unable to place controls on an instance of this control.
False	(Default) The control cannot contain controls placed on it.

Remarks

Contained control support does work on a Visual Basic form.

Contained controls placed on a control with a transparent background are only visible where their location overlaps any constituent controls. Mouse events will be passed to the contained control only if they occur where the contained control is visible.

Visual Basic Extensibility Reference

Visual Studio 6.0

ControlObject Property

See Also Example Applies To

Returns a reference to an instance of the design-time IDispatch pointer provided by the control. If there isnt one, this property returns **Nothing**.

Syntax

object.**ControlObject**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

For example, the **Toolbar** control provides an object through a **Property Page's ControlObject** property to set the number of buttons.

Visual Basic Reference

Visual Studio 6.0

Controls Property

See Also Example Applies To

Returns a reference to a collection of **Control** objects.

Syntax

object.**Controls**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

You can manipulate **Control** objects using the reference returned by the **Controls** property.

Visual Basic: Windows Controls

Visual Studio 6.0

Controls Property (Toolbar Control)

See Also Example Applies To

Returns a reference to a collection of controls contained on an object.

Syntax

object.**Controls**(*index*)

object.**Controls.Item**(*index*)

The **Controls** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	A value that identifies a member of a Controls collection.

Remarks

The **Controls** property is similar to the **Controls** collection on the **Form** object, and is accessed in a similar manner. For example, use the following code to get the **Top** property of the second control on a **Toolbar** control:

```
MsgBox Toolbar1.Controls(2).Top
```

With the **Controls** property, you can iterate through all the controls on a **Tab** object or a **Toolbar** control and change the properties of each control as in the following code:

```
Dim ctlX As Control
For Each ctlX In Toolbar1.Controls
   ctlX.Width = _
   Toolbar1.Width / Toolbar1.Controls.Count
Next
```

Note The **Controls** collection refers to controls contained by the **Toolbar** control, such as a **ComboBox** control, and not the **Button** objects, which are part of the control itself.

Visual Basic Extensibility Reference

Visual Studio 6.0

ControlType Property

See Also Example Applies To

Returns the type of run-time window that a control creates.

Syntax

object.**ControlType As vbext_ControlType**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Settings

The settings for **vbext_ControlType** are:

Constant	Value	Description
vbext_ct_Light	1	(Default) No hWnd at run-time.
vbext_ct_Standard	2	hWnd at run-time.
vbext_ct_Container	3	hWnd at run-time, and can contain other controls.

Visual Basic Reference

Visual Studio 6.0

Copies Property

See Also Example Applies To

Returns or sets a value that determines the number of copies to be printed. For the **Printer** object, not available at design time.

Syntax

object.**Copies** [= *number*]

The **Copies** property syntax has these parts:

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
Number	A numeric expression that specifies the number of copies to print. This value must be an integer.

Remarks

For the Print dialog box, this property returns the number of copies entered by the user in the Copies box. If the **cdlPDUseDevModeCopies** flag is set for the **CommonDialog** control, this property always returns 1.

For the **Printer** object, multiple copies may or may not be collated, depending on the printer driver. Multiple copies of the entire document or multiple copies of each page may be printed. For printers that don't support collating, set **Copies** = 1, and then use a loop in code to print multiple copies of the entire document.

Note The effect of the properties of the **Printer** object depends on the driver supplied by the printer manufacturer. Some property settings may have no effect, or several different property settings may all have the same effect. Settings outside the accepted range may produce an error. For more information, see the manufacturer's documentation for the specific driver.

Visual Basic Reference

Visual Studio 6.0

Count Property (ActiveX Controls)

See Also Example Applies To

Returns the number of objects in a collection.

Syntax

object.**Count**

The *object* placeholder is an object expression that evaluates to an object in the Applies To list.

Remarks

You can use this property with a **For...Next** statement to carry out an operation on the forms or controls in a collection. For example, the following code moves all controls on a form 0.5 inches to the right (**ScaleMode** property setting is 1 or **vbTwips**):

```
For I = 0 To Form1.Controls.Count - 1
   Form1.Controls(I).Left = Form1.Controls(I).Left + 720
Next I
```

You can also use this kind of structure to quickly enable or disable all controls on a form.

