# Visual Basic: MSTab Control

**Visual Studio 6.0**

# Tab Property (SSTab Control)

See Also Example Applies To

Returns or sets the current tab for an **SSTab** control.

**Syntax**

*object*.**Tab** [ = *tabnumber* ]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tabnumber* | A numeric expression that indicates a specific tab. The first tab is always 0. |

**Remarks**

The current tab moves to the front and becomes the active tab.

Typically, the user of your application clicks a tab to make it the current tab. However, you may need to select the current tab in code. For example, you may want the same tab to be the current tab each time you display a certain dialog box in your application. If you dismiss the dialog box by using the **Hide** method of the **Form**, the last tab to be the active tab when the **Form** was hidden will be the active tab the next time the dialog box appears. You can set the **Tab** property of the **SSTab** control so the same tab is active every time the dialog box appears.

# Visual Basic: MSTab Control

# Tab Property (SSTab Control) Example

This example always makes the first tab in the **SSTab** control the active tab just before showing the form which contains the control. To try this example, create two **Form** objects. Place a **CommandButton** control on Form1 and an **SSTab** control on Form2. Paste the code into the Click event of the **CommandButton** on Form1, and then run the example.

```
Private Sub Command1_Click()
   Form2.SSTab1.Tab = 1
   Form2.Show
End Sub
```

# Visual Basic: DataGrid Control

**Visual Studio 6.0**

# TabAcrossSplits Property

See Also Example Applies To

Sets or returns the behavior of the tab and arrow keys at split borders.

### Syntax

*object*.**TabAcrossSplits** [= *value*]

The **TabAcrossSplits** 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 behavior of the tab and arrow keys at split borders, as described in Settings. |

### Settings

The settings for *value* are:

| Setting | Description |
|---|---|
| **True** | Tab and arrow keys will move the current cell across split boundaries. When at the last column of the rightmost split (or the first column of the leftmost split), they will either wrap to the next row, stop, or move to other controls depending on the values of the **WrapCellPointer** and **TabAction** properties. |
| **False** | (Default) The tab and arrow keys will not move the current cell across split boundaries. They will either wrap to the next row, stop, or move to other controls depending on the values of the **WrapCellPointer** and **TabAction** properties. |

### Remarks

The **TabAcrossSplits** property does not determine if the tab and arrow keys will move from cell to cell, or from control to control, or wrap to the next row. Use the **AllowArrows**, **WrapCellPointer**, and **TabAction** properties to control this behavior. If the tab and arrow keys are able to move from cell to cell, this property determines whether they will move across split boundaries to adjacent splits.

# Visual Basic: DataGrid Control

**Visual Studio 6.0**

# TabAction Property

See Also Example Applies To

Sets or returns a value that defines the behavior of the tab key.

**Syntax**

*object*.**TabAction** [= *value*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *value* | A number or constant that defines the behavior of the tab key, as described in Settings. |

**Settings**

The settings for *value* are:

| Constant | Value | Description |
|---|---|---|
| **dbgControlNavigation** | 0 | (Default) The tab key moves to the next or previous control on the form. |
| **dbgColumnNavigation** | 1 | The tab key moves the current cell to the next or previous column. However, if this action would cause the current row to change, then the next or previous control on the form receives focus. |
| **dbgGridNavigation** | 2 | The tab key moves the current cell to the next or previous column. The behavior of the tab key at row boundaries is determined by the **WrapCellPointer** property. When this setting is used, the tab key never results in movement to another control. |

**Remarks**

The **TabAction** property does not determine if the tab key will cross split boundaries. Use the **TabAcrossSplits** property to control this behavior.

The value of the **TabAction** property overrides the **WrapCellPointer** property behavior. For example, if **WrapCellPointer** is **True** and **TabAction** is set to **dbgColumnNavigation**, and you're on the last column of the **DataGrid**, pressing the Tab key brings you to the next control in the tab index order instead of to the first column of the next row.

# Visual Basic: MSTab Control

**Visual Studio 6.0**

# TabCaption Property (SSTab Control)

See Also Example Applies To

Returns or sets the caption for each tab for an **SSTab** control.

**Syntax**

*object*.**TabCaption**(*tab*) [ = *text* ]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tab* | A numeric expression that specifies the tab you want the caption to appear on. |
| *text* | A string expression that evaluates to the text displayed as the caption for the specified tab. |

**Remarks**

At design time, you can set the **TabCaption** property by clicking a tab and then setting the **Caption** property in the Properties window. Or you can select (Custom) in the Properties window and set the **TabCaption** property in the General tab of the Properties dialog box.

At run time, you can read or change the caption of any tab using the **TabCaption** property. You can also use the **Caption** property to change the **TabCaption** property for just the active tab.

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

# Visual Basic: MSTab Control

# TabCaption Property (SSTab Control) Example

This example adds or removes an extra word from the tabs of an **SSTab** control that lists the defensive players of a sport on one tab and the offensive players on another tab. By clicking the **CheckBox** control on the **Form**, the user can toggle between longer captions or shorter ones.

```
Private Sub Check1_Click()
   Dim X As Integer
   For X = 0 To SSTab1.Tabs - 1
      Select Case Check1.Value
         Case 0   ' Toggle to short captions.
            SSTab1.TabCaption(X) = Left(SSTab1.TabCaption(X), 7)
         Case 1   ' Toggle to long captions.
            SSTab1.TabCaption(X) = SSTab1.TabCaption(X) & " Players"
      End Select
   Next X
End Sub
```

# Visual Basic: MSTab Control

**Visual Studio 6.0**

# TabEnabled Property (SSTab Control)

See Also Example Applies To

Returns or sets a value that determines whether a tab in an **SSTab** control is available when clicked.

**Syntax**

*object*.**TabEnabled**(*tab*)[ = *boolean* ]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tab* | A numeric expression that specifies the tab. |
| *boolean* | A Boolean expression that specifies if the tab will respond to being clicked, as described in Settings. |

**Settings**

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) The tab responds when clicked. |
| **False** | The tab doesn't respond when clicked. |

**Remarks**

When a tab is disabled, the text on the tab appears dimmed and the user cannot select that tab.

The **TabEnabled** property enables or disables a single tab. Use the **Enabled** property to enable or disable the entire **SSTab** control.

# Visual Basic: Windows Controls

**Visual Studio 6.0**

# TabFixedHeight, TabFixedWidth Properties

See Also Example Applies To

Return or set the fixed height and width of all **Tab** objects in a **TabStrip** control, but only if the **TabWidthStyle** property is set to **tabFixed**.

**Syntax**

*object*.**TabFixedHeight** [= *integer*]

*object*.**TabFixedWidth** [= *integer*]

The **TabFixedHeight** and **TabFixedWidth** properties syntax has these parts:

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TabStrip** control. |
| *integer* | The number of pixels or twips of the height or width of a **TabStrip** control. The scale used for *integer* is dependent on the **ScaleMode** of the container. |

**Remarks**

The **TabFixedHeight** property applies to all **Tab** objects in the **TabStrip** control. It defaults either to the height of the font as specified in the **Font** property, or the height of the **ListImage** object specified by the **Image** property, whichever is higher, plus a few extra pixels as a border. If the **TabWidthStyle** property is set to **tabFixed**, and the value of the **TabFixedWidth** property is set, the width of each **Tab** object remains the same whether you add or delete **Tab** objects in the control.

# Visual Basic: MSTab Control

**Visual Studio 6.0**

# TabHeight Property (SSTab Control)

See Also Example Applies To

Returns or sets the height of all tabs on an **SSTab** control.

**Syntax**

*object*.**TabHeight** [ = *height* ]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *height* | A numeric expression that specifies the height of the tab, based on the scale mode of its container. |

# Visual Basic Reference

**Visual Studio 6.0**

# TabIndex Property

See Also Example Applies To

Returns or sets the tab order of most objects within their parent form.

**Syntax**

*object*.**TabIndex** [= *index*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *index* | An integer from 0 to (*n*1), where *n* is the number of controls on the form that have a **TabIndex** property. Assigning a **TabIndex** value of less than 0 generates an error. |

**Remarks**

By default, Visual Basic assigns a tab order to controls as you draw them on a form, with the exception of the **Menu**, **Timer**, **Data**, **Image**, **Line** and **Shape** controls, which are not included in the tab order. At run time, invisible or disabled controls and controls that can't receive the focus (**Frame** and **Label** controls) remain in the tab order but are skipped during tabbing.

Each new control is placed last in the tab order. If you change the value of a control's **TabIndex** property to adjust the default tab order, Visual Basic automatically renumbers the **TabIndex** of other controls to reflect insertions and deletions. You can make changes at design time using the Properties window or at run time in code

The **TabIndex** property isn't affected by the **ZOrder** method.

**Note** A control's tab order doesn't affect its associated access key. If you press the access key for a **Frame** or **Label** control, the focus moves to the next control in the tab order that can receive the focus.

When loading forms saved as ASCII text, controls with a **TabIndex** property that aren't listed in the form description are automatically assigned a **TabIndex** value. In subsequently loaded controls, if existing **TabIndex** values conflict with earlier assigned values, the controls are automatically assigned new values.

When you delete one or more controls, you can use the **Undo** command to restore the controls and all their properties except for the **TabIndex** property, which can't be restored. **TabIndex** is reset to the end of the tab order when you use Undo.

# Visual Basic Reference

# TabIndex Property Example

This example reverses the tab order of a group of buttons by changing the **TabIndex** property of a command button array. To try this example, paste the code into the Declarations section of a form that contains four **CommandButton** controls. Set the **Name** property to CommandX for each button to create the control array, and then press F5 and click the form to reverse the tab order of the buttons.

```
Private Sub Form_Click ()
   Dim I, X   ' Declare variables.
   ' Reverse tab order by setting start value of X.
   If CommandX(0).TabIndex = 0 Then X = 4 Else X = 1
      For I = 0 To 3
         CommandX(I).Caption = X   ' Set caption.
         CommandX(I).TabIndex = X - 1   ' Set tab order.
         If CommandX(0).TabIndex = 3 Then
            X = X - 1   ' Decrement X.
         Else
            X = X + 1   ' Increment X.
         End If
      Next I
End Sub
```

# Visual Basic: MSTab Control

**Visual Studio 6.0**

# TabMaxWidth Property (SSTab Control)

See Also Example Applies To

Returns or sets the maximum width of each tab on an **SSTab** control.

