Add a Toolbar to Your App

Click & Retrieve Source CODE!

Use VB4's new ToolBar control to give the user easy access to the features of your application.

by Chris Barlow

You probably haven't seen a recent professional Windows application without a toolbar. Toolbars give users quick access to your application features without forcing them to navigate menus. If you want your Visual Basic application to look and feel like these professional applications, you need to use this control. Fortunately, the new ToolBar control included with Visual Basic 4.0 is easy to use.

First let's get to a common starting point. You can follow the code examples in this column to add a toolbar to any of your existing applications. I'll build upon the text editor example from last month's column. If you want to use the same code and you haven't downloaded the source code for the text editor from *VBPJ's* online sites, just draw a RichTextBox and CommonDialog control on an empty form and create a standard File menu with New, Save, and Exit menu items. For more information on the new RichTextBox and CommonDialog controls in Visual Basic 4.0, refer to my last column.

Creating menus can be time consuming because you have to type all the menu properties for each form. How about a shortcut? If you have another form with a standard Windows menu, open it with Notepad and look at the source code for the form. You'll see a set of code starting with "Begin VB.Menu…"(see Listing 1). Copy this code to the clipboard, then open your new FRM file with Notepad and paste in this code. I've saved this code in a text file so I can paste it into my FRM file when I need a standard menu. It saves a lot of typing! Be sure to save the file as ASCII text so Visual Basic can load it.

MORE COMPLICATED, BUT WORTH IT

This control is a bit more complicated than the RichTextBox and CommonDialog controls you've learned about so far. Visual Basic 4.0 is designed to use OLE Automation throughout, so you should become familiar with using these Visual Basic objects with their own properties, methods, and collections. One of the reasons the ToolBar control is more complicated than the other controls is that the toolbar buttons are a collection of Button objects with their own properties and methods. And to make it a bit more complex, the images on the button faces are contained with the ListImages collection of an ImageList control (see Figure 1).

To add a toolbar to your application, you need to perform several steps. I always recommend you start writing a procedure by listing the steps in pseudocode:

```
1. Add an ImageList control to your form.
2. Insert pictures for the button faces to 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.
```

I'll go through each of these steps in detail. Start by drawing 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 you'll find a good selection of bitmaps for the toolbar buttons. Insert pictures for New, Open, Save, Print, Find, Left, Center, and Right. Now go to the Colors tab and change the BackColor

VB4

```
Begin VB.Menu mnuFile
    Caption  =  "&File"
    Begin VB.Menu mnuNew
       Caption  =  "&New"
    End
    Begin VB.Menu mnuOpen
       Caption  =  "&Open"
    End
    Begin VB.Menu mnuSave
       Caption  =  "&Save"
    End
    Begin VB.Menu Sep1
       Caption  =  "-"
    End
    Begin VB.Menu mnuFont
       Caption  =  "&Font"
    End
    Begin VB.Menu mnuPrint
       Caption  =  "&Print"
    End
    Begin VB.Menu Sep2
       Caption  =  "-"
    End
    Begin VB.Menu _
       mnuExit
       Caption  =  _
          "E&xit"
    End
End
Begin VB.Menu mnuEdit
    Caption  =  "&Edit"
    Begin VB.Menu _
       mnuFind
       Caption  =  _
          "&Find"
    End
    Begin VB.Menu _
       mnuNext
       Caption  =  _
          "Find &Next"
    End
End
```

LISTING 1 ***How About a Shortcut?*** *Creating menus can be time consuming because you have to type all the menu properties for each form. If you have another form with a standard Windows menu, open it with Notepad and copy the source code to the clipboard. Then open your new FRM file with Notepad and paste in this code. Be sure to save the file as ASCII text so Visual Basic can load it.*

Chris Barlow is president and CEO of SunOpTech, a developer of manufacturing decision-support applications including the ObjectBank and the ObjectJob Systems, where he and Ken Henderson hold a software patent related to decentralized distributed asynchronous object-oriented systems. Chris holds degrees from Harvard Business School and Dartmouth College where he worked with Drs. Kemeny and Kurtz on the BASIC language. Reach Chris on the Internet at ChrisB@SunOpTech.com or through SunOpTech's World Wide Web server at www.SunOpTech.com.

