Visual Basic Reference

Visual Studio 6.0

Global Object

See Also Example Properties Methods Events

A **Global** object is an application object that enables you to access application-level properties and methods.

Syntax

Global

Remarks

Global is an Object data type. Because the **Global** object is an application object that is referenced automatically, it is not necessary to code a specific reference to this object.

Visual Basic Reference

Visual Studio 6.0

HScrollBar, VScrollBar Controls

See Also Example Properties Methods Events

Scroll bars provide easy navigation through a long list of items or a large amount of information. They can also provide an analog representation of current position. You can use a scroll bar as an input device or indicator of speed or quantityfor example, to control the volume of a computer game or to view the time elapsed in a timed process.

Syntax

HScrollBar

VScrollBar

Remarks

When you're using a scroll bar as an indicator of quantity or speed or as an input device, use the **Max** and **Min** properties to set the appropriate range for the control.

To specify the amount of change to report in a scroll bar, use the **LargeChange** property for clicking in the scroll bar, and the **SmallChange** property for clicking the arrows at the ends of the scroll bar. The scroll bar's **Value** property increases or decreases by the values set for the **LargeChange** and **SmallChange** properties. You can position the scroll box at run time by setting **Value** between 0 and 32,767, inclusive.

Visual Basic Extensibility Reference

Visual Studio 6.0

IDTExtensibility Interface

See Also Example Properties Methods Events

The **IDTExtensibility** interface contains methods that Visual Basic calls when an add-in is connected to it, whether through the Add-In Manager, or some other manner.

The **IDTExtensibility** interface contains pre-configured procedure templates (which includes their parameter lists) that you need to manage add-ins in Visual Basic.

Syntax

Implements IDTExtensiblity

Remarks

The usage of interfaces was introduced in Visual Basic 5.0. Interfaces enable you to choose a pre-configured procedure template from a module's **Procedure** drop down list, eliminating parameter list entry errors and allowing you to program your applications a bit faster.

An interface's methods are exposed through the **Implements** statement. When the above syntax is entered in the Declarations section of the Class module that handles an add-in's events, the interface's methods become available for your use through the module's **Procedure** and **Object** drop down boxes. To add the code to the module, simply select it from the drop down box.

The **IDTExtensiblity** interface currently contains four methods:

- OnAddinsUpdate Method
- OnConnection Method
- OnDisconnection Method
- OnStartupComplete Method

While these are methods to the **IDTExtensibility** interface, to you as a Visual Basic programmer, though, they act and behave like events. In other words, when an add-in is connected to Visual Basic, the **OnConnection** method is called automatically, similar to an event firing. When it is disconnected, the **OnDisconnection** method is called automatically, and so forth.

Important Since an interface is a contract between an object and Visual Basic, you must be sure to implement *all* of the methods in the interface. This means that all four **IDTExtensibility** interface methods are present in your Class module, each containing at least one executable statement. This can consist of as little as a single remark statement, but they must each contain at least one executable statement to prevent the compiler from removing them as empty procedures.

Visual Basic Reference

Visual Studio 6.0

Image Control

See Also Example Properties Methods Events

Use the **Image** control to display a graphic. An **Image** control can display a graphic from a bitmap, icon, or metafile, as well as enhanced metafile, JPEG, or GIF files.

Syntax

Image

Remarks

The **Image** control uses fewer system resources and repaints faster than a **PictureBox** control, but it supports only a subset of the **PictureBox** properties, events, and methods. Use the **Stretch** property to determine whether the graphic is scaled to fit the control or vice versa. Although you can place an **Image** control within a container, an **Image** control can't act as a container.

Note The Unisys Corporation has a patent that it alleges covers certain aspects of GIF-LZW compression technology, which the PictureBox and Image controls use. Microsoft Corporation obtained a license to the Unisys LZW patents in September, 1996. Microsoft's license does not, however, extend to software developers or third parties who use any Microsoft toolkit, language development, or operating system products to provide GIF read/write and/or any other LZW capabilities in their own products (for example, by way of DLLs and APIs).

If your commercial application uses one of these controls (and thus, the LZW technology), you may wish to obtain an independent legal opinion on the effect of the patent, or contact Unisys USA at http://www.unisys.com/ for more information.

Visual Basic Reference

Visual Studio 6.0

Image Control (Data Report Designer)

See Also Example Properties Methods Events

An Image control can display a graphic from a bitmap, icon, or metafile, as well as enhanced metafile, .jpeg, or .gif files.

Syntax

RptImage

Remarks

The Data Report designer version of the Image control is similar to the standard Visual Basic intrinsic **Image** control in displaying images on any application. Beyond this basic capability, however, some of the standard **Image** control's properties are unavailable on the Data Report version.

Visual Basic: Windows Controls

Visual Studio 6.0

ImageCombo Control

See Also Example Properties Methods Events

The **ImageCombo** control is a picture-enabled version of the standard Windows combo box. Each item in the list portion of the control can have a picture assigned to it.

In addition to supporting pictures, the **ImageCombo** provides an object and collection-based list control. Each item in the list portion of the control is a distinct **ComboItem** object, and together the items in the list make up the **ComboItems** collection. This makes it easy to specify properties such as tag text, ToolTip text, key value, and level of indentation on an item-by-item basis.