When used with the **If TypeOf** statement, you can cycle through all controls and change, for example, the **Enabled** property setting of only the text boxes or the **BackColor** property setting of only the option buttons.

Visual Basic Reference

Visual Studio 6.0

Count Property (DEDesigner Extensibility)

See Also Example Applies To

Returns the number, or count, of members contained in a collection. A member can be any **DECommand**, **DEConnection**, **DEParameter**, **DEField**, **DEAggregates**, or **DERelationCondition** object contained in the DataEnvironment object.

Syntax

object.**Count** [=*value*]

The **Count** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
value	Long. A numeric expression that specifies the number of members contained in the collection.

Remarks

For additional information, see Visual Basic's Count property.

Visual Basic for Applications Reference

Visual Studio 6.0

Count Property

See Also Example Applies To Specifics

Description

Returns the number of items in a collection or **Dictionary** object. Read-only.

Syntax

object.**Count**

The *object* is always the name of one of the items in the Applies To list.

Remarks

The following code illustrates use of the **Count** property:

```
Dim a, d, i             'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"     'Add some keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
a = d.Keys              'Get the keys
For i = 0 To d.Count -1 'Iterate the array
    Print a(i)          'Print key
Next
...
```

Visual Basic Reference

Visual Studio 6.0

Count Property (VB Collections)

See Also Example Applies To

Returns the number of objects in a collection.

Syntax

object.**Count**

The *object* placeholder is an object expression that evaluates to an object in the Applies To list.

Remarks

You can use this property with a **For...Next** statement to carry out an operation on the forms or controls in a collection. For example, the following code moves all controls on a form 0.5 inches to the right (**ScaleMode** property setting is 1 or **vbTwips**):

```
For I = 0 To Form1.Controls.Count - 1
   Form1.Controls(I).Left = Form1.Controls(I).Left + 720
Next I
```

You can also use this kind of structure to quickly enable or disable all controls on a form.

When used with the **If TypeOf** statement, you can cycle through all controls and change, for example, the **Enabled** property setting of only the text boxes or the **BackColor** property setting of only the option buttons.

Visual Basic Extensibility Reference

Visual Studio 6.0

Count Property

See Also Example Applies To Specifics

Returns a Long containing the number of items in a collection. Read-only.

Visual Basic Extensibility Reference

Count Property Example

The following example uses the **Count** property to return the number of **VBComponent** objects in a particular project.

```
Debug.Print Application.VBE.VBProjects(1).VBComponents.Count
```

Visual Basic for Applications Reference

Visual Studio 6.0

Count Property

See Also Example Applies To Specifics

Returns a Long (long integer) containing the number of objects in a collection. Read-only.

Visual Basic for Applications Reference

Count Property Example

This example uses the **Collection** object's **Count** property to specify how many iterations are required to remove all the elements of the collection called `MyClasses`. When collections are numerically indexed, the base is 1 by default. Since collections are reindexed automatically when a removal is made, the following code removes the first member on each iteration.

```
Dim Num, MyClasses
For Num = 1 To MyClasses.Count   ' Remove name from the collection.
   MyClasses.Remove 1   ' Default collection numeric indexes
Next   ' begin at 1.
```

Visual Basic Extensibility Reference

Visual Studio 6.0

CountOfDeclarationLines Property

See Also Example Applies To Specifics

Returns a Long containing the number of lines of code in the Declarations section of a code module. Read-only.

Visual Basic Extensibility Reference

CountOfDeclarationLines Property Example

The following example uses the **CountOfDeclarationLines** property to return the number of declaration lines in a particular code pane.

```
Debug.Print Application.VBE.CodePanes(2).CodeModule.CountOfDeclarationLines
```

Visual Basic Extensibility Reference

Visual Studio 6.0

CountOfLines Property

See Also Example Applies To Specifics

Returns a Long containing the number of lines of code in a code module. Read-only.