**Syntax**

*object*.**TabMaxWidth** [ = *width* ]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *width* | A numeric expression that determines the maximum width of each tab in the scale mode of its container. |

**Remarks**

When the **Style** property setting is **ssStyleTabbedDialog** and the **TabMaxWidth** property is set to zero (0), the **SSTab** control automatically sizes the tabs, based on the **TabsPerRow** property, to fit evenly across the control.

If you select the **ssStylePropertyPage** setting in the **Style** property, the **TabMaxWidth** property is ignored. The width of each tab adjusts automatically to the length of the text in the **TabCaption** property.

# Visual Basic: Windows Controls

**Visual Studio 6.0**

# TabMinWidth Property

See Also Example Applies To

Returns or sets the minimum allowable width of a tab.

**Syntax**

*object*.**TabMinWidth** [= *number*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *number* | The minimum width of a **Tab** object. The scale used for *number* is determined by the **ScaleMode** property of the container. |

**Remarks**

The **TabMinWidth** property has no effect if the **TabWidthStyle** property is set to **tabFixed**.

# Visual Basic: MSTab Control

**Visual Studio 6.0**

# TabOrientation Property (SSTab Control)

See Also Example Applies To

Returns or sets the location of the tabs on the **SSTab** control.

**Syntax**

*object*.**TabOrientation** [ = *number* ]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *number* | A numeric expression that specifies the location of the tabs, as described in Settings. |

**Settings**

The settings for *number* are:

| Constant | Value | Description |
|---|---|---|
| **ssTabOrientationTop** | **0** | The tabs appear at the top of the control. |
| **ssTabOrientationBottom** | **1** | The tabs appear at the bottom of the control. |
| **ssTabOrientationLeft** | **2** | The tabs appear on the left side of the control. |
| **ssTabOrientationRight** | **3** | The tabs appear on the right side of the control. |

**Remarks**

If you are using TrueType fonts, the text is rotated when the **TabOrientation** property is set to **ssTabOrientationLeft** or **ssTabOrientationRight**.

# Visual Basic: MSTab Control

**Visual Studio 6.0**

# TabPicture Property (SSTab Control)

See Also Example Applies To

Returns or sets the bitmap or icon to display on the specified tab of an **SSTab** control.

### Syntax

*object*.**TabPicture**(*tab*) [ = *picture* ]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tab* | A numeric expression that specifies the tab on which to display the picture. |
| *picture* | A string expression that specifies a graphic, as described in Settings. |

### Settings

The settings for *picture* are:

| Setting | Description |
|---|---|
| (None) | (Default) No picture. |
| (Bitmap, icon, metafile) | Specifies a graphic. At run time, you can set this property using the **LoadPicture** function or the **Picture** property of another control or **Form** object. |

### Remarks

At design time, you can set the **TabPicture** property by clicking a tab then setting the **Picture** property in the Properties window. Or you can select (Custom) in the Properties window and set the **Picture** property in the Pictures tab of the Properties dialog box.

At run time, you can refer to or change the graphic on any tab using the **TabPicture** property or use the **Picture** property to work with the active tab.

# Visual Basic: MSTab Control

**Visual Studio 6.0**

# Tabs Property (SSTab Control)

See Also Example Applies To

Returns or sets the total number of tabs on an **SSTab** control.

**Syntax**

*object*.**Tabs** [ = *tabnumber* ]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tabnumber* | A numeric expression that specifies the number of tabs you want on the control. The tabs are automatically given the captions Tab x where x is 0, 1, 2, 3, and so on. |

**Remarks**

You can change the **Tabs** property at run time to add new tabs or remove tabs.

At design time, use the **Tabs** property in conjunction with the **TabsPerRow** property to determine the number of rows of tabs displayed by the control. At run time, use the **Rows** property.

# Visual Basic: Windows Controls

**Visual Studio 6.0**

# Tabs Property (TabStrip Control)

See Also Example Applies To

Returns a reference to the collection of **Tab** objects in a **TabStrip** control.

**Syntax**

*object*.**Tabs(***index***)**

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TabStrip** control. |
| *index* | A value that identifies a **Tab** object in the **Tabs** collection. This may either be the **Index** property or the **Key** property of the desired **Tab** object. |

**Remarks**

The **Tabs** collection can be accessed by using the standard collection methods, such as the **Item** method.

# Visual Basic: MSTab Control

**Visual Studio 6.0**

# TabsPerRow Property (SSTab Control)

See Also Example Applies To

Returns or sets the number of tabs for each row of an **SSTab** control.

**Syntax**

*object*.**TabsPerRow** [ = *tabnumber* ]

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tabnumber* | A numeric expression that specifies the number of tabs you want on each row. |

**Remarks**

Use this property at design time in conjunction with the **Tabs** property to determine the number of rows displayed by the control. At run time, use the **Rows** property.

# Visual Basic Reference

**Visual Studio 6.0**

# TabStop Property

See Also Example Applies To

Returns or sets a value indicating whether a user can use the TAB key to give the focus to an object.

### Syntax

*object*.**TabStop** [= *boolean*]

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

| Part | Description |
|---|---|
| *Object* | An object expression that evaluates to an object in the Applies To list. |
| *Boolean* | A Boolean expression specifying whether the object is a tab stop, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) Designates the object as a tab stop. |
| **False** | Bypasses the object when the user is tabbing, although the object still holds its place in the actual tab order, as determined by the **TabIndex** property. |

### Remarks

This property enables you to add or remove a control from the tab order on a form. For example, if you're using a **PictureBox** control to draw a graphic, set its **TabStop** property to **False**, so the user can't tab to the **PictureBox**.

# Visual Basic: Windows Controls

**Visual Studio 6.0**

# TabStyle Property

See Also Example Applies To

Returns or sets a value that determines how remaining rows of tabs in front of a selected tab are repositioned.

**Syntax**

*object*.**TabStyle** [= *integer*]

The **TabStyle** 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 specifying the how the remaining tabs will be repositioned, as shown in Settings. |

**Settings**

The settings for *integer* are:

| Constant | Value | Description |
|---|---|---|
| **tabTabStandard** | 0 | (Default) The remaining tabs remain on the same side of the control. |
| **tabTabOpposite** | 1 | The row of tabs in front of the selected tab are repositioned at the opposite side of the control. |

# Visual Basic: MSTab Control

**Visual Studio 6.0**

# TabVisible Property (SSTab Control)

See Also Example Applies To

Returns or sets a value indicating if a tab in an **SSTab** control is visible or hidden. Not available at design time.

### Syntax

*object*.**TabVisible**(*tab*) [ = *boolean* ]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an **SSTab** control. |
| *tab* | A numeric expression that specifies the tab you want to be visible or hidden. |
| *boolean* | A Boolean expression that specifies if the tab is visible or hidden, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) Tab is visible. |
| **False** | Tab is hidden. Other tabs adjust their position so there are no gaps between tabs. |

### Remarks

The **TabVisible** property hides or displays a single tab. Use the **Visible** property to hide or display the entire **SSTab** control.

# Visual Basic: Windows Controls

**Visual Studio 6.0**

# TabWidthStyle Property

See Also Example Applies To

Returns or sets a value that determines the justification or width of all **Tab** objects in a **TabStrip** control.

### Syntax

*object*.**TabWidthStyle** [=*value*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **TabStrip** control. |
| *value* | An integer or constant that determines whether tabs are justified or set to a fixed width, as described in Settings. |

### Settings

The settings for *value* are:

| Constant | Value | Description |
|---|---|---|
| **tabJustified** | 0 | (Default) Justified. If the **MultiRow** property is set to **True**, each tab is wide enough to accommodate its contents and, if needed, the width of each tab is increased so that each row of tabs spans the width of the control. If the **MultiRow** property is set to **False**, or if there is only a single row of tabs, this setting has no effect. |
| **tabNonJustified** | 1 | Nonjustified. Each tab is just wide enough to accommodate its contents. The rows are not justified, so multiple rows of tabs are jagged. |
| **tabFixed** | 2 | Fixed. All tabs have an identical width, which is determined by the **TabFixedWidth** property. |

### Remarks

At design time you can set the **TabWidthStyle** property on the General tab of the Properties Page of the **TabStrip** control. The setting of the **TabWidthStyle** property affects how wide each **Tab** object appears at run time.

At run time, you can set the **TabWidthStyle** property as follows:

```
' Justifies all the tabs in a row to fit the width of the control.
TabStrip1.MultiRow = True
TabStrip1.TabWidthStyle = tabJustified

' Creates ragged rows of tabs.
TabStrip1.MultiRow = True
TabStrip1.TabWidthStyle = tabNonJustified

' Sets the same width for all tabs.
TabStrip1.TabFixedWidth = 500
TabStrip1.TabWidthStyle = tabFixed
```

# Visual Basic Reference

**Visual Studio 6.0**

# Tag Property (ActiveX Controls)

See Also Example Applies To

Returns or sets an expression that stores any extra data needed for your program. Unlike other properties, the value of the **Tag** property isn't used by Visual Basic; you can use this property to identify objects.

**Syntax**

*object*.**Tag** [= *expression*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *expression* | A string expression identifying the object. The default is a zero-length string (""). |

**Remarks**

You can use this property to assign an identification string to an object without affecting any of its other property settings or causing side effects. The **Tag** property is useful when you need to check the identity of a control or **MDIForm** object that is passed as a variable to a procedure.