©1991–1996 Fawcette Technical Publications

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 that the images are dithered when they appear on the toolbar buttons and they'll be difficult to see on the button faces. At run time you can use the Add method of the ListImages collection to add other images to the ImageList control.

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 property dialog. On the General tab change the ImageList property to bind the toolbar to your ImageList control. Be sure you have added all the images that you will need to your ImageList control before you bind it to the toolbar because you cannot 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 Buttons collection. At design time, on the Buttons tab of the ToolBar control's property dialog, click on Insert Button to add a button. Button objects can contain either an image or a caption, or both. Because each Button object has a ToolTipText property, however, you probably won't need to design a toolbar with both images and captions. You'll want to 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," 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 Style property of the Button object. 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 "separator."

On the Buttons tab, move back to the Save button by changing the index to three and click on the Insert Button button. Then change the Style property to "separator." Insert another separator button between the Print and Find buttons and after the Find button.

Buttons can also be part of a button group where only one button at a time can be pressed. For example, if the text in our RichTextBox

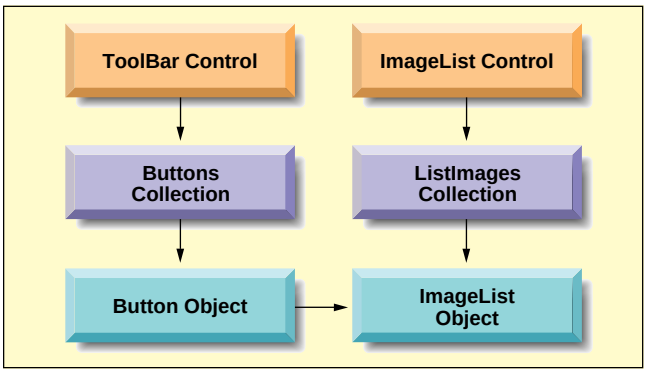

FIGURE 1 ***Objects and Collections.*** *You'll need to know how these objects relate to each other to develop a working toolbar. The toolbar buttons are a collection of Button objects with their own properties and methods. And to make it a bit more complex, the images on the button faces are contained with the ListImages collection of an ImageList control.*

VB4

```
Option Explicit
Public sFind As String

Private Sub Combo1_Change()
RichTextBox1.SelFontSize = Combo1
RichTextBox1.SetFocus
End Sub

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

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

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

Private Sub mnuExit_Click()
Unload Me
End
End Sub

Private Sub mnuFind_Click()
sFind = InputBox("Find what?", , sFind)
RichTextBox1.Find sFind
End Sub

Private Sub mnuFont_Click()
CommonDialog1.Flags = cdlCFBoth + cdlCFEffects
CommonDialog1.ShowFont
With RichTextBox1
  .SelFontName = CommonDialog1.FontName
  .SelFontSize = CommonDialog1.FontSize
  .SelBold = CommonDialog1.FontBold
  .SelItalic = CommonDialog1.FontItalic
  .SelStrikethru = CommonDialog1.FontStrikethru
  .SelUnderline = CommonDialog1.FontUnderline
End With
End Sub

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

Private Sub mnuNext_Click()
RichTextBox1.SelStart = RichTextBox1.SelStart + _
  RichTextBox1.SelLength + 1
RichTextBox1.Find sFind, , Len(RichTextBox1)
End Sub

Private Sub mnuOpen_Click()
CommonDialog1.ShowOpen
```

CONTINUED ON NEXT PAGE.