Visual Basic Extensibility Reference

CountOfLines Property Example

The following example uses the **CountOfLines** property to return the total number of lines in a particular code pane.

```
Application.VBE.CodePanes(2).CodeModule.CountOfLines
```

Visual Basic Extensibility Reference

Visual Studio 6.0

CountOfVisibleLines Property

See Also Example Applies To Specifics

Returns a Long containing the number of lines visible in a code pane. Read-only.

Visual Basic Extensibility Reference

CountOfVisibleLines Property Example

The following example uses the **CountOfVisibleLines** property to return the number of lines visible at one time in a particular code pane, based on the height of the pane.

```
Debug.Print Application.VBE.Codepanes(3).CountOfVisibleLines
```

Visual Basic: MSComm Control

Visual Studio 6.0

CTSHolding Property

See Also Example Applies To

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

Syntax

object.**CTSHolding**

The **CTSHolding** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

The following table lists the **CTSHolding** property settings for the **MSComm** control.

Setting	Description
True	Clear To Send line high.
False	Clear To Send line low.

Remarks

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

Boolean

Visual Basic: DataGrid Control

Visual Studio 6.0

CurrentCellModified Property

See Also Example Applies To

Sets or returns modification status of the current cell. Not available at design time.

Syntax

object.**CurrentCellModified** [= *value*]

The **CurrentCellModified** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A Boolean expression that determines the modification status of the current cell, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
True	Editing is in progress and the current cell (indicated by the **Bookmark** and **Col** properties) has been modified by the user.
False	The cell has not been modified or editing is not in progress.

Remarks

You can use this property to cancel any changes the user has made to the current text. For example, to program a function key to discard the user's changes (like the ESC key), trap the key code in the grid's KeyDown event and set **CurrentCellModified** to **False**. This will revert the current cell to its original contents.

Visual Basic: DataGrid Control

Visual Studio 6.0

CurrentCellVisible Property

See Also Example Applies To

Sets or returns the visibility of the current cell. Not available at design time.

Syntax

object.**CurrentCellVisible** [= *value*]

The **CurrentCellVisible** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A Boolean expression that determines the visibility of the current cell, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
True	The current cell (indicated by the **Bookmark** and **Col** properties) is visible within the displayed area of a grid or split.
False	The cell is not visible.

Remarks

For a **DataGrid** control, setting the **CurrentCellVisible** property to **True** causes the grid to scroll so that the current cell is brought into view. If a grid contains multiple splits, then the current cell becomes visible in each split.

For a **Split** object, setting the **CurrentCellVisible** property to **True** makes the current cell visible in that split only.

In all cases, setting this property to **False** is meaningless and is ignored.

Visual Basic: DataRepeater Control

Visual Studio 6.0

CurrentRecord Property

See Also Example Applies To

Returns the bookmark of the current record.

Syntax

object.**CurrentRecord**

The *object* placeholder represents an object expression that resolves to an object in the Applies To list.

Return Type

Variant

Visual Basic: DataRepeater Control

CurrentRecord, VisibleRecords Properties Example

The example causes the current record to become the first record visible. If the user scrolls the **DataRepeater** control so that the current record is hidden, invoking the procedure causes the current record to reappear.

```
Private Sub MakeFirstVisibleRecord()
   DataRepeater1.VisibleRecords(1)=DataRepeater1.CurrentRecord
End Sub
```

Visual Basic Reference

Visual Studio 6.0

CurrentX, CurrentY Properties

See Also Example Applies To

Return or set the horizontal (**CurrentX**) or vertical (**CurrentY**) coordinates for the next printing or drawing method. Not available at design time.

Syntax

object.**CurrentX** [= *x*]
object.**CurrentY** [= *y*]

The **CurrentX** and **CurrentY** properties syntax have these parts:

Part	Description
Object	An object expression that evaluates to an object in the Applies To list.
X	A number that specifies the horizontal coordinate.
Y	A number that specifies the vertical coordinate.

Remarks

Coordinates are measured from the upper-left corner of an object. The **CurrentX** property setting is 0 at an object's left edge, and the **CurrentY** property setting is 0 at its top edge. Coordinates are expressed in twips, or the current unit of measurement defined by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, **ScaleTop**, and **ScaleMode** properties.

