

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

Tab Object

[See Also](#) [Example](#) [Properties](#) [Methods](#) [Events](#)

A **Tab** object represents an individual tab in the **Tabs** collection of a **TabStrip** control.

Remarks

For each **Tab** object, you can use various properties to specify its appearance, and you can specify its state with the **Selected** property.

At design time, use the Insert Tab and Remove Tab buttons on the Tabs tab in the Properties Page of the **TabStrip** control to insert and remove tabs, and use the text boxes to specify any of these properties for a **Tab** object: **Caption**, **Image**, **ToolTipText**, **Tag**, **Index**, and/or **Key**. You can also specify these properties at run time.

Use the **Caption** and **Image** properties, separately or together, to label or put an icon on a tab.

- To use the **Caption** property, in the Caption text box on the Tabs tab in the Properties Page of the **TabStrip** control, type the text you want to appear on the tab or button at run time.
- To use the **Image** property, put an **ImageList** control on the form and fill the **ListImages** collection with **ListImage** objects, each of which has an index number and an optional key, if you add one. On the General tab in the Properties Page of the **TabStrip** control, select that **ImageList** to associate it with the **TabStrip** control. In the Image text box on the Tabs tab, type the index number or key of the **ListImage** object that should appear on the **Tab** object.

Use the **ToolTipText** property to temporarily display a string of text in a small rectangular box at run time when the user's cursor hovers over the tab. To set the **ToolTipText** property at design time, select the **ShowTips** checkbox on the General tab, and then in the ToolTipText text box on the Tabs tab, type the ToolTip string.

To return a reference to a **Tab** object a user has selected, use the **SelectedItem** property; to determine whether a specific tab is selected, use the **Selected** property. These properties are useful in conjunction with the BeforeClick event to verify or record data associated with the currently-selected tab before displaying the next tab the user selects.

Each **Tab** object also has read-only properties you can use to reference a single **Tab** object in the **Tabs** collection: **Left**, **Top**, **Height** and **Width**.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

Tabs Collection

[See Also](#) [Example](#) [Properties](#) [Methods](#) [Events](#)

A **Tabs** collection contains a collection of **Tab** objects.

Syntax

tabstrip.**Tabs**(*index*)

tabstrip.**Tabs.Item**(*index*)

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

Part	Description
<i>tabstrip</i>	An object expression that evaluates to a TabStrip control.
<i>index</i>	An integer or string that uniquely identifies a member of an object collection. The integer is the value of the Index property of the desired Tab object; the string is the value of the Key property of the desired Tab object.

At design time, use the Insert Tab and Remove Tab buttons on the Tabs tab in the Properties Page of the **TabStrip** control to add and remove **Tab** objects from the **Tabs** collection.

The **Tabs** collection uses the **Count** property to return the number of tabs in the collection. To manipulate the **Tab** objects in the **Tabs** collection, use these methods at run time:

- **Add** adds **Tab** objects to the **TabStrip** control.
- **Item** retrieves the **Tab** identified by its **Key** or **Index** from the collection.
- **Clear** removes all **Tab** objects from the collection.
- **Remove** removes the **Tab** identified by its **Key** or **Index** from the collection.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

TabStrip Control

[See Also](#) [Example](#) [Properties](#) [Methods](#) [Events](#)

A **TabStrip** control is like the dividers in a notebook or the labels on a group of file folders. By using a **TabStrip** control, you can define multiple pages for the same area of a window or dialog box in your application.

Syntax

TabStrip

Remarks

The control consists of one or more **Tab** objects in a **Tabs** collection. At both design time and [run time](#), you can affect the **Tab** object's appearance by setting properties. You can also add and remove tabs using the Properties Page of the **TabStrip** control at design time, or add and remove **Tab** objects at run time using methods.

The **Style** property determines whether the **TabStrip** control looks like push buttons (Buttons) or notebook tabs (Tabs). At design time when you put a **TabStrip** control on a form, it has one notebook tab. If the **Style** property is set to **tabTabs**, then there will be a border around the **TabStrip** control's internal area. When the **Style** property is set to **tabButtons**, no border is displayed around the internal area of the control, however, that area still exists.

