Add a Toolbar To Your App

Use the Visual Basic ToolBar control to give users easy access to your application's features.

by Chris Barlow

WHAT YOU NEED
VB5 or VB6

You probably haven't seen a recent professional Windows application without a toolbar. Toolbars enable users to quickly access your application's features without navigating menus. If you want your Visual Basic app to look and feel like professional applications, you need to add a toolbar. Fortunately, VB's ToolBar control is easy to use. In this column, I'll show you how to create a simple text editor and add a toolbar to it.

First, start a new Visual Basic Standard EXE project and click on the Components menu item on the Project menu. This ensures that the Common Dialog, Rich Textbox, and Common Controls are selected (see Figure 1). (The controls will have slightly different names if you're using VB5 instead of VB6.) Draw a RichTextBox and CommonDialog control on the form and use the Menu Editor to create a standard File menu with New, Open, Save, Print, and Exit menu items (see Figure 2).

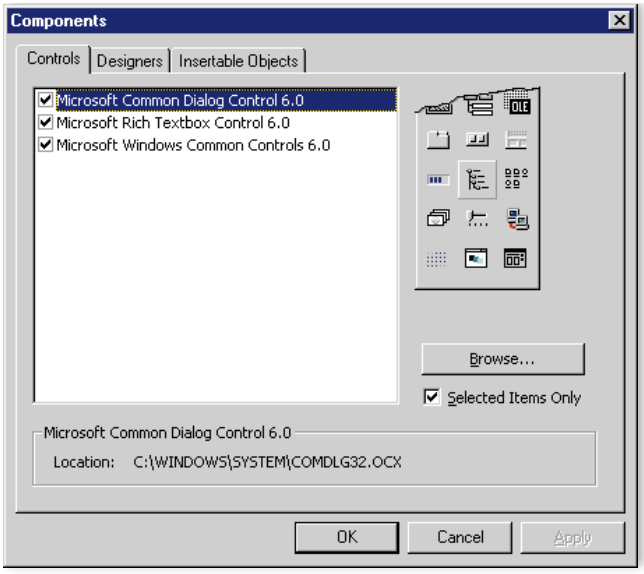

Figure 1 Select the Controls. You can display this dialog by right-clicking on the control toolbox or using the Ctrl-T shortcut key. Uncheck the "Selected Items Only" checkbox to see all the controls registered on your system.

Creating menus can be time-consuming because, even with the VB6 menu editor, you have to type all the menu properties for each form. To save time, you can make a shortcut for creating menus. First, create a form with standard Windows menus including File, Edit, and Help menus. Add this code to implement the standard New, Open, and Save menu items, then save both the FRM and FRX files into the Visual Basic\Template\Forms folder:

```
Private Sub mnuNew_Click()
RichTextBox1.Text = ""
End Sub

Private Sub mnuOpen_Click()
CommonDialog1.ShowOpen
RichTextBox1.LoadFile (CommonDialog1.FileName)
End Sub

Private Sub mnuSave_Click()
CommonDialog1.ShowSave
RichTextBox1.SaveFile (CommonDialog1.FileName)
End Sub
```

Anytime you need a form with standard menus, use the Add Form menu item on the Project menu to choose your form templates, including your new menu form. Next, add this code to enable the Print menu:

```
Private Sub mnuPrint_Click()
On Error Resume Next
CommonDialog1.Flags = cdlPDReturnDC + _
   cdlPDNoPageNums
```

```
CommonDialog1.CancelError = True
If RichTextBox1.SelLength = 0 Then
   CommonDialog1.Flags = CommonDialog1.Flags + _
      cdlPDAllPages
Else
   CommonDialog1.Flags = CommonDialog1.Flags + _
      cdlPDSelection
End If
CommonDialog1.ShowPrinter
If Err = 0 Then
   Printer.Print ""
   RichTextBox1.SelPrint Printer.hDC
   Printer.EndDoc
End If
End Sub
```

Notice that the fourth line sets the CancelError property to True so the CommonDialog control returns an error if the user clicks on the Cancel button. Then the 12th line prints only if no error is set. Now that you have a simple text editor, you can add a toolbar.

More Complicated, But Worth It

Because the ToolBar control features a hierarchical object model, it's more complicated to use than the RichTextBox and CommonDialog controls. The ToolBar control contains a Buttons collection of Button objects with their own properties and methods. To make everything a bit more complex, the images on the button faces aren't contained within the ToolBar control, but within the ListImages collection of an ImageList control.

I'll go through each step in detail to help you add a toolbar to your app. (For a short list of the steps, see the sidebar, "How to Add a Toolbar to Your Application.") Draw an ImageList control on your form, right-click on the control to display the Properties dialog, go to the Images tab, and click on the Insert Picture button. If you look in the Bitmaps\TlBr_W95 folder located under the folder where VB is installed, you'll find a good selection of bitmaps for the toolbar buttons. Insert pictures for New, Open, Save, Print, Find, Left, Center, and Right. Go to the Colors tab and change the BackColor property to the system color "Menu Bar" and the MaskColor property to the system color "Button Face." If you don't adjust the colors, you'll find the images are dithered when they appear on the toolbar buttons, and they'll be difficult to see on the button faces. You can use the Add method of the ListImages collection to add other images to the ImageList control at run time.