**Tip** When you create a new instance of a form, assign a unique value to the **Tag** property.

**Note** The **Tag** property is of type Variant for ActiveX control collections such as **Toolbar Button** objects, **TreeView Node** objects, **ListView ListItem** and **ColumnHeader** objects, **ImageList ListImage** objects, **TabStrip Tab** objects, and **StatusBar Panel** objects. You can use the **Tag** property to pass values, but it does not allow you to pass objects.

# Visual Basic Reference

# Tag Property Example

This example displays a unique icon for each control being dragged. To try this example, paste the code into the Declarations section of a form that contains three **PictureBox** controls. Set the **DragMode** property to 1 for Picture1 and Picture2, and then press F5. Use the mouse to drag Picture1 or Picture2 over Picture3 controls.

```
Private Sub Form_Load ()
  Picture1.Tag = "ICONS\ARROWS\POINT03.ICO"
  Picture2.Tag = "ICONS\ARROWS\POINT04.ICO"
End Sub

Private Sub Picture3_DragOver (Source As Control, X As Single, Y As Single, State As Integer)
   If State = vbEnter Then
      ' Select based on each PictureBoxs Name property.
      Select Case Source.Name
      Case "Picture1"
         ' Load icon for Picture1.
         Source.DragIcon = LoadPicture(Picture1.Tag)        Case "Picture2"
         ' Load icon for Picture2.
         Source.DragIcon = LoadPicture(Picture2.Tag)
      End Select
   ElseIf State = vbLeave Then
      ' When source isn't over Picture3, unload icon.
      Source.DragIcon = LoadPicture ()
   End If
End Sub
```

# Visual Basic Reference

**Visual Studio 6.0**

# Tag Property

See Also Example Applies To

Returns or sets an expression that stores any extra data needed for your program. Unlike other properties, the value of the **Tag** property isn't used by Visual Basic; you can use this property to identify objects.

### Syntax

*object*.**Tag** [= *expression*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *expression* | A string expression identifying the object. The default is a zero-length string (""). |

### Remarks

You can use this property to assign an identification string to an object without affecting any of its other property settings or causing side effects. The **Tag** property is useful when you need to check the identity of a control or **MDIForm** object that is passed as a variable to a procedure.

**Tip** When you create a new instance of a form, assign a unique value to the **Tag** property.

# Visual Basic Reference

# Tag Property Example

This example displays a unique icon for each control being dragged. To try this example, paste the code into the Declarations section of a form that contains three **PictureBox** controls. Set the **DragMode** property to 1 for Picture1 and Picture2, and then press F5. Use the mouse to drag Picture1 or Picture2 over Picture3 controls.

```
Private Sub Form_Load ()
  Picture1.Tag = "ICONS\ARROWS\POINT03.ICO"
  Picture2.Tag = "ICONS\ARROWS\POINT04.ICO"
End Sub

Private Sub Picture3_DragOver (Source As Control, X As Single, Y As Single, State As Integer)
   If State = vbEnter Then
      ' Select based on each PictureBoxs Name property.
      Select Case Source.Name
      Case "Picture1"
         ' Load icon for Picture1.
         Source.DragIcon = LoadPicture(Picture1.Tag)        Case "Picture2"
         ' Load icon for Picture2.
         Source.DragIcon = LoadPicture(Picture2.Tag)
      End Select
   ElseIf State = vbLeave Then
      ' When source isn't over Picture3, unload icon.
      Source.DragIcon = LoadPicture ()
   End If
End Sub
```

# Visual Basic Reference

**Visual Studio 6.0**

# TagPrefix Property

See Also Example Applies To

Sets or returns the string used to prefix replacement tagnames in an HTML template. The first character must be alphabetic.

**Syntax**

*object*.**TagPrefix[**=*string***]**

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *string* | Identifies the TagPrefix sequence used to recognize replacement tags within an HTML template. |

**Remarks**

**TagPrefix** can be set at design time or at run time. The **WebClass** run time will look for any tag in a template file that begins with the specified prefix. When it finds it, it will fire the ProcessTag event on the **WebClass** object.

# Visual Basic Reference

**Visual Studio 6.0**

# Target Property

See Also Example Applies To

Returns the **Target** parameter that was passed to **AsyncRead**.

**Syntax**

*object*.**Target**

The **Target** property syntax has this part:

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

**Remarks**

The **Target** property returns a string that is the same value as the first parameter passed to the **AsyncRead** method, usually a URL.

# Visual Basic Reference

**Visual Studio 6.0**

# TargetObject Property

See Also Example Applies To

Returns the bound object, or another object as determined by the bound control. Read-only at run time.

**Syntax**

*object*.**TargetObject**

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

**Remarks**

The **TargetObject** property provides access in the Format and Unformat events to the bound control's properties, methods, and events. If the bound control passes an object other than the control itself, you can use its properties, methods, and events to manipulate data or navigate up the hierarchy to other objects in the control's object model.

# Visual Basic Reference

**Visual Studio 6.0**

# TaskVisible Property

See Also Example Applies To

Returns or sets a value that determines if the application appears in the Windows task list.

**Syntax**

*object*.**TaskVisible** [= *boolean*]

The **TaskVisible** 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 determines if the application appears in the task list, as described in Settings. |

**Settings**

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) The application appears in the Windows task list. |
| **False** | The application does not appear in the Windows task list. |

**Remarks**

The **TaskVisible** property can only be set to **False** in applications that do not display an interface, such as ActiveX components that do not contain or display **Form** objects. While the application displays an interface, the **TaskVisible** property is automatically set to **True**.

# Visual Basic Reference

**Visual Studio 6.0**

# Template Property

See Also Example Applies To

Returns or sets the HTML template to use in exporting a report.

**Syntax**

*object*.**Template** [=*string*]

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

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to an object in the Applies To list. |
| *string* | Optional. The header and/or footer code to be used. |

**Remarks**

If no template is specified when invoking the **ExportReport** method, a default template appropriate to the report type will be used. Four default types are provided:

- HTML template
- UTF-8 encoded HTML
- Text
- Unicode text

You can print the default templates for use as a starting point in creating your own template by using the code below:

```
Dim i As Integer
For i = 1 To rptNwind.ExportFormats.Count
   Debug.Print "Template " & i
   Debug.Print rptNwind.ExportFormats(i).Template
   Debug.Print
Next i
```

You can create your own templates using HTML tags such as <HTML>, <HEAD>, <TITLE>, <BODY> and their requisite closing tags. To specify where the body of the report should be placed, use the Visual Basic constant **rptTagBody**. For example, the code fragment below places the body of the report between the <BODY> tags:

```
Dim strTemplateBody As String
strTemplateBody = _
"<BODY>" & vbCrLf & _
rptTagBody & vbCrLf & _
"</BODY>"
```

Similarly, you can use the constant **rptTagTitle** to place the title (as determined by the **Title** property) anywhere in the template. The fragment below places the title just above the body:

```
Dim strTemplateBodyAndTitle As String
strTemplateBodyAndTitle = _
rptTagTitle & vbCrLf & _
"<BODY>" & vbCrLf & _
rptTagBody & vbCrLf & _
"</BODY>"
```

Accordingly, the default template for Text export consists simply of the **rptTagBody** constant. Thus you could create a simple text template that prefaces the report with the author's name, followed by the **rptTagBody** constant, as shown in the example below:

```
Dim strTemplate As String
strTemplate = _
"Author: John Smith" & vbCrLf & _
rptTagBody
```

# Visual Basic Reference

# Add Method, ExportFormats Collection, Template Property Example

This example creates a template, adds an **ExportFormat** object to the **ExportFormats** collection using the new template, and exports the report using the **ExportFormat** object.

```
Private Sub ExportDailyReport()
   DataReport1.Title = "Daily Report" ' This title appears in the report.
   Dim strTemplate As String
   ' Create the template.
   strTemplate = _
   "<HTML>" & vbCrLf & _
   "<HEAD>" & vbCrLf & _
   "<TITLE>" & "MyCompany: " & rptTagTitle & _
   "</TITLE>" & vbCrLf & _
   "<BODY>" & vbCrLf & _
   rptTagBody & vbCrLf & _
   "<BODY>" & vbCrLf & _
   "</HTML>"

   ' Add a new ExportFormat object using the template.
   DataReport1.ExportFormats.Add _
   Key:="DailyReport", _
   FormatType:=rptFmtHTML, _
   FileFormatString:="Daily Report (*.htm)", _
   FileFilter:="*.HTM", _
   Template:=strTemplate

   ' Export the report using the new ExportFormat object.
   DataReport1.ExportReport _
   FormatIndexOrKey:="DailyReport", _
   FileName:="C:\Temp\DailyRpt", _
   Overwrite:=True, _
   ShowDialog:=False, _
   Range:=rptRangeFromTo, _
   Pagefrom:=1, _
   Pageto:=10
End Sub
```

# Visual Basic Extensibility Reference

**Visual Studio 6.0**

# TemplatePath Property

See Also Example Applies To

Returns the full pathname where Visual Basic stores template files.

**Syntax**

*object*.**TemplatePath**

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

# Visual Basic Reference

**Visual Studio 6.0**

# Text Property (ActiveX Controls)

See Also Example Applies To

Returns or sets the text contained in an object.

**Syntax**

*object*.**Text** [= *string*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *string* | A string expression specifying the text appearing in the object. |

# Visual Basic Reference

# Text Property (ActiveX Controls) Example

This example populates a **TreeView** control with the titles of files in a **ListBox** control. When an item in the **TreeView** control is clicked, the **Text** property is displayed in a **Label** on the form. To try the example, place **TreeView**, **Label**, and **ListBox** controls on a form and paste the code into the form's Declarations section. Run the example and click on any item to see its **Text** property.

```
Private Sub Form_Load()
   Dim nodX As Node  ' Declare an object variable for the Node.
   Dim i As Integer  ' Declare a variable for use as a counter.

   ' Add one Node to the TreeView control, and call it the first node
   Set nodX = TreeView1.Nodes.Add()
   nodX.Text = "First Node"

   'Populate the ListBox
      List1.AddItem "Node1"   ' Add each item to list.
      List1.AddItem "Node2"
      List1.AddItem "Node3"
      List1.AddItem "Node4"
      List1.AddItem "Node5"
      List1.AddItem "Node6"
      List1.AddItem "Node7"

   ' Add child nodes to the first Node object. Use the
   ' ListBox to populate the control.
   For i = 0 To List1.ListCount - 1
      Set nodX = TreeView1.Nodes.Add(1, tvwChild)
      nodX.Text = List1.List(i)
   Next i
   Treeview1.Nodes(1).Selected = True
   nodX.EnsureVisible   ' Make sure the node is visible.
End Sub

Private Sub TreeView1_NodeClick(ByVal Node As Node)
   ' Display the clicked Node object's Text property.
   Label1.Caption = Node.Text
End Sub
```

# Visual Basic: MSFlexGrid/MSHFlexGrid Controls

**Visual Studio 6.0**

# Text Property (MSHFlexGrid)

See Also Example Applies To

Returns or sets the text content of a cell or range of cells.