To set the overall size of the **TabStrip** control, use its drag handles and/or set the **Top**, **Left**, **Height**, and **Width** properties. Based on the control's overall size at run time, Visual Basic automatically determines the size and position of the internal area and returns the Client-coordinate properties **ClientLeft**, **ClientTop**, **ClientHeight**, and **ClientWidth**. The **MultiRow** property determines whether the control can have more than one row of tabs, the **TabWidthStyle** property determines the appearance of each row, and, if **TabWidthStyle** is set to **tabFixed**, you can use the **TabFixedHeight** and **TabFixedWidth** properties to set the same height and width for all tabs in the **TabStrip** control.

The **TabStrip** control is not a container. To contain the actual pages and their objects, you must use **Frame** controls or other containers that match the size of the internal area which is shared by all **Tab** objects in the control. If you use a control array for the container, you can associate each item in the array with a specific **Tab** object, as in the following example:

Option Explicit

```
Private mintCurFrame As Integer ' Current Frame visible
```

```
Private Sub Tabstrip1_Click()  
    If Tabstrip1.SelectedItem.Index = mintCurFrame _  
        Then Exit Sub ' No need to change frame.  
    ' Otherwise, hide old frame, show new.  
    Frame1(Tabstrip1.SelectedItem.Index).Visible = True  
    Frame1(mintCurFrame).Visible = False  
    ' Set mintCurFrame to new value.  
    mintCurFrame = Tabstrip1.SelectedItem.Index  
End Sub
```

Note When grouping controls on a container, you must use the show/hide strategy shown above instead of using the **Zorder Method** to bring a frame to the front. Otherwise, controls that implement access keys (ALT + access key) will still

respond to keyboard commands, even if the container is not the topmost control. Also note that you must segregate groups of **OptionButton** controls by placing each group on its own container, or else all **OptionButtons** on the form will behave as one large group of **OptionButtons**.

Tip Use a **Frame** control with its **BorderStyle** set to None as the container instead of a **PictureBox** control. A **Frame** control uses less overhead than a **PictureBox** control.

The **Tabs** property of the **TabStrip** control is the [collection](#) of all the **Tab** objects. Each **Tab** object has properties associated with its current state and appearance. For example, you can associate an **ImageList** control with the **TabStrip** control, and then use images on individual tabs. You can also associate a **ToolTip** with each **Tab** object.

Distribution Note The **TabStrip** control is part of a group of custom controls that are found in the MSCOMCTL.OCX file. To use the **TabStrip** control in your application, you must add the MSCOMCTL.OCX file to the project. When distributing your application, install the MSCOMCTL.OCX file in the user's Microsoft Windows SYSTEM folder. For more information on how to add a custom control to a project, see the *Programmer's Guide*.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

TextBox Control

[See Also](#) [Example](#) [Properties](#) [Methods](#) [Events](#)

A **TextBox** control, sometimes called an edit field or edit control, displays information entered at design time, entered by the user, or assigned to the control in code at [run time](#).

Syntax

TextBox

Remarks

To display multiple lines of text in a **TextBox** control, set the **MultiLine** property to **True**. If a multiple-line **TextBox** doesn't have a horizontal scroll bar, text wraps automatically even when the **TextBox** is resized. To customize the scroll bar combination on a **TextBox**, set the **ScrollBars** property.

Scroll bars will always appear on the **TextBox** when its **MultiLine** property is set to **True**, and its **ScrollBars** property is set to anything except **None** (0).

If you set the **MultiLine** property to **True**, you can use the **Alignment** property to set the alignment of text within the **TextBox**. The text is left-justified by default. If the **MultiLine** property is **False**, setting the **Alignment** property has no effect.

A **TextBox** control can also act as a destination link in a DDE conversation.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

TextBox Control (Data Report Designer)

See Also [Example](#) [Properties](#) [Methods](#) [Events](#)


The Data Report designer's TextBox control is a data-bound control that can only display text from a database at [run time](#).