When you use the following graphics methods, the **CurrentX** and **CurrentY** settings are changed as indicated:

This method	Sets CurrentX, CurrentY to
Circle	The center of the object.
Cls	0, 0.
EndDoc	0, 0.
Line	The end point of the line.
NewPage	0, 0.

Print	The next print position.
Pset	The point drawn.

Visual Basic: RDO Data Control

Visual Studio 6.0

CursorDriver Property (Remote Data)

See Also Example Applies To

Returns or sets a value that specifies the type of cursor to be created.

Syntax

object.**CursorDriver** [= *value*]

The **CursorDriver** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An **Integer** or constant as described in Settings.

Settings

Constant	Value	Description
rdUseIfNeeded	0	The ODBC driver will choose the appropriate style of cursors. Server-side cursors are used if they are available.
rdUseOdbc	1	**RemoteData** will use the ODBC cursor library.
rdUseServer	2	Use server-side cursors.
rdUseClientBatch	3	RDO will use the optimistic batch cursor library.

Remarks

The **CursorDriver** property only affects connections established *after* the **CursorDriver** property has been set the property is read-only on existing connections.

When the initial (default), and each subsequent **rdoEnvironment** object is created, the **CursorDriver** property is set from the **rdoEngine** object's **rdoDefaultCursorDriver** property which is set using the same constants.

Choosing a Cursor Driver

Choosing the right cursor driver can have a significant impact on the overall performance of your application, what resources are consumed by the cursor, and limit the type or complexity of the cursors you create. Each type of cursor has its own benefits and limitations. In many cases, the best choice is no cursor at all because your application often does not need to scroll through the data or perform update operations against a keyset.

The following paragraphs outline the functionality and suggested purposes for each of the cursor types.

- Server-Side Cursors
  - This cursor library maintains the cursor keyset on the server (in *TempDB*) which eliminates the need to transmit the keyset to the workstation where it consumes needed resources. However, this cursor driver consumes *TempDB* space on the remote server so this database must be expanded to meet this requirement. Cursors created with the server-side driver cannot contain more than one SELECT statement if they do, a trappable error is fired. You can still use the server-side cursor driver with multiple result set queries if you disable the cursor by creating a forward-only, read-only cursor with a rowset size of one. Not all remote servers support server-side cursors. Note that server-side cursors are enabled when using either **rdUseIfNeeded** or **rdUseServer** against Microsoft SQL Server databases.
- ODBC Client-Side Cursors
  - This cursor library builds keysets on the workstation in local RAM overflowing to disk if necessary. Because of this design considerably more network operations must be performed to initially create the keyset, but with small cursors this should not impose a significant load on the workstation or network. ODBC client-side cursors do not impose any type of restriction on the type of query executed. This option gives better performance for small result sets, but degrades quickly for larger result sets.
- Client-Batch Cursors
  - This cursor library is designed to deal with the special requirements of optimistic batch updates and several other more complex cursor features. Client-batch cursors are required for dissociate connections, batch mode, and multi-table updates. This cursor also supports delayed BLOB column fetch, buffered cursors, and additional control over updates. This library is somewhat larger than the others, but also performs better in many situations.
- The No-Cursor Option
  - In cases where you need to fetch rows quickly, or perform action queries against the database without the overhead of a cursor, you can choose to instruct RDO to bypass creation of a cursor. Basically, this option creates a forward-only, read-only result set with a **RowsetSize** set to 1. This option can improve performance in many operations. While you cannot update rows or scroll between rows with this cursor, you can submit independent action queries to manipulate data. This option is especially useful when accessing data through stored procedures.

Visual Basic: RDO Data Control

Visual Studio 6.0

CursorDriver, rdoDefaultCursorDriver Property Constants

See Also

These constants are used to determine which type of RDO cursor library is chosen by the ODBC Driver Manager.

Constant	Value	Description
rdUseIfNeeded	0	The ODBC driver will choose the appropriate style of cursors. Server-side cursors are used if they are available.
rdUseOdbc	1	The RDO layer will use the ODBC cursor library. This gives better performance for small result sets but degrades quickly for larger result sets.
rdUseServer	2	Use server-side cursors. For most large operations this will give better performance, but can cause more network traffic.
rdUseClientBatch	3	Use the client batch cursor library used for batch update operations.
rdUseNone	4	Do not open a cursor. Create a forward-only, read-only single-row result set. This uses data access fetch techniques similar to those used by VBSQL and ODBC API access methods.