**Syntax**

*object*.**Text** [=*string*]

The **Text** 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 containing the text from a cell or range of cells. |

**Remarks**

When retrieving, the **Text** property always retrieves the contents of the current cell as defined by the **Row** and **Col** properties.

When setting, the **Text** property sets the contents of the current cell or selection depending on the setting of the **FillStyle** property.

**Visual Studio 6.0**

*Visual Basic: MSChart Control*

# Text Property (MSChart)

See Also Example Applies To

Returns or sets the text used to display a chart element such as an axis title, data point label, footnote, or chart title.

**Syntax**

*object*.**Text** [ = *text* ]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *text* | String. A string that contains the text used for the chart element. |

**Remarks**

The **Text** property is the default property for each of the objects to which it applies.

# Visual Basic Reference

**Visual Studio 6.0**

# Text Property

See Also Example Applies To

- **ComboBox** control (**Style** property set to 0 [Dropdown Combo] or to 1 [Simple Combo]) and **TextBox** control returns or sets the text contained in the edit area.
- **ComboBox** control (**Style** property set to 2 [Dropdown List]) and **ListBox** control returns the selected item in the list box; the value returned is always equivalent to the value returned by the expression `List(ListIndex)`. Read-only at design time; read-only at run time.

**Syntax**

*object*.**Text** [= *string*]

The **Text** 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 text. |

**Remarks**

At design time only, the defaults for the **Text** property are:

- **ComboBox** and **TextBox** controls the control's **Name** property.
- **ListBox** control a zero-length string ("").

For a **ComboBox** with the **Style** property set to 0 (Dropdown Combo) or to 1 (Simple Combo) or for a **TextBox**, this property is useful for reading the actual string contained in the edit area of the control. For a **ComboBox** or **ListBox** control with the **Style** property set to 2 (Dropdown List), you can use the **Text** property to determine the currently selected item.

The **Text** setting for a **TextBox** control is limited to 2048 characters unless the **MultiLine** property is **True**, in which case the limit is about 32K.

# Visual Basic Reference

# Text Property Example

This example illustrates the **Text** property. To try this example, paste the code into the Declarations section of a form that contains three **TextBox** controls and a **CommandButton** control, and then press F5 and enter text in Text1.

```
Private Sub Text1_Change ()
   Text2.Text = LCase(Text1.Text)   ' Display text as lowercase.
   Text3.Text = UCase(Text1.Text)   ' Display text as uppercase.
End Sub

Private Sub Command1_Click ()   ' Delete text.
   Text1.Text = ""
End Sub
```

# Visual Basic Reference

**Visual Studio 6.0**

# TextAlign Property

See Also Example Applies To

Returns an enumerated value of type TextAlignChoices stating what kind of text alignment the container would like the control to do.

**Syntax**

*object*.**TextAlign**

The **TextAlign** property syntax has this part:

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

**Settings**

The possible enumerated return values from the **TextAlign** property are:

| Setting | Description |
|---|---|
| **0-General** | General alignment: text to the left, numbers to the right. If the container does not implement this ambient property, this will be the default value. |
| **1-Left** | Align to the left. |
| **2-Center** | Align in the center. |
| **3-Right** | Align to the right. |
| **4-FillJustify** | Fill justify. |

**Remarks**

This ambient property is the way that a container communicates to a contained control how to perform justification; this is a hint from the container that the control may or may not choose to follow.

# Visual Basic: Windows Controls

**Visual Studio 6.0**

# TextAlignment Property

See Also Example Applies To

Returns or sets a value that determines the position of text relative to the button.

**Syntax**

*object*.**TextAlignment** [= *integer*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *integer* | The position of the text, as shown in Settings. |

**Settings**

The settings for *integer* are:

| Constant | Value | Description |
|---|---|---|
| **tbrTextAlignBottom** | 0 | Text is aligned at the bottom of the button. |
| **tbrTextAlignRight** | 1 | Text is aligned to the right of the button. |

# Visual Basic: MSFlexGrid/MSHFlexGrid Controls

**Visual Studio 6.0**

# TextArray Property

See Also Example Applies To

Returns or sets the text content of an arbitrary cell.

**Syntax**

*object*.**TextArray(***cellindex***)** [=*string*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *cellindex* | Integer. A numeric expression that specifies which cell to read or write. See Remarks. |
| *string* | A string expression containing the contents of an arbitrary cell. |

**Remarks**

This property allows you to set or retrieve the contents of a cell without changing the **Row** and **Col** properties.

The *cellindex* argument determines which cell to use. It is calculated by multiplying the preferred row by the **Cols** property and adding the preferred column. The clearest and most convenient way to calculate *cellindex* is to define a function to do it, as shown in Example.

# Visual Basic: MSFlexGrid/MSHFlexGrid Controls

# TextArray Property Example

The following example shows how to calculate *cellindex* by defining a function.

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

```
' Calculate index for use with TextArray property.
Function faIndex(row As Integer, col As Integer) As Long
faIndex =row * MSHFlexGrid1.Cols + col
End Function
Sub Form_Load()
Dim i as Integer
' Fill MSHFlexGrid with data using TextArray property.
For i =MSHFlexGrid1.FixedRows to MSFlexGrid1.Rows - 1
' ** column 1
MSHFlexGrid1.TextArray(faIndex(i, 1)) =RandomName()
' Column 2.
MSHFlexGrid1.TextArray(faIndex(i, 2)) =RandomNumber()
Next
```

# Visual Basic: Windows Controls

**Visual Studio 6.0**

# TextBackground Property

See Also Example Applies To

Returns or sets a value that determines if a **ListItem** object's text background is opaque or transparent.

**Syntax**

*object*.**TextBackground** [ = *integer*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *integer* | A constant or value that determines the style of the text background, as shown in Settings. |

**Settings**

| Constant | Value | Description |
|---|---|---|
| **lvwTransparent** | 0 | The text background is transparent. |
| **lvwOpaque** | 1 | The text background is the same color as the **BackColor** property. |

**Remarks**

The text background is the rectangular field which surrounds the **ListItem** object's text.

The **TextBackground** property is used when the **ListView** control displays a picture (by assigning a picture to the **Picture** property). When a **ListItem** object is positioned over the picture and the property is set to opaque, the picture will not show through the text background. If the property is set to transparent, the background picture will be visible behind the text.

**Visual Studio 6.0**

*Visual Basic: MSChart Control*

# TextLayout Property

See Also Example Applies To

Returns a reference to a **TextLayout** object that describes text positioning and orientation.

**Syntax**

*object*.**TextLayout**

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*

# TextLength Property

See Also Example Applies To

Returns or sets the number of characters in the text of a chart axis title, data point label, footnote, or chart title.

**Syntax**

*object*.**TextLength** [ = *size*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *size* | Integer. The number of characters in the text. |

**Visual Studio 6.0**

*Visual Basic: MSChart Control*

# TextLengthType Property

See Also Example Applies To

Returns or sets a value that specifies how text is drawn to optimize the appearance either on the screen or printed page.

**Syntax**

*object*.**TextLengthType** [ = *type*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *type* | Integer. A VtTextLengthType constant indicating the method used to draw text. |

# Visual Basic: MSFlexGrid/MSHFlexGrid Controls

**Visual Studio 6.0**

# TextMatrix Property

See Also Example Applies To

Returns or sets the text contents of an arbitrary cell.

**Syntax**

*object*.**TextMatrix(***rowindex, colindex***)** [=*string*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *rowindex, colindex* | Integer. A numeric expression that specifies which cell to read or write. |
| *string* | A string expression containing the contents of an arbitrary cell. |

**Remarks**

This property allows you to set or retrieve the contents of a cell without changing the **Row** and **Col** properties.

# Visual Basic: MSFlexGrid/MSHFlexGrid Controls

# Sort, TextMatrix Properties (MSHFlexGrid) Example

The following example uses the **Sort** and **TextMatrix** properties. It performs an **MSHFlexGrid** sort according to the value of a **ComboBox** control. To use the example, place an **MSHFlexGrid** control and a **ComboBox** control on a form. Paste the following code into the Declarations section, and then press F5.