Syntax

RptTextBox

The Data Report designer's TextBox control cannot display information entered by the user at design time, and cannot be used for data input at run time.

© 2017 Microsoft



This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

TextLayout Object

See Also [Example](#) Properties Methods Events

Represents text positioning and orientation.

Syntax

TextLayout

© 2017 Microsoft

TextLayout Object Example

The following example sets the title text position and orientation for a chart.

```
Private Sub Command1_Click()  
    ' Sets the title text position and orientation.  
    With Form1.MSChart1.Title  
        .Location.Visible = True  
        .Location.LocationType = VtChLocationTypeLeft  
        .Text = "Title TextLayout"  
    End With  
    With Form1.MSChart1.Title.TextLayout  
        .Orientation = VtOrientationUp  
        .HorzAlignment = VtHorizontalAlignmentCenter  
        .VertAlignment = VtVerticalAlignmentCenter  
    End With  
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

TextStream Object

[See Also](#) [Example](#) [Properties](#) [Methods](#) [Events](#) [Specifics](#)

Description

Facilitates sequential access to file.

Syntax

TextStream.{*property* | *method*}

The *property* and *method* arguments can be any of the properties and methods associated with the **TextStream** object. Note that in actual usage **TextStream** is replaced by a variable placeholder representing the **TextStream** object returned from the **FileSystemObject**.


Remarks

In the following code, *a* is the **TextStream** object returned by the **CreateTextFile** method on the **FileSystemObject**:

```
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.CreateTextFile("c:\testfile.txt", True)
a.WriteLine("This is a test.")
a.Close
```

WriteLine and **Close** are two methods of the **TextStream** Object.

© 2017 Microsoft



This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

Tick Object

See Also [Example](#) [Properties](#) [Methods](#) [Events](#)

A marker indicating a division along a chart axis.

Syntax

Tick

© 2017 Microsoft

Tick Object Example

The following example sets the tick length and style for the y axis on a chart.

```
Private Sub Command1_Click()  
    ' Set the tick for y axis.  
    With Form1.MSChart1.Plot.Axis(VtChAxisIdY, 1).Tick  
        .Length = 500  
        .Style = VtChAxisTickStyleOutside  
    End With  
End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic Reference

Visual Studio 6.0

Timer Control

[See Also](#) [Example](#) [Properties](#) [Methods](#) [Events](#)

A **Timer** control can execute code at regular intervals by causing a Timer event to occur.

Syntax

Timer

Remarks

The **Timer** control, invisible to the user, is useful for background processing.

You can't set the **Enabled** property of a **Timer** for a multiple selection of controls other than **Timer** controls.

There is no practical limit on the number of active timer controls you can have in Visual Basic running under Windows 95, Windows 98, or Windows NT.

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Studio 6.0

Visual Basic: MSChart Control

Title Object

[See Also](#) [Example](#) [Properties](#) [Methods](#) [Events](#)

Text identifying the chart.

Syntax

Title

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

Toolbar Control

[See Also](#) [Example](#) [Properties](#) [Methods](#) [Events](#)

A **Toolbar** control contains a collection of **Button** objects used to create a toolbar that is associated with an application.

Syntax

Toolbar

Remarks

Typically, a toolbar contains buttons that correspond to items in an application's menu, providing a graphic interface for the user to access an application's most frequently used functions and commands.

The **Toolbar** control allows you to create toolbars by adding **Button** objects to a **Buttons** collection. Each **Button** object can have optional text or an image, or both, supplied by an associated **ImageList** control. You can display an image on a button with the **Image** property, or display text with the **Caption** property, or both, for each **Button** object. At design time, you can add **Button** objects to the control using the Properties Page of the **Toolbar** control. At run time, you can add or remove buttons from the **Buttons** collection using the **Add** and **Remove** methods.

To program the **Toolbar**, add code to the **ButtonClick** event to respond to the selected button. You can also determine the behavior and appearance of each **Button** object using the **Style** property. For example, if four buttons are assigned the **ButtonGroup** style, only one button can be pressed at any time and at least one button is always pressed.