LISTING 2 *Complete Toolbar Code. Use this code to add a toolbar to your application. The project files, contained in a file called TOOLBAR.ZIP, are also available on the VBCD, in the Magazine Library (#3) of the* VBPJ *Forum on CompuServe (GO WINDX with WinCIM), the* VBPJ *Development Exchange World Wide Web site (http://www.windx.com), or the* VBPJ *site on The Microsoft Network (GO WINDX).*

©1991–1996 Fawcette Technical Publications

CONTINUED FROM PREVIOUS PAGE.

```
RichTextBox1.LoadFile (CommonDialog1.filename)
End Sub

Private Sub mnuPrint_Click()
CommonDialog1.Flags = cdlPDReturnDC + cdlPDNoPageNums
If RichTextBox1.SelLength = 0 Then
  CommonDialog1.Flags = CommonDialog1.Flags + _
     cdlPDAllPages
Else
  CommonDialog1.Flags = CommonDialog1.Flags + _
     cdlPDSelection
End If
CommonDialog1.ShowPrinter
RichTextBox1.SelPrint CommonDialog1.hDC
End Sub

Private Sub mnuSave_Click()
CommonDialog1.ShowSave
RichTextBox1.SaveFile (CommonDialog1.filename)
End Sub

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

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
```

control can be only left-aligned, centered, or right-aligned, then only one of the Left, Center, or Right buttons should be pressed. A button group is defined as a group of buttons with a "button group" style surrounded by buttons with a "separator" style.

Add three more buttons with Key and ToolTipText properties of "Left," "Center," and "Right" to your toolbar and set their style to "button group."

Buttons can also have a style property of "placeholder" that allows you to add other controls to the toolbar. Add a combo box that will allow the user to set the size of the font. Add two more buttons to your toolbar—one with a "separator" style and the other with a "placeholder" style. Set the Key property of the last button to "combo1," and the width property to 1000. Then draw a ComboBox control on the toolbar.

You'll need to add one more invisible button to your toolbar with a style of "default." A bug in Visual Basic prevents the toolbar from wrapping properly if the last button on the toolbar has a placeholder style. Now that your toolbar design is complete (see Figure 2), it is time to add the code.

THE CODE BEHIND THE BAR

Because you have a combo box on your toolbar, you'll need to add code to initialize the combo box and to make sure it is

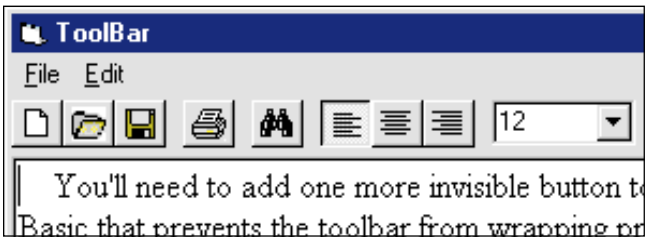

FIGURE 2 ***Your New Toolbar.*** *The toolbar you've created features a combo box control for font selection, in addition to buttons for New, Open, Save, Print, Find, and text alignment.*

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 "tracking" code to the Form_Resize event so the combo box always stays in the proper place:

```
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
```

Handling the toolbar clicks will be fairly easy because you already have menu items for most of these functions. The toolbar ButtonClick event passes the button object that 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
```


©1991–1996 Fawcette Technical Publications

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 will change the alignment of the selected paragraphs or, if no text is selected, the current paragraph. You can set these alignment buttons to display the actual alignment of the text as you move through the text by setting each button's Value property within the SelChange event of the RichTextBox control. 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 = _
      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. Now you add code to change the font size of the selected text when the user makes a selection from the combo box on your toolbar. This is easy—just 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
```

Now you have a fully functional toolbar for your application (see Listing 2). One of the cool things you get for "free" with the ToolBar control is the ability to allow the user to customize the toolbar by reordering and removing buttons. If you set the Customize property of the toolbar to True, the user can double-click on the toolbar at run time to bring up a customize dialog by which he or she can modify the toolbar you set up at design time. You can even use the SaveToolBar method to store the current toolbar settings in the Registry and use the RestoreToolBar method to reload it the next time the user runs your application.

The code discussed in this column, contained in a file called TOOLBAR.ZIP, is also available on the VBCD, in the Magazine Library (#3) of the *VBPJ* Forum on CompuServe (GO WINDX with WinCIM), the *VBPJ* Development Exchange World Wide Web site (http://www.windx.com), or the *VBPJ* site on The Microsoft Network (GO WINDX). ×

 ©1991–1996 Fawcette Technical Publications