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

```
Private Sub Combo1_Click()
' Select Column according to Sort method.
Select Case Combo1.ListIndex
Case 0 To 2
MSHFlexGrid1.Col =1
Case 3 To 4
MSHFlexGrid1.Col =2
Case 4 To 8
MSHFlexGrid1.Col =1
End Select
' Sort according to Combo1.ListIndex.
MSHFlexGrid1.Sort =Combo1.ListIndex
End Sub
Private Sub Form_Load()
Dim i As Integer
' Fill MSHFlexGrid with random data.
MSHFlexGrid1.Cols =3 ' Create three columns.

For i =1 To 11 ' Add ten items.
MSHFlexGrid1.AddItem ""
MSHFlexGrid1.Col =2
MSHFlexGrid1.TextMatrix(i, 1) =SomeName(i)
MSHFlexGrid1.TextMatrix(i, 2) =Rnd()
Next i
' Fill combo box with Sort choices
With Combo1
.AddItem "flexSortNone" ' 0
.AddItem "flexSortGenericAscending" '1
.AddItem "flexSortGenericDescending" '2
.AddItem "flexSortNumericAscending" '3
.AddItem "flexSortNumericDescending" '4
.AddItem "flexSortStringNoCaseAsending" '5
.AddItem "flexSortNoCaseDescending" '6
.AddItem "flexSortStringAscending" '7
.AddItem "flexSortStringDescending" '8
.ListIndex =0
End With
End Sub
Private Function SomeName(i As Integer) As String
Select Case i
```

```
Case 1
SomeName ="Ann"
Case 2
SomeName ="Glenn"
Case 3
SomeName ="Sid"
Case 4
SomeName ="Anton"
Case 5
SomeName ="Hoagie"
Case 6
SomeName ="Traut 'Trane"
Case 7
SomeName ="MereD Wah"
Case 8
SomeName ="Kemp"
Case 9
SomeName ="Sandy"
Case 10
SomeName ="Lien"
Case 11
SomeName ="Randy"
End Select
End Function
```

# Visual Basic: Windows Controls

**Visual Studio 6.0**

# TextPosition Property

See Also Example Applies To

Returns or sets a value that determines the position of displayed text, in relation to the object.

**Syntax**

*object*.**TextPosition** [= *integer*]

The **TextPosition** 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 position of **ToolTipText**, as shown in Settings. |

**Settings**

The settings for *integer* are:

| Constant | Value | Description |
|---|---|---|
| **sldAboveLeft** | 0 | Text is displayed above or to the left of the control. |
| **sldBelowRight** | 1 | Text is displayed below or to the right of the control. |

**Remarks**

Above and Below are used for horizontal sliders only. Left and Right are used for vertical sliders only.

# Visual Basic: RichTextBox Control

**Visual Studio 6.0**

# TextRTF Property

See Also Example Applies To

Returns or sets the text of a **RichTextBox** control, including all .rtf code.

**Syntax**

*object*.**TextRTF** [= *string*]

The **TextRTF** 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 in .rtf format. |

**Remarks**

Setting the **TextRTF** property replaces the entire contents of a **RichTextBox** control with the new string.

You can use the **TextRTF** property along with the **Print** function to write .rtf files. The resulting file can be read by any other word processor capable of reading RTF-encoded text.

# Visual Basic: RichTextBox Control

# TextRTF Property Example

This example saves the entire contents of a **RichTextBox** control to an .rtf file. To try this example, put a **RichTextBox** control and a **CommandButton** control on a form. Paste this code into the Click event of the **CommandButton** control. Then run the example.

```
Private Sub Command1_Click ()
   Open "mytext.rtf" For Output As 1
   Print #1, RichTextBox1.TextRTF
   Close 1

End Sub
```

# Visual Basic: MSFlexGrid/MSHFlexGrid Controls

**Visual Studio 6.0**

# TextStyle, TextStyleBand, TextStyleFixed, TextStyleHeader Properties (MSHFlexGrid)

See Also Example Applies To

Returns or sets the three-dimensional style for text within a specific cell or range of cells.

- **TextStyle** determines the style of regular **MSHFlexGrid** cells.
- **TextStyleBand** determines the style of bands.
- **TextStyleFixed** determines the style of fixed rows and columns.
- **TextStyleHeader** determines the style of headers.

**Syntax**

*object*.**TextStyle** [=*style*]
*object*.**TextStyleBand(***BandNumber***)** [=*style*]
*object*.**TextStyleFixed** [=*style*]
*object*.**TextStyleHeader(***BandNumber***)** [=*style*]

Syntax for the **TextStyle**, **TextStyleBand**, **TextStyleFixed**, and **TextStyleHeader** properties 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 being affected. |
| *style* | An integer or constant that specifies the text style, as described in Settings. |

**Settings**

The settings for *style* are:

| Constant | Value | Description |
|---|---|---|
| | | |

| | | |
|---|---|---|
| **flexTextFlat** | 0 | The text is normal, flat text. This is the default. |
| **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.

# Visual Basic Reference

**Visual Studio 6.0**

# ThreadID Property

See Also Example Applies To

Returns the Win32 ID of the executing thread. (Used for Win32 API calls.)

**Syntax**

*object*.**ThreadID**

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

**Return Type**

Long

**Visual Studio 6.0**

*Visual Basic: MSChart Control*

# Tick Property

See Also Example Applies To

Returns a reference to a **Tick** object that describes a marker indicating a division along a chart axis.

**Syntax**

*object*.**Tick**

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**

# TickFrequency Property

See Also Example Applies To

Returns or sets the frequency of tick marks on a **Slider** control in relation to its range. For example, if the range is 100, and the **TickFrequency** property is set to 2, there will be one tick for every 2 increments in the range.

**Syntax**

*object*.**TickFrequency** [= *number*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Slider** control. |
| *number* | A numeric expression specifying the frequency of tick marks. |

# Visual Basic: Windows Controls

# TickFreqency Property Example

This example matches a **TextBox** control's width to that of a **Slider** control. While the **Slider** control's **Value** property is above a certain value, the **TextBox** control's width matches the **Slider** control's value. The **TickFrequency** depends on the value of the **Slider** control's **Max** property. To try the example, place a **Slider** and a **TextBox** control on a form and paste the code into the form's Declarations section. Run the example and click the slider several times.

```
Private Sub Form_Load()
   Text1.Width = 4500 ' Set a minimum width for the TextBox.
   Slider1.Left = Text1.Left ' Align the Slider to the TextBox.
   ' Match the width of the Slider to the TextBox.
   Slider1.Max = Text1.Width
   ' Place the Slider a little below the Textbox.
   Slider1.Top = Text1.Top + Text1.Height + 50
   ' Set TickFrequency to a fraction of the Max value.
   Slider1.TickFrequency = Slider1.Max * 0.1
   ' Set LargeChange and SmallChange value to a fraction of Max.
   Slider1.LargeChange = Slider1.Max * 0.1
   Slider1.SmallChange = Slider1.Max * 0.01
End Sub

Private Sub Slider1_Change()
   ' If the slider is under 1/3 the size of the textbox, no change.
   ' Else, match the width of the textbox to the Slider's value.
   If Slider1.Value > Slider1.Max / 3 Then
      Text1.Width = Slider1.Value
   End If
End Sub
```

# Visual Basic: Windows Controls

**Visual Studio 6.0**

# TickStyle Property

See Also Example Applies To

Returns or sets the style (or positioning) of the tick marks displayed on the **Slider** control.

**Syntax**

*object*.**TickStyle** [= *number*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to a **Slider** control. |
| *number* | A constant or integer that specifies the **TickStyle** property, as described in Settings. |

**Settings**

The settings for *number* are:

| Constant | Value | Description |
|---|---|---|
| **sldBottomRight** | 0 | (Default) Bottom/Right. Tick marks are positioned along the bottom of the **Slider** if the control is oriented horizontally, or along the right side if it is oriented vertically. |
| **sldTopLeft** | 1 | Top/Left. Tick marks are positioned along the top of the **Slider** if the control is oriented horizontally, or along the left side if it is oriented vertically. |
| **sldBoth** | 2 | Both. Tick marks are positioned on both sides or top and bottom of the **Slider**. |
| **sldNoTicks** | 3 | None. No tick marks appear on the **Slider**. |

# Visual Basic: Windows Controls

# TickStyle Property Example

This example allows you to see the various tick styles available in a drop-down list. To try the example, place a **Slider** control and a **ComboBox** control on a form. Paste the code into the Declarations section of the form, and run the example. Click on the **ComboBox** to change the **TickStyle** property value.

```
Private Sub Form_Load()
   With combo1
      .AddItem "Bottom/Right"
      .AddItem "Top/Left"
      .AddItem "Both"
      .AddItem "None"
      .ListIndex = 0
   End With
End Sub

Private Sub combo1_Click()
   Slider1.TickStyle = combo1.ListIndex
End Sub
```

# Visual Basic: Multimedia MCI Control

**Visual Studio 6.0**

# TimeFormat Property (Multimedia MCI Control)

See Also Example Applies To

Specifies the time format used to report all position information.

**Syntax**

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

**Remarks**

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

| Value | Setting/Time format |
|---|---|
| 0 | **mciFormatMilliseconds** |
| | Milliseconds are stored as a 4-byte integer variable. |
| 1 | **mciFormatHms** |
| | Hours, minutes, and seconds are packed into a 4-byte integer. From least significant byte to most significant byte, the individual data values are: |
| | Hours (least significant byte) |
| | Minutes |
| | Seconds |
| | Unused (most significant byte) |
| 2 | **mciFormatMsf** |
| | Minutes, seconds, and frames are packed into a 4-byte integer. From least significant byte to most significant byte, the individual data values are: |
| | Minutes (least significant byte) |
| | Seconds |

| | |
|---|---|
| | Frames |
| | Unused (most significant byte) |
| 3 | **mciFormatFrames** |
| | Frames are stored as a 4-byte integer variable. |
| 4 | **mciFormatSmpte24** |
| | 24-frame SMPTE packs the following values in a 4-byte variable from least significant byte to most significant byte: |
| | Hours (least significant byte) |
| | Minutes |
| | Seconds |
| | Frames (most significant byte) |
| | SMPTE (Society of Motion Picture and Television Engineers) time is an absolute time format expressed in hours, minutes, seconds, and frames. The standard SMPTE division types are 24, 25, and 30 frames per second. |
| 5 | **mciFormatSmpte25** |
| | 25-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE. |
| 6 | **mciFormatSmpte30** |
| | 30-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE. |
| 7 | **mciFormatSmpte30Drop** |
| | 30-drop-frame SMPTE packs data into the 4-byte variable in the same order as 24-frame SMPTE. |
| 8 | **mciFormatBytes** |
| | Bytes are stored as a 4-byte integer variable. |
| 9 | **mciFormatSamples** |
| | Samples are stored as a 4-byte integer variable. |
| 10 | **mciFormatTmsf** |
| | Tracks, minutes, seconds, and frame are packed in the 4-byte variable from least significant byte to most significant byte: |
| | Tracks (least significant byte) |
| | Minutes |
| | Seconds |
| | |

| | |
|---|---|
| | Frames (most significant byte) |
| | **Note** MCI uses continuous track numbering. |

**Note** Not all formats are supported by every device. If you try to set an invalid format, the assignment is ignored.

The current timing information is always passed in a 4-byte integer. In some formats, the timing information returned is not really an integer, but single bytes of information packed in the long integer. Properties that access or send information in the current time format are: **From**, **Length**, **Position**, **Start**, **To**, **TrackLength**, **TrackPosition**.