You can create space for other controls on the toolbar by assigning a **Button** object the **Placeholder** style, then positioning a control over the placeholder. For example, to place a drop-down combo box on a toolbar at design time, add a **Button** object with the **Placeholder** style and size it as wide as a **ComboBox** control. Then place a **ComboBox** control on the placeholder.

Double clicking a toolbar at run time invokes the **Customize Toolbar** dialog box, which allows the user to hide, display, or rearrange toolbar buttons. To enable or disable the dialog box, use the **AllowCustomize** property. You can also invoke the **Customize Toolbar** dialog box using the **Customize** method. If you wish to save and restore the state of a toolbar, or allow the user to do so, two methods are provided: the **SaveToolbar** and **RestoreToolbar** methods. The **Change** event, generated when a toolbar is altered, is typically used to invoke the **SaveToolbar** method.

Note The **Customize** dialog box also includes a **Help** button. Use the **HelpFile** and **HelpContextID** properties to determine which (if any) help file is displayed when the end user clicks the **Help** button.

Usability is further enhanced by programming **ToolTipText** descriptions of each **Button** object. To display **ToolTips**, the **ShowTips Property** must be set to **True**. When the user invokes the **Customize Toolbar** dialog box, clicking a button causes a description of the button to be displayed in the dialog box; this description can be programmed by setting the **Description** property.

Distribution Note The **Toolbar** control is part of a group of ActiveX controls that are found in the **MSCOMCTL.OCX** file. To use the **Toolbar** control in your application, you must add the **MSCOMCTL.OCX** file to the project. When distributing your application, install the **MSCOMCTL.OCX** file in the user's Microsoft Windows System or System32 (on Windows NT platforms)

folder. For more information on how to add an ActiveX control to a project, see "Loading ActiveX Controls," in the *Component Tools Guide*.

© 2017 Microsoft

Visual Basic: Windows Controls

Toolbar Control Example

This example adds **Button** objects to a **Toolbar** control using the **Add** method and assigns images supplied by the **ImageList** control. The behavior of each button is determined by the **Style** property. The code creates buttons that can be used to open and save files and includes a **ComboBox** control that is used to change the backcolor of the form. To try the example, place a **Toolbar**, **ImageList**, and a **ComboBox** on a form and paste the code into the form's Declarations section. Make sure that you insert the **ComboBox** directly on the **Toolbar** control. Run the example, click the various buttons and select from the combo box.

```
Private Sub Form_Load()
    ' Create object variable for the ImageList.
    Dim imgX As ListImage

    ' Load pictures into the ImageList control.
    Set imgX = ImageList1.ListImages. _
    Add(, "open", LoadPicture("Graphics\bitmaps\tlbr_w95\open.bmp"))
    Set imgX = ImageList1.ListImages. _
    Add(, "save", LoadPicture("Graphics\bitmaps\tlbr_w95\save.bmp"))
    Toolbar1.ImageList = ImageList1

    ' Create object variable for the Toolbar.
    Dim btnX As Button
    ' Add button objects to Buttons collection using
    ' the
    ' Add method. After creating each button, set both
    ' Description and ToolTipText properties.
    Toolbar1.Buttons.Add , , , tbrSeparator
    Set btnX = Toolbar1.Buttons.Add(, "open", , tbrDefault, "open")
    btnX.ToolTipText = "Open File"
    btnX.Description = btnX.ToolTipText
    Set btnX = Toolbar1.Buttons.Add(, "save", , tbrDefault, "save")
    btnX.ToolTipText = "Save File"
    btnX.Description = btnX.ToolTipText
    Set btnX = Toolbar1.Buttons.Add(, , , tbrSeparator)

    ' The next button has the Placeholder style. A
    ' ComboBox control will be placed on top of this
    ' button.
    Set btnX = Toolbar1.Buttons.Add(, "combo1", , tbrPlaceholder)
    btnX.Width = 1500 ' Placeholder width to accommodate a combobox.

    Show ' Show form to continue configuring ComboBox.

    ' Configure ComboBox control to be at same location
    ' as the
    ' Button object with the Placeholder style (key =
    ' "combo1").
    With Combo1
        .Width = Toolbar1.Buttons("combo1").Width
        .Top = Toolbar1.Buttons("combo1").Top
        .Left = Toolbar1.Buttons("combo1").Left
        .AddItem "Black" ' Add colors for text.
        .AddItem "Blue"
```



```
.AddItem "Red"
.ListIndex = 0
End With

End Sub

Private Sub Form_Resize()
' Configure ComboBox control.
With Combo1
.Width = Toolbar1.Buttons("combo1").Width
.Top = Toolbar1.Buttons("combo1").Top
.Left = Toolbar1.Buttons("combo1").Left
End With

End Sub

Private Sub toolbar1_ButtonClick(ByVal Button As Button)
' Use the Key property with the SelectCase statement to specify
' an action.
Select Case Button.Key
Case Is = "open"      ' Open file.
    MsgBox "Add code to open file here!"
Case Is = "save"     ' Save file.
    MsgBox "Add code to save file here!"
End Select

End Sub

Private Sub Combo1_Click()
' Change bgcolor of form using the ComboBox.
Select Case Combo1.ListIndex
Case 0
    Form1.BackColor = vbBlack
Case 1
    Form1.BackColor = vbBlue
Case 2
    Form1.BackColor = vbRed
End Select

End Sub
```