Now that you have a collection of images for your toolbar, add the ToolBar control to your form and right-click on the control to display its Properties dialog. On the General tab, change the ImageList property to bind the toolbar to your ImageList control. Add all the images you need to your ImageList control before you bind it to the toolbar, because you can't make changes to a bound ImageList control.

Each toolbar has a collection of Button objects. You can add Button objects to the toolbar at run time with the Add method of the ToolBar control's Buttons collection. To add a button at design time, click on Insert Button on the Buttons tab of the ToolBar control's Properties dialog. Button objects can contain an image, a caption, or both. You probably won't need to design a toolbar with both images and captions because each Button object has a ToolTipText property. Set the Key property of each Button object so you can identify which button the user has clicked on.

Insert five buttons on your toolbar, set their Key and ToolTipText properties to New, Open, Save, Print, and Find, respectively, and bind them to images one through five of the ImageList control. Click on the Apply button to see your new toolbar.

You can design a better toolbar by using the Button object's Style property. The default style is a normal button, just like the buttons now on your toolbar. To separate the Print button slightly from the New, Open, and Save buttons, insert another button with a Style property of "tbrSeparator."

On the Buttons tab, move back to the Save button by changing the index to 3, and click on Insert Button. Change the Style property to "tbrSeparator" and insert separator buttons between the Print and Find buttons and after the Find button.

Buttons can also be part of a button group: a group of buttons with a "tbrButtonGroup" style surrounded by buttons with a "tbrSeparator" style. You can press only one group button at a time. For example, if the text in your RichTextBox control can be only left-

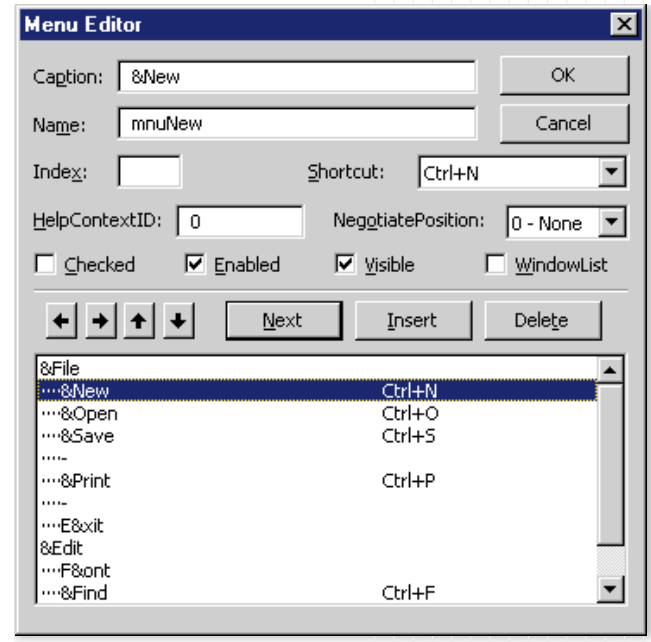

Figure 2 Create a File Menu. This menu editor has changed little since VB1. Save yourself some time and put your common forms in your Template folder so you don't have to create a menu from scratch for each project.

Figure 3 The Finished Project. Notice the placement of the buttons and their images on the completed toolbar. You've added New, Open, Save, Print, and Find buttons, as well as buttons to left-justify, center, or right-justify your text.

aligned, centered, or right-aligned, then users can press only one of the Left, Center, or Right buttons. Add three more buttons with Key and ToolTipText properties of Left, Center, and Right to your toolbar and set their Style properties to tbrButtonGroup.

Buttons can also have a Style property of "tbrPlaceholder" that enables you to add other controls to the toolbar. Add a combo box that enables the user to set the size of the font. Add two more buttons to your toolbar—one with a "tbrSeparator" style and the other with a "tbrPlaceholder" style. Set the Key property of the last button to "combo1," and the width property to 1,000. Then draw a ComboBox control on the toolbar. You can add code now that your toolbar design is complete (see Figure 3).

One of the cool things you get with the ToolBar control is the ability to let the user customize the toolbar by reordering and removing buttons.