**Data Type**

Long (Enumerated)

# Visual Basic Reference

**Visual Studio 6.0**

# Title Property (DataReport Object)

See Also Example Applies To

Returns or sets the title of the report.

**Syntax**

*object*.**Title**[=*string*]

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

| Part | Description |
|---|---|
| *object* | Required. An object expression that evaluates to an object in the Applies To list. |
| *string* | Optional. The title of the report. |

**Remarks**

The title assigned with the **Title** property is used in two places:

1. In a **TextBox** control with the caption set to %i.
2. In an exported reportthe title is represented by the **rptTagTitle** constant when creating a template (using the **Template** property) of the **ExportFormat** object.

# Visual Basic Reference

# Add Method, ExportFormats Collection, Template Property Example

This example creates a template, adds an **ExportFormat** object to the **ExportFormats** collection using the new template, and exports the report using the **ExportFormat** object.

```
Private Sub ExportDailyReport()
   DataReport1.Title = "Daily Report" ' This title appears in the report.
   Dim strTemplate As String
   ' Create the template.
   strTemplate = _
   "<HTML>" & vbCrLf & _
   "<HEAD>" & vbCrLf & _
   "<TITLE>" & "MyCompany: " & rptTagTitle & _
   "</TITLE>" & vbCrLf & _
   "<BODY>" & vbCrLf & _
   rptTagBody & vbCrLf & _
   "<BODY>" & vbCrLf & _
   "</HTML>"

   ' Add a new ExportFormat object using the template.
   DataReport1.ExportFormats.Add _
   Key:="DailyReport", _
   FormatType:=rptFmtHTML, _
   FileFormatString:="Daily Report (*.htm)", _
   FileFilter:="*.HTM", _
   Template:=strTemplate

   ' Export the report using the new ExportFormat object.
   DataReport1.ExportReport _
   FormatIndexOrKey:="DailyReport", _
   FileName:="C:\Temp\DailyRpt", _
   Overwrite:=True, _
   ShowDialog:=False, _
   Range:=rptRangeFromTo, _
   Pagefrom:=1, _
   Pageto:=10
End Sub
```

# Visual Basic Reference

**Visual Studio 6.0**

# Title Property

See Also Example Applies To

Returns or sets the title of the application that is displayed in the Microsoft Windows Task List. If changed at run time, changes aren't saved with the application.

**Syntax**

*object*.**Title** [= *value*]

The **Title** 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 specifying the title of the application. The maximum length of *value* is 40 characters. In DBCS (double-byte character set) systems, this means the maximum length is 40 bytes. |

**Remarks**

This property is available at design time in the dialog box for the Project Properties command on the Project menu.

**Visual Studio 6.0**

*Visual Basic: MSChart Control*

# Title Property (MSChart)

See Also Example Applies To

Reference to a **Title** object that describes the text used to title a chart.

**Syntax**

*object*.**Title**

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**

# TitleBackColor, TitleForeColor Properties

See Also Example Applies To

Return or set values that specify the background and foreground colors of the title area of the control.

**Syntax**

*object*.**TitleBackColor** [= *color*]

*object*.**TitleForeColor** [= *color*]

The **TitleBackColor** and **TitleForeColor** properties' syntax 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**

Visual Basic uses the Microsoft Windows operating system red-green-blue (RGB) color scheme. The settings for *color* are:

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

**Remarks**

The **TitleBackColor and TitleForeColor** properties can be used with the **MonthBackColor** and **TrailingForeColor** 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 three 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

# TitleBackColor, TitleForeColor, TrailingForeColor Properties Example

The example changes the appearance of a **MonthView** control by resetting the **TitleBackColor**, **TitleForeColor**, and **TrailingForeColor** properties. To try the example, place a **MonthView** control on a form, and paste the code into the Declarations section of the code module. Run the project, and double-click the form to see the calendar change.

```
Private Sub Form_DblClick()
   With MonthView1
      .TitleBackColor = vbBlue
      .TitleForeColor = vbWhite
      .TrailingForeColor = vbRed
   End With
End Sub
```

# TitleText Property

See Also Example Applies To

Returns or sets the text displayed as the chart title.

**Syntax**

*object*.**TitleText** [ = *text*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *text* | The text used to display a chart title. |

**Remarks**

This property provides a simple means to set or return the chart title. This property is functionally identical to using `MSChart.Title.Text`.

# Visual Basic: Multimedia MCI Control

**Visual Studio 6.0**

# To Property (Multimedia MCI Control)

See Also Example Applies To

Specifies the ending point, as defined in the **Multimedia MCI** control **TimeFormat** property, for the **Play** or **Record** command. This property is not available at design time.

**Syntax**

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

**Remarks**

The argument *location&* specifies the ending point for the play or record operation. The current time format is given by the **TimeFormat** property.

The value assigned to this property is used only with the next **MCI** command. Subsequent **MCI** commands ignore the **To** property until it is assigned another (different or identical) value.

**Data Type**

Long

# Visual Basic Reference

**Visual Studio 6.0**

# ToolboxBitmap Property

See Also Example Applies To

Returns or sets a bitmap that will be used as the picture representation of the control in the toolbox. The size of the space for the bitmap in the toolbox is 16x15 pixels; the bitmap specified by this property will be scaled to these dimensions if necessary. The **ToolboxBitmap** property is read/write at the controls authoring time, and not available at the controls run time.

**Remarks**

**Important** Do not assign an icon to the **ToolboxBitmap** property. Icons do not scale well to Toolbox bitmap size.

Visual Basic automatically uses the class name of the control as the tool tip text when users hover the mouse pointer over the icon in the Toolbox.

**Tip** When creating bitmaps, remember that for many forms of color-blindness, colors with the same overall level of brightness will appear to be the same. You can avoid this by restricting the bitmap to white, black, and shades of gray, or by careful color selection.

# Visual Basic: Windows Controls

**Visual Studio 6.0**

# ToolTipText Property (ActiveX Controls)

See Also Example Applies To

Returns or sets a ToolTip.

**Syntax**

*object*.**ToolTipText** [= *ToolTipText*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *ToolTipText* | The string that is displayed when the mouse pointer hovers over the object. |

# Visual Basic Reference

**Visual Studio 6.0**

# ToolTipText Property

See Also Example Applies To

Returns or sets a ToolTip.

**Syntax**

*object*.**ToolTipText** [= *string*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *string* | A string associated with an object in the Applies To list. that appears in a small rectangle below the object when the user's cursor hovers over the object at run time for about one second. |

**Remarks**

If you use only an image to label an object, you can use this property to explain each object with a few words.

At design time you can set the **ToolTipText** property string in the control's properties dialog box.

For the **Toolbar** and **TabStrip** controls, you must set the **ShowTips** property to True to display ToolTips.

# Visual Basic Extensibility Reference

**Visual Studio 6.0**

# Top Property

See Also Example Applies To Specifics

Returns or sets a Single specifying the location of the top of the window on the screen in twips. Read/write.

**Remarks**

The value returned by the **Top** property depends on whether or not the window is docked, linked, or in docking view.

**Note** Changing the **Top** property setting of a linked or docked window has no effect as long as the window remains linked or docked.

# Visual Basic Reference

**Visual Studio 6.0**

# TopIndex Property

See Also Example Applies To

Returns or sets a value that specifies which item in a **ComboBox, DirListBox, DriveListBox, FileListBox,** or **ListBox** control is displayed in the topmost position. Not available at design time.

**Syntax**

*object*.**TopIndex** [= *value*]

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

| Part | Description |
|---|---|
| *Object* | An object expression that evaluates to an object in the Applies To list. |
| *Value* | The number of the list item that is displayed in the topmost position. The default is 0, or the first item in the list. |

**Remarks**

Use this property to scroll through a control without selecting an item.

If the **Columns** property is set to 0 for the **ListBox** control, the item is displayed at the topmost position if there are enough items below it to fill the visible portion of the list.

If the **Columns** property setting is greater than 0 for the **ListBox** control, the item's column moves to the leftmost position without changing its position within the column.

# Visual Basic Reference

# TopIndex Property Example

This example fills a **ListBox** control with names of screen fonts and then scrolls through the **ListBox** when you click the form. To try this example, paste the code into the Declarations section of a form that contains a **ListBox** control, and then press F5 and click the form.

```
Private Sub Form_Load ()
   Dim I   ' Declare variable.
   For I = 0 To Screen.FontCount -1   ' Fill list box with
      List1.AddItem Screen.Fonts(I)   ' screen font names.
   Next I
End Sub

Private Sub Form_Click ()
   Dim X   ' Declare variable.
   X = List1.TopIndex   ' Get current index.
   List1.TopIndex = List1.TopIndex + 5   ' Reset topmost item.
   If List1.TopIndex = X Then List1.TopIndex = 0
End Sub
```

# Visual Basic Extensibility Reference

**Visual Studio 6.0**

# TopLine Property

See Also Example Applies To Specifics

Returns a Long specifying the line number of the line at the top of the code pane or sets the line showing at the top of the code pane. Read/write.

**Remarks**

Use the **TopLine** property to return or set the line showing at the top of the code pane. For example, if you want line 25 to be the first line showing in a code pane, set the **TopLine** property to 25.

The **TopLine** property setting must be a positive number. If the **TopLine** property setting is greater than the actual number of lines in the code pane, the setting will be the last line in the code pane.