© 2017 Microsoft

This documentation is archived and is not being maintained.

Visual Basic: Windows Controls

Visual Studio 6.0

TreeView Control

[See Also](#) [Example](#) [Properties](#) [Methods](#) [Events](#)

A **TreeView** control displays a hierarchical list of **Node** objects, each of which consists of a label and an optional bitmap. A **TreeView** is typically used to display the headings in a document, the entries in an index, the files and directories on a disk, or any other kind of information that might usefully be displayed as a hierarchy.

Syntax

TreeView

Remarks

After creating a **TreeView** control, you can add, remove, arrange, and otherwise manipulate **Node** objects by setting properties and invoking methods. You can programmatically expand and collapse **Node** objects to display or hide all child nodes. Three events, the Collapse, Expand, and NodeClick event, also provide programming functionality.

You can navigate through a tree in code by retrieving a reference to **Node** objects using **Root**, **Parent**, **Child**, **FirstSibling**, **Next**, **Previous**, and **LastSibling** properties. Users can navigate through a tree using the keyboard as well. UP ARROW and DOWN ARROW keys cycle downward through all expanded **Node** objects. **Node** objects are selected from left to right, and top to bottom. At the bottom of a tree, the selection jumps back to the top of the tree, scrolling the window if necessary. RIGHT ARROW and LEFT ARROW keys also tab through expanded **Node** objects, but if the RIGHT ARROW key is pressed while an unexpanded **Node** is selected, the **Node** expands; a second press will move the selection to the next **Node**. Conversely, pressing the LEFT ARROW key while an expanded **Node** has the focus collapses the **Node**. If a user presses an ANSI key, the focus will jump to the nearest **Node** that begins with that letter. Subsequent pressings of the key will cause the selection to cycle downward through all expanded nodes that begin with that letter.

Several styles are available which alter the appearance of the control. **Node** objects can appear in one of eight combinations of text, bitmaps, lines, and plus/minus signs.

The **TreeView** control uses the **ImageList** control, specified by the **ImageList** property, to store the bitmaps and icons that are displayed in **Node** objects. A **TreeView** control can use only one **ImageList** at a time. This means that every item in the **TreeView** control will have an equal-sized image next to it when the **TreeView** control's **Style** property is set to a style which displays images.

Distribution Note The **TreeView** control is part of a group of ActiveX controls that are found in the MSCOMCTL.OCX file. To use the **TreeView** control in your application, you must add the MSCOMCTL.OCX file to the project. When distributing your application, install the MSCOMCTL.OCX file in the user's Microsoft Windows System or System32 directory.

© 2017 Microsoft