The Code Behind the Bar

Add code to initialize the toolbar's combo box with font size data, and make sure it's located on the placeholder button of the toolbar. In the Form_Load event, add this code:

```
Private Sub Form_Load()
'Initialize the combo box
Show
With Combo1
   .Width = _
      Toolbar1.Buttons("combo1").Width
   .Left = _
      Toolbar1.Buttons("combo1").Left
   .Top = _
      Toolbar1.Buttons("combo1").Top
   .AddItem "10"
   .AddItem "12"
   .AddItem "14"
   .AddItem "16"
   .ListIndex = 0
   .ZOrder
End With
End Sub
```

You need to copy this code to the Form_Resize event so the combo box always stays in the proper place on the toolbar:

```
Private Sub Form_Resize()
With Combo1
   .Width = _
      Toolbar1.Buttons("combo1").Width
   .Left = _
      Toolbar1.Buttons("combo1").Left
   .Top = _
      Toolbar1.Buttons("combo1").Top
End With
End Sub
```

It's easy to handle the toolbar clicks because you already have menu items for most of these functions, which include New, Save, and Open. You simply call these menu functions. The toolbar ButtonClick event passes the Button object the user clicked on, so you can write a Select statement based on the Key property of the button:

```
Private Sub Toolbar1_ButtonClick(ByVal _
   Button As Button)
Select Case Button.Key
   Case "New": mnuNew_Click
   Case "Open": mnuOpen_Click
   Case "Save": mnuSave_Click
   Case "Print": mnuPrint_Click
   Case "Find": mnuFind_Click
   Case "Left": _
      RichTextBox1.SelAlignment = _
      rtfLeft
   Case "Center": _
      RichTextBox1.SelAlignment = _
      rtfCenter
   Case "Right": _
      RichTextBox1.SelAlignment = _
      rtfRight
End Select
End Sub
```

The only new code—the Case "Left," Case "Center," and Case "Right" statements—sets the alignment property of the RichTextBox control based on which button the user clicked on. This changes the alignment of the selected paragraphs or, if no text is selected, the current paragraph. Setting each button's Value property within the RichTextBox control's SelChange event sets these alignment buttons to display the text's actual alignment as you move through the text. Notice how the "Case Else" statement "unpresses" all buttons in the group if the alignment is other than left, centered, or right:

```
Private Sub RichTextBox1_SelChange()
Select Case RichTextBox1.SelAlignment
   Case rtfLeft
      Toolbar1.Buttons("Left").Value = _
         tbrPressed
   Case rtfCenter
      Toolbar1.Buttons("Center").Value = _
         tbrPressed
   Case rtfRight
      Toolbar1.Buttons("Right").Value = _
         tbrPressed
   Case Else
      Toolbar1.Buttons("Left").Value = _
```

How to Add a Toolbar to Your Application

1. Add an ImageList control to your form.
2. Insert pictures for the button faces into the ListImages collection of the ImageList control.
3. Add a ToolBar control to your form.
4. Set the ToolBar ImageList property to bind the toolbar to your ImageList control.
5. Add Button objects to the toolbar.
6. Set each button's properties to bind the button image to the proper image.
7. Write code to handle toolbar button clicks.

```
        tbrUnpressed
    Toolbar1.Buttons("Center").Value = _
        tbrUnpressed
    Toolbar1.Buttons("Right").Value = _
        tbrUnpressed
End Select
Combo1.Text = RichTextBox1.SelFontSize
End Sub
```

The last line of code in the RichTextBox1_SelChange( ) procedure uses the SelFontSize property to display the font size as the cursor moves through the text. It's easy to add code to change the font size of the selected text when the user makes a selection from the combo box on your toolbar. Simply set the SelFontSize property of the RichTextBox control to the default value of the combo box and return the focus to the RichTextBox control:

```
Private Sub Combo1_Click()
RichTextBox1.SelFontSize = Combo1
RichTextBox1.SetFocus
End Sub
```

You now have a fully functional toolbar for your application (download Listing 1 from The Development Exchange; see the Download Free Code box for details). One of the cool things you get with the ToolBar control is the ability to enable the user to customize the toolbar by reordering and removing buttons. If you set the AllowCustomize property of the toolbar to True, the user can double-click on the toolbar at run time to bring up a customize dialog where he or she can modify the toolbar you set up at design time. You can even add code to the toolbar Change event to use the SaveToolBar method to store the current toolbar settings in the Registry, then use the RestoreToolBar method to reload it the next time the user runs your application. VBPJ

About the Author

Chris Barlow, a recognized expert in the Internet, Web, messaging, and applications development fields, is a frequent speaker at VBITS, Tech•Ed, and DevDays and has been featured in two Microsoft videos. Chris holds degrees from Harvard Business School and Dartmouth College. Reach him at Chris@VBExpert.com or on the Web at http://www.VBExpert.com.

DOWNLOAD FREE CODE

Download the code for this issue of **VBPJ free** from http://www.vbpj.com.

To get the free code for this entire issue, click on Locator+, the rightmost option on the menu bar at the top of the VBPJ home page, and type **VBPJ0299** into the box. (You first need to register, for free, on DevX.) The free code for this article includes all code listings, plus the toolbar application.

To get the code for this article only, available to DevX Premier Club members, type **VBPJ0299GS** into the Locator+ field.