# Visual Basic Extensibility Reference

## TopLine Property Example

The following example uses the **TopLine** property to return the line number of the top line in the specified code pane.

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

# Visual Basic: MSFlexGrid/MSHFlexGrid Controls

**Visual Studio 6.0**

# TopRow Property (MSHFlexGrid)

See Also Example Applies To

Returns or sets the uppermost visible row (other than a fixed row) in the **MSHFlexGrid**. This property is not available at design time.

**Syntax**

*object*.**TopRow** [=*number*]

The **TopRow** 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 specifies the uppermost row in the **MSHFlexGrid**. |

**Remarks**

You can use this property to programmatically read or set the visible top row of the **MSHFlexGrid**. Use the **LeftCol** property to determine the leftmost visible column in the **MSHFlexGrid**.

The largest row number that you can use when setting **TopRow** is the total number of rows minus the number of rows that are visible in the **MSHFlexGrid**. If this property is set to a greater row number, the **MSHFlexGrid** will reset it to this largest possible value.

# Visual Basic for Applications Reference

**Visual Studio 6.0**

# TotalSize Property

See Also Example Applies To Specifics

**Description**

Returns the total space, in bytes, of a drive or network share.

**Syntax**

*object*.**TotalSize**

The *object* is always a **Drive** object.

**Remarks**

The following code illustrates the use of the **TotalSize** property:

```
Sub ShowSpaceInfo(drvpath)
    Dim fs, d, s
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set d = fs.GetDrive(fs.GetDriveName(fs.GetAbsolutePathName(drvpath)))
    s = "Drive " & d.DriveLetter & ":"
    s = s & vbCrLf
    s = s & "Total Size: " & FormatNumber(d.TotalSize/1024, 0) & " Kbytes"
    s = s & vbCrLf
    s = s & "Available: " & FormatNumber(d.AvailableSpace/1024, 0) & " Kbytes"
    MsgBox s
End Sub
```

# Visual Basic: Multimedia MCI Control

**Visual Studio 6.0**

# Track Property (Multimedia MCI Control)

See Also Example Applies To

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

**Syntax**

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

**Remarks**

The argument *track&* specifies the track number.

This property is used only to get information about a particular track. It has no relationship to the current track.

**Data Type**

Long

# Visual Basic Reference

**Visual Studio 6.0**

# TrackDefault Property

See Also Example Applies To

Returns or sets a value that determines whether the **Printer** object always points to the same printer or changes the printer it points to if you change the default printer setting in the operating system's Control Panel. Not available at design time.

### Syntax

*object*.**TrackDefault** [= *boolean*]

The **TrackDefault** 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 the printer *object* points to, as described in Settings. |

### Settings

The settings for *boolean* are:

| Setting | Description |
|---|---|
| **True** | (Default) The **Printer** object changes the printer it points to when you change the default printer settings in the operating system's Control Panel. |
| **False** | The **Printer** object continues to point to the same printer even though you change the default printer settings in the operating system's Control Panel. |

### Remarks

Changing the **TrackDefault** property setting while a print job is in progress sends an implicit **EndPage** statement to the **Printer** object.

# Visual Basic: Multimedia MCI Control

**Visual Studio 6.0**

# TrackLength Property (Multimedia MCI Control)

See Also Example Applies To

Specifies the length, as defined in the **Multimedia MCI** control **TimeFormat** property, of the track given by the **Track** property. This property is not available at design time and is read-only at run time.

**Syntax**

[*form.*]*MMControl.***TrackLength**

**Data Type**

Long

# Visual Basic: Multimedia MCI Control

**Visual Studio 6.0**

# TrackPosition Property (Multimedia MCI Control)

See Also Example Applies To

Specifies the starting position, as defined in the **Multimedia MCI** control **TimeFormat** property, of the track given by the **Track** property. This property is not available at design time and is read-only at run time.

**Syntax**

[*form.*]*MMControl*.**TrackPosition**

**Data Type**

Long

# Visual Basic: Multimedia MCI Control

**Visual Studio 6.0**

# Tracks Property (Multimedia MCI Control)

See Also Example Applies To

Specifies the number of tracks available on the current MCI device. This property is not available at design time and is read-only at run time.

**Syntax**

[*form.*]*MMControl.***Tracks**

**Data Type**

Long

# Visual Basic: Windows Controls

**Visual Studio 6.0**

# TrailingForeColor Property

See Also Example Applies To

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

**Syntax**

*object*.**TrailingForeColor** [= *color*]

The **TrailingForeColor** 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, as described in Settings. |

**Settings**

Visual Basic uses the Microsoft Windows operating environment red-green-blue (RGB) color scheme. The settings for *color* are:

| Setting | Description |
|---|---|
| *Normal RGB colors* | Colors specified by using the Color palette or by using the RGB or QBColor functions in code. |
| *System default colors* | Colors specified by system color constants listed in the Visual Basic (VBRUN) object library in the Object Browser. The Windows operating system 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. By default, trailing dates are displayed in vbWhite.

The **TrailingForeColor** property can be used with the **MonthBackColor**, **TitleBackColor** and **TitleForeColor** 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 three 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

# TitleBackColor, TitleForeColor, TrailingForeColor Properties Example

The example changes the appearance of a **MonthView** control by resetting the **TitleBackColor**, **TitleForeColor**, and **TrailingForeColor** properties. To try the example, place a **MonthView** control on a form, and paste the code into the Declarations section of the code module. Run the project, and double-click the form to see the calendar change.

```
Private Sub Form_DblClick()
   With MonthView1
      .TitleBackColor = vbBlue
      .TitleForeColor = vbWhite
      .TrailingForeColor = vbRed
   End With
End Sub
```

# Visual Basic: RDO Data Control

**Visual Studio 6.0**

# Transactions Property (Remote Data)

See Also Example Applies To

Returns a value that indicates whether an object supports the recording of a series of changes that can later be rolled back (undone) or committed (saved).

**Syntax**

*object*.**Transactions**

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

**Return Values**

The **Transactions** property return values are:

| Value | Description |
|---|---|
| **True** | The object supports transactions. |
| **False** | The object doesn't support transactions. |

**Remarks**

Check the **Transactions** property before using the **BeginTrans** method to make sure that transactions are supported. When **Transactions** is **False,** using the **BeginTrans**, **CommitTrans**, or **RollbackTrans** method has no effect.

The **Transactions** property calls the ODBC **SQLGetInfo** function to determine if the ODBC driver is *capable* of supporting transactions, not if the current result set is updatable. You can always call the **BeginTrans** method on the **rdoConnection** object if the **Transactions** property is **True** even for read-only **rdoResultset** objects.

# Visual Basic Reference

**Visual Studio 6.0**

# TrueValue Property

See Also Example Applies To

Sets or returns a value used to format and unformat Boolean **True** values. Read/write both at design time and run time.

**Syntax**

*object*.**TrueValue** [= *value*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *value* | Optional Variant. In the Format event, if the data is a Boolean **True**, *value* is returned; if **TrueValue** is empty, then the integer value 0 is returned. In the Unformat event, if **TrueValue** is empty and the value is 0, or if the data matches *value*, a Boolean **True** is written to the database. Defaults to True. |

**Remarks**

Ignored unless the **Type** property is set to **fmtBoolean**. The **TrueValue** property is read each time data is fetched.

# Visual Basic Reference

**Visual Studio 6.0**

# TwipsPerPixelX, TwipsPerPixelY Properties

See Also Example Applies To

Return the number of logical twips per pixel for an object measured horizontally (**TwipsPerPixelX**) or vertically (**TwipsPerPixelY**).

**Syntax**

*object*.**TwipsPerPixelX**

*object*.**TwipsPerPixelY**

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

**Remarks**

Windows API routines generally require measurements in pixels. You can use these properties to convert measurements quickly without changing an object's **ScaleMode** property setting.

# Visual Basic Reference

**Visual Studio 6.0**

# Type Property (DEDesigner Extensibility)

See Also Example Applies To

Returns or sets the data type of the DEField or DEParameter object. This property is read-only for the **DEField** object and is read-write for the **DEParameter** object.

**Syntax**

*object*.**Type** [=*value*]

The **Type** 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 DEField or DEParameter type. |

**Remarks**

This property corresponds to the ADO Field or Parameter Type properties.

# Visual Basic for Applications Reference

**Visual Studio 6.0**

# Type Property

See Also Example Applies To Specifics

**Description**

Returns information about the type of a file or folder. For example, for files ending in .TXT, "Text Document" is returned.

**Syntax**

*object*.**Type**

The *object* is always a **File** or **Folder** object.

**Remarks**

The following code illustrates the use of the **Type** property to return a folder type. In this example, try providing the path of the Recycle Bin or other unique folder to the procedure.

```
Sub ShowFileSize(filespec)
    Dim fs, f, s
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set f = fs.GetFolder(filespec)
    s = UCase(f.Name) & " is a " & f.Type
    MsgBox s, 0, "File Size Info"
End Sub
```

**Visual Studio 6.0**

*Visual Basic: MSChart Control*

# Type Property (MSChart)

See Also Example Applies To

Returns or sets the scale type of an axis.

**Syntax**

*object*.**Type** [ = *type*]

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *type* | A VtChScaleType constant describing the axis scale type. |

# Visual Basic Reference

**Visual Studio 6.0**

# Type Property (Picture)

See Also Example Applies To

Returns the graphic format of a **Picture** object. Not available at design time; read-only at run time.

**Syntax**

*object*.**Type**

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

**Return Values**

The return values for the **Type** property are:

| Constant | Value | Description |
|---|---|---|
| **vbPicTypeNone** | 0 | Picture is empty |
| **vbPicTypeBitmap** | 1 | Bitmap (.bmpBMP files) |
| **vbPicTypeMetafile** | 2 | Metafile (.wmfWMF files) |
| **vbPicTypeIcon** | 3 | Icon (.icoICO files) |
| **vbPicTypeEMetafile** | 4 | Enhanced Metafile (.emfEMF files) |

**Remarks**

These constants are listed in the Visual Basic (VB) object library in the Object Browser.