Visual Basic Reference

Visual Studio 6.0

CursorLocation Property

See Also Example Applies To

Returns or sets the location of the cursor library for the **DEConnection** or **DECommand** object.

Syntax

object.**CursorLocation** [=*value*]

The **CursorLocation** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
value	A constant or numeric expression that specifies the cursor library location.

Remarks

Any **DECommand** object that is associated with a **DEConnection** object that has this property set automatically obtains the same setting. However, the setting can be overwritten while the Recordset is closed. When the Recordset is open, this property is read-only.

This property corresponds to the ADO Command and Connection CursorLocation property.

Visual Basic Reference

Visual Studio 6.0

CursorType Property (DEDesigner Extensibility)

See Also Example Applies To

Returns or sets the type of cursor used when opening the **DECommand** object. You can only set this property when the cursor is closed.

Syntax

object.**CursorType** [=*value*]

The **CursorType** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an item in the Applies To list.
value	A value or constant that specifies the type of cursor used.

Remarks

If a provider cannot support the requested cursor type, the closest cursor type is used. If the **CursorLocation** property is **adUseClient**, the only cursor type supported is **adOpenStatic**. If the **CursorLocation** property is **adUseServer**, the available cursor types depend on the OLE DB provider or ODBC driver being used.

This property corresponds to the ADO Recordset CursorType property.

This documentation is archived and is not being maintained.

Visual Basic: RDO Data Control

Visual Studio 6.0

CursorType Property

See Also Example Applies To

Returns or sets a value that specifies the default type of cursor to use when opening a result set from the specified query.

Syntax

object.**CursorType** [= *value*]

The **CursorType** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A **Long** integer representing the type of cursor as described by one of the following constants:

Constant	Value	rdoResultset type
rdOpenForwardOnly	0	(Default) Fixed set, non-scrolling**.**
rdOpenKeyset	1	Updatable, fixed set, scrollable query result set cursor.
rdOpenDynamic	2	Updatable, dynamic set, scrollable query result set cursor.
rdOpenStatic	3	Read-only, fixed set.

Remarks

Determines the cursor type to use when opening an **rdoResultset** object from this query.

When creating a stand-alone **rdoQuery** object whose query is to be used as a method, you should set the **CursorType** before the query is executed because there is no option to do so when the query is executed.

The value of the **CursorType** property is used as the **Type** argument of the **OpenResultset** method.

Not all cursor libraries support all types of cursors. For example, the ODBC client-side driver can only support **rdOpenStatic** and **rdOpenForwardOnly** cursor types, while the SQL Server server-side driver supports all four types. Generally, most drivers support forward-only and static cursors.

Visual Basic: RDO Data Control

CursorType Property Example

```
Option Explicit
Dim er As rdoError
Dim cn As New rdoConnection
Dim qy As New rdoQuery
Dim rs As rdoResultset
Dim col As rdoColumn

Private Sub Form_Load()

cn.CursorDriver = rdUseClientBatch
cn.Connect = "uid=;pwd=;server=sequel;" _
   & "driver={SQL Server};database=pubs;dsn='';"
cn.EstablishConnection

'
'   Setup the query
'
With qy
    .Name = "ShowAuthor"
    .SQL = "Select * from Authors " _
      & "where Au_LName = ? "
    .LockType = rdConcurReadOnly
    .CursorType = rdOpenForwardOnly
    .RowsetSize = 1
    Set .ActiveConnection = cn
End With

'
'   Execute the Query by Name
'   Pass in a parameter to the query
'
cn.ShowAuthor "White"
'
'   Process the resulting rows
'

If cn.LastQueryResults is Nothing then
Else
   Set rs = cn.LastQueryResult
   For Each col In rs.rdoColumns
       Print col.Name,
   Next
   Print String(80, "-")

   Do Until rs.EOF
       For Each col In rs.rdoColumns
           Print col,
       Next
       Print
       rs.MoveNext
   Loop
End if
```

```
End Sub
```

Visual Studio 6.0

Visual Basic: MSChart Control

Custom Property

See Also Example Applies To

Returns or sets a value that determines if custom text is used to label a data point on a chart.