Syntax

ImageCombo

Remarks

With an **ImageCombo** control, you can display a list of items that includes pictures. Each item can have its own picture, or you can use the same picture for multiple list items.

The **ImageCombo** control contains a collection of **ComboItem** objects. A **ComboItem** object defines the various characteristics of the items that appear in the list portion of the control.

In addition to displaying pictures with list items, the **ImageCombo** control uses collections and objects to manage the list portion of the control. This makes it easy to manipulate entries in the list using familiar object and collection concepts, such as the **Add**, **Remove** and **Clear** methods, and the **For Each** and **With... End With** constructions.

Note The **ImageCombo** control is part of a group of ActiveX controls that are found in the MSCOMCTL.ocx file. To use the **ImageCombo** 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. For more information on how to add an ActiveX control to a project, see "Adding Controls to a Project" in the *Programmer's Guide*.

Visual Basic: Windows Controls

Visual Studio 6.0

ImageList Control

See Also Example Properties Methods Events

An **ImageList** control contains a collection of **ListImage** objects, each of which can be referred to by its index or key. The **ImageList** control is not meant to be used alone, but as a central repository to conveniently supply other controls with images.

Syntax

ImageList

Remarks

The **ImageList** control functions as a storehouse for images, and as such, it needs a second control to display the stored images. The second control can be any control that can display an image's **Picture** object, or it can be one of the Windows Common Controls that were specifically designed to bind to the **ImageList** control. These include the **ListView**, **ToolBar**, **TabStrip**, **ImageCombo**, and **TreeView** controls. In order to use an **ImageList** with one of these controls, you must bind a particular **ImageList** control with the second control through an appropriate property. For the **ListView** control, you must set the **Icons** and **SmallIcons** properties to **ImageList** controls. For the **TreeView**, **TabStrip**, **ImageCombo**, and **Toolbar** controls, you must set the **ImageList** property to an **ImageList** control.

At design time, you can add images using the Images tab of the ImageList Control Properties dialog box. At run time, you can add images using the **Add** method for the **ListImages** collection.

For the Windows Common Controls, you can specify an **ImageList** at design time using the Custom Properties dialog box. At run time, you can also specify an **ImageList** control using the **ImageList** property, as in the following example:

```
TreeView1.ImageList = ImageList1  ' Specify ImageList
```

Important When using the **ImageList** control with a Windows Common Control, insert all of the images you will require, in the order you desire, into the **ImageList** before binding it to the second control. Once the **ImageList** is bound to a second control, you cannot delete images, and you cannot insert images into the middle of the **ListImages** collection. However, you can add images to the end of the collection.

Once you associate an **ImageList** with a Windows Common Control, you can use the value of either the **Index** or **Key** property to refer to a **ListImage** object in a procedure. The following example sets the **Image** property of a **TreeView** control's third **Node** object to the first **ListImage** object in an **ImageList** control:

```
' Use the value of the Index property of ImageList1.
TreeView1.Nodes(3).Image = 1
' Or use the value of the Key property.
TreeView1.Nodes(3).Image = "image 1"   ' Assuming Key is "image 1."
```

To use the **ImageList** control with other controls (that can't be bound to the **ImageList** control), assign the **Picture** property of the second control to the **Picture** object of any image in the **ImageList** control. For example, the following code assigns

the **Picture** object of the first **ListImage** object in a **ListImages** collection to the **Picture** property of a newly created **StatusBar** panel:

```
Dim pnlX As Panel
Set pnlX = StatusBar1.Panels.Add() ' Add a new Panel object.
Set pnlX.Picture = ImageList1.ListImages(1).Picture ' Set Picture.
```

Note You must use the **Set** statement when assigning an image to a **Picture** object.

You can insert any size image into the **ImageList** control. However, the size of the image displayed by the second control depends on one factor: whether or not the second control is also a Windows Common control bound to the **ImageList** control.

When the **ImageList** control is bound to another Windows Common Control, images of different sizes can be added to the control, however the size of the image displayed in the associated Windows Common Control will be constrained to the size of the first image added to the **ImageList**. For example, if you add an image that is 16 by 16 pixels to an **ImageList** control, then bind the **ImageList** to a **TreeView** control (to be displayed with **Node** objects), all images stored in the **ImageList** control will be displayed at 16 by 16 pixels, even if they are much larger or smaller.

On the other hand, if you display images using the **Picture** object, any image stored in the **ImageList** control will be displayed at its original size, no matter how small or large.

Distribution Note The **ImageList** control is part of a group of ActiveX controls that are found in the MSCOMCTL.OCX file. To use the **ImageList** 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. For more information on how to add an ActiveX control to a project, see the *Programmer's Guide*.

Visual Studio 6.0

Visual Basic: MSChart Control

Intersection Object

See Also Example Properties Methods Events

The point at which an axis intersects an intersecting axis on a chart.

Syntax

Intersection

Visual Basic: MSChart Control

Intersection Object Example

The following example sets manual intersection position properties and displays labels with the axis.

```
Private Sub Command1_Click()
      With MSChart1.Plot.Axis(VtChAxisIdX).Intersection
      ' Set Intersection Properties.
      .Auto = False   ' Set positioning to manual.
      .Point = 20      ' Set intersection with the Y
                     ' Axis to 20.
      .LabelsInsidePlot = True   ' Display Labels
                                 ' with Axis not at
                                 ' the base.
   End With
End Sub
```