# Visual Basic Reference

# Type, Width Properties Example

This example reads the setting of the **Type** and **Width** properties of a **Picture** object in a **PictureBox** control. To try this example, paste the code into the Declarations section of a form that contains a **PictureBox** whose **Picture** property is set to an icon, and then press F5 and click the form.

```
Private Sub Form_Click()
   If Picture1.Picture.Type = vbPicTypeIcon Then
      Print "The graphic in the picture box is an icon."
   Else
      Print "The Picture property isn't set to an icon."
   End If
   Print "Width of the graphic in HiMetrics is " & Picture1.Picture.Width
   Print "Width of picture box itself in twips is " & Picture1.Width
End Sub
```

# Visual Basic: RDO Data Control

**Visual Studio 6.0**

# Type Property (Remote Data)

See Also Example Applies To

Returns or sets a value that indicates the type or data type of an object.

**Syntax**

*object*.**Type** [= *value*]

The **Type** 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 value that specifies a datatype, as described in Return Values. |

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

**Return Values**

For an **rdoColumn** or **rdoParameter** object, the **Type** property returns an Integer. You can also set the **Type** property on the **rdoParameter** object to indicate the datatype of a specific procedure argument. The valid values are:

| Constant | Value | Description |
|---|---|---|
| **rdTypeCHAR** | 1 | Fixed-length character string. Length set by **Size** property. |
| **rdTypeNUMERIC** | 2 | Signed, exact, numeric value with precision p and scale s (1 ≤ p ≤15; 0 ≤ s ≤ p). |
| **rdTypeDECIMAL** | 3 | Signed, exact, numeric value with precision p and scale s (1 ≤ p ≤15; 0 ≤ s ≤ p). |
| **rdTypeINTEGER** | 4 | Signed, exact numeric value with precision 10, scale 0 (signed: $-2^{31} \le n \le 2^{31}-1$; unsigned: $0 \le n \le 2^{32}-1$). |
| **rdTypeSMALLINT** | 5 | Signed, exact numeric value with precision 5, scale 0 (signed: -32,768 ≤ n ≤ 32,767, unsigned: 0 ≤ n ≤ 65,535). |
| **rdTypeFLOAT** | 6 | Signed, approximate numeric value with mantissa precision 15 (zero or absolute value $10^{-308}$ to $10^{308}$). |

| | | |
|---|---|---|
| **rdTypeREAL** | 7 | Signed, approximate numeric value with mantissa precision 7 (zero or absolute value $10^{-38}$ to $10^{38}$). |
| **rdTypeDOUBLE** | 8 | Signed, approximate numeric value with mantissa precision 15 (zero or absolute value $10^{-308}$ to $10^{308}$). |
| **rdTypeDATE** | 9 | Date data source dependent. |
| **rdTypeTIME** | 10 | Time data source dependent. |
| **rdTypeTIMESTAMP** | 11 | TimeStamp data source dependent. |
| **rdTypeVARCHAR** | 12 | Variable-length character string. Maximum length 255. |
| **rdTypeLONGVARCHAR** | -1 | Variable-length character string. Maximum length determined by data source. |
| **rdTypeBINARY** | -2 | Fixed-length binary data. Maximum length 255. |
| **rdTypeVARBINARY** | -3 | Variable-length binary data. Maximum length 255. |
| **rdTypeLONGVARBINARY** | -4 | Variable-length binary data. Maximum data source dependent. |
| **rdTypeBIGINT** | -5 | Signed, exact numeric value with precision 19 (signed) or 20 (unsigned), scale 0; (signed: $-2^{63} \le n \le 2^{63}-1$; unsigned: $0 \le n \le 2^{64}-1$). |
| **rdTypeTINYINT** | -6 | Signed, exact numeric value with precision 3, scale 0; (signed: $-128 \le n \le 127$, unsigned: $0 \le n \le 255$). |
| **rdTypeBIT** | -7 | Single binary digit. |

For an **rdoQuery** object, the **Type** property returns an Integer. The return values are:

| Constant | Value | Query type |
|---|---|---|
| **rdQSelect** | 0 | Select |
| **rdQAction** | 1 | Action |
| **rdQProcedures** | 2 | Procedural |
| **rdQCompound** | 3 | The query contains both action and select statements |

For an **rdoResultset** object, the **Type** property returns an Integer that determines the type of **rdoResultset**. The return values are:

| Constant | Value | rdoResultset type |
|---|---|---|
| **rdOpenForwardOnly** | 0 | 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. |

**Note** Not all ODBC drivers or data sources support every type of **rdoResultset** cursor type. If you choose a cursor that is not supported, the ODBC driver attempts to revert to a supported type. If no supported type is available, a trappable error is fired.

For an **rdoTable** object, the **Type** property returns a String. The settings for *value* are determined by the data source driver.

Typically, this string value is "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY". "LOCAL TEMPORARY", "ALIAS", "SYNONYM" or some other data source-specific type identifier.

**Remarks**

Depending on the object, the **Type** property indicates:

| Object | Type indicates |
|---|---|
| **rdoColumn, rdoParameter** | Object data type |
| **rdoQuery** | Type of query |
| **rdoResultset** | Type of **rdoResultset** |
| **rdoTable** | Type of table on data source |

In some cases, you must override the **Type** property assignment made by RDO when creating some types of parameter queries. For example, if a parameter is passed to an expression inside of an SQL statement, the ODBC driver might not be able to determine the correct type. In these cases, you can force a specific parameter to be handled as the correct type by simply setting the **rdoParameter** object's **Type** property. This is the only situation that permits you to change the **Type** property. In all other cases, this property is read-only.

# Visual Basic Extensibility Reference

**Visual Studio 6.0**

# Type Property (VBA Add-In Object Model)

See Also Example Applies To Specifics

Returns a numeric or string value containing the type of object. Read-only.

**Return Values**

The **Type** property settings for the **Window** object are described in the following table:

| Constant | Value | Description |
|---|---|---|
| **vbext_wt_CodeWindow** | 0 | **Code** window |
| **vbext_wt_Designer** | 1 | Designer |
| **vbext_wt_Browser** | 2 | **Object Browser** |
| **vbext_wt_Immediate** | 5 | **Immediate** window |
| **vbext_wt_ProjectWindow** | 6 | Project window |
| **vbext_wt_PropertyWindow** | 7 | Properties window |
| **vbext_wt_Find** | 8 | **Find** dialog box |
| **vbext_wt_FindReplace** | 9 | **Search and Replace** dialog box |
| **vbext_wt_LinkedWindowFrame** | 11 | Linked window frame |
| **vbext_wt_MainWindow** | 12 | Main window |
| **vbext_wt_Watch** | 3 | **Watch** window |
| **vbext_wt_Locals** | 4 | **Locals** window |
| **vbext_wt_Toolbox** | 10 | **Toolbox** |
| **vbext_wt_ToolWindow** | 15 | **Tool** window |

The **Type** property settings for the **VBComponent** object are described in the following table:

| Constant | Value | Description |
|---|---|---|
| **vbext_ct_StdModule** | 1 | Standard module |
| **vbext_ct_ClassModule** | 2 | Class module |
| **vbext_ct_MSForm** | 3 | Microsoft Form |
| **vbext_ct_ActiveXDesigner** | 11 | ActiveX Designer |
| **vbext_ct_Document** | 100 | Document Module |

The **Type** property settings for the **Reference** object are described in the following table:

| Constant | Value | Description |
|---|---|---|
| **vbext_rk_TypeLib** | 0 | Type library |
| **vbext_rk_Project** | 1 | Project |

# Visual Basic Extensibility Reference

## Type Property Example

The following example uses the **Type** property to return a value indicating the type of the specified member of the **VBComponents** collection in a particular project. The value returned is a number that corresponds to a predefined constant for one of the component object types.

```
Debug.Print Application.VBE.VBProjects(1).VBComponents(1).Type
```

# Visual Basic Reference

**Visual Studio 6.0**

# Type Property

See Also Example Applies To

Sets or returns the type of formatting applied by the **StdDataFormat** object. Based on this setting, other properties of the object are used to format the value. Read/write both at design time and run time.

**Syntax**

*object*.**Type** = *formattype*

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

| Part | Description |
|---|---|
| *object* | An object expression that evaluates to an object in the Applies To list. |
| *formattype* | Required enumerated integer. Specifies the type of formatting desired, as described in Settings. |

**Settings**

The settings for *formattype* are:

| Constant | Setting | Description |
|---|---|---|
| fmtGeneral | 0 | No formatting is applied to the value. |
| fmtCustom | 1 | The format string contained in the **Format** property is used to format and unformat the value. |
| fmtPicture | 2 | The value is treated as a picture. A binary object read from the database is translated into a picture object, and written back to the database as a binary object. |
| fmtObject | 3 | The value is treated as an object. The **StdDataFormat** object expects to retrieve a CLSID from the database. The CLSID is used to instantiate the object, which is returned from the **StdDataFormat** object. The CLSID is then read from the object being formatted when the value is returned to the database. |
| fmtCheckbox | 4 | This setting is used to bind a check box to a database field. The database value is treated as the check box **Value** property. Return values are specified in the table under Remarks. The formatted values are only meaningful for a check box control whose **Value** property is of type OLE_TRISTATE. For general Boolean types, use **fmtBoolean**. |

| | | |
|---|---|---|
| fmtBoolean | 5 | The value is treated as a Boolean value. When this type is chosen the **TrueValue** and **FalseValue** properties, and, depending on the type of database, the **NullValue** property, are used to determine how the value should be formatted. If none of those properties are set, the default behavior is to return the logical value from the database. |
| fmtBytes | 6 | The value is treated as a Byte. |

### Remarks

This table lists return values associated with the **fmtCheckbox** setting, as described above.

| Database Value | Formatted Value |
|---|---|
| Logical False | 0 |
| Logical True | 1 |
| Null | 2 |

To reset the **Type** property at run time you must unbind, set the property, and then rebind.