Syntax

object.**Custom** [= *boolean*]

The **Custom** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression that specifies whether custom text is used, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	The label contains custom text.
False	(Default) Information specified by the **DataPointLabel** object's **Components** property is used to label the data point.

Visual Basic: Windows Controls

Visual Studio 6.0

CustomFormat Property

See Also Example Applies To

Returns or sets a value that specifies the format used by the control when displaying date/time information.

Syntax

object.**CustomFormat** [= *string*]

The **CustomFormat** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
string	A string expression specifying the format of the display.

Settings

The DateTimePicker control supports the following format characters:

String Fragment	Description
d	The one- or two-digit day.
dd	The two-digit day. Single digit day values are preceded by a zero.
ddd	The three-character weekday abbreviation.
dddd	The full weekday name.
h	The one- or two-digit hour in 12-hour format.
hh	The two-digit hour in 12-hour format. Single digit values are preceded by a zero.
H	The one- or two-digit hour in 24-hour format.
HH	The two-digit hour in 24-hour format. Single digit values are preceded by a zero.

m	The one- or two-digit minute.
mm	The two-digit minute. Single digit values are preceded by a zero.
M	The one- or two-digit month number.
MM	The two-digit month number. Single digit values are preceded by a zero.
MMM	The three-character month abbreviation.
MMMM	The full month name.
s	The one- or two- digit seconds.
ss	The two-digit seconds. Single digit values are proceeded by a zero.
t	The one-letter AM/PM abbreviation (that is, "AM" is displayed as "A").
tt	The two-letter AM/PM abbreviation (that is, "AM" is displayed as "AM").
X	A callback field. The control still uses the other valid format characters, and queries the owner to fill in the "X" portion. The owner must be ready to respond to events that request information about how to fill in these fields. Multiple 'X' characters can be used in series to signify unique callback fields.
y	The one-digit year (that is, 1997 would be displayed as "7").
yy	The last two digits of the year (that is, 1997 would be displayed as "97").
yyy	The full year (that is, 1997 would be displayed as "1997").

Remarks

The **CustomFormat** property acts as an input mask. To display a custom format, the **Format** property must be set to **dtpCustom**

You can add body text to the format string. For example, if you want the control to display the current date with the format "Today is:Friday July 25, 1997 Time: 8:34 AM", the **CustomFormat** string is "`'Today is:' ddddMMMMdd, yyy ' Time: 'h:mtt`". Body text must be enclosed in single quotation marks.

One of the custom format fields described above is a callback field. A callback field allows you to customize output by specifying certain parts of a format string as callback fields. To declare a callback field, you must include one or more 'X' characters (ASCII Code 88) anywhere in the body of the format string. Callback fields are displayed in left-to-right order.

When a new date is displayed in a format that includes one or more callback fields, the Format and FormatSize events are raised for each callback field. You can use the Format event to specify a custom response string, and the FormatSize event to determine the space needed to display the string. This behavior gives you complete control of how a callback field is displayed.

Each sequence of X's has a specific meaning. For example, X might mean "st/nd/rd/th" (for "1st" "2nd" "3rd" or "4th") and "XX" may mean "first" "second" "third" or "fourth". These fields do not format the user's text. They format the date into a displayable format.

For example, let's say you want to display the month in Spanish as well as English in a format like this:

```
July (Julio) 29
```

You would create a format string that looked like this:

```
MMMM XXXX d
```

When processing the Format and FormatSize events, you can check which callback field is being called by comparing the input format sting with "XXXX". If the field string matches, an output string "(Julio)" can be built and the length of the output string can be supplied.

The number of Xs is only used by an application to determine what text to supply for a callback field. When processing the FormatSize event, the size of the text can be programmatically calculated.

You can create unique callback fields by repeating the "X" character. Thus, the format string "XXddddMMMdd', 'yyyXXX" contains two callback fields. Callback fields are treated as valid fields, so the application must be prepared to handle Key events.