C# How-to 7: Use the Windows Forms Toolbar control

The toolbar control supplied with Visual Studio .NET is very similar in use to the status bar, with the added proviso that it is necessary to link it to an ImageList control to provide the toolbar images. Unfortunately the toolbar itself is extremely basic in operation; regrettably it won't let you duplicate the fancy toolbar from Visual Studio .NET itself. No doubt third-party controls will fill that gap soon enough.

Important updates, January 1 2002:

(1) A bug in the code for Simple Editor has been corrected to resize the rich text box correctly. Unfortunately this revealed problem (2) below. The downloadable code contains the corrected version.

(2) There appears to be a bug in Visual Studio which causes the height of the toolbar control to be returned incorrectly. On my system, the height is returned in code as 22 pixels, but in the IDE it is correctly given in the Properties window as 25 pixels. This causes the rich text box to be resized incorrectly so that in a document which is longer than can be accommodated in one window, the bottom arrow on the vertical scrollbar is partially obscured. Hopefully this will be fixed in the release version of VS.NET.

1. Design time

Add the control to a form from the toolbox, and it will dock automatically to the top of the window. You might want to change the Appearance property to Flat, which gives you flat buttons which are drawn with a 3D border when the mouse is over them. You can ignore the ButtonSize property because it will change anyway depending on the size of the images in the buttons, the button style, and whether there is a text label for the button.

The next thing is to add an ImageList control. The only thing you have to do here is to add one or more images, depending on how many buttons you will have. One nice thing about this control compared to the previous version in VS6 is that you can remove or add images even while it is bound to another control.

You add images to the ImageList by clicking the button next to Images (Collection) in the Properties window. There aren't any properties you can set for these images, other than to add, remove, or reorder them.

Once you have added the images, the only other properties of the ImageList you might want to set are the color depth (by default it is 8-bit but you can change that if you are using more colourful images) and the Transparent color. This colour won't be drawn when the button images are rendered which is useful if you are using images in a format which doesn't support transparency (e.g. the .BMP format). It does mean though that only one transparent colour can be set, so the colour has to be the same for all images. If you are using the bitmaps supplied with VS.NET (the same old collection it has contained since VB3!) then the transparent colour is always silver gray. This is #C0C0C0 in RGB format, or in .NET terms, System.Drawing.Color.Silver. You can set this from the dropdown list next to the Transparency colour in the Properties window.

Finally, set the toolbar's ImageList property to point to the new ImageList control.

2. Adding buttons

Now you can add buttons to the toolbar by selecting the toolbar and clicking the button next to Buttons (Collection) in the Properties window. In the dialog box you can, as usual, add, remove, or re-order buttons. The properties most comonly used will probably be:

ImageIndex: points to the index of the required image in the ImageList bound to the toolbar control

Style: sets the button to be one of the following styles: PushButton: a conventional button

ToggleButton: a button which toggles between a down (pushed) state and an up state when clicked

DropDownButton: a button with an attached dropdown menu

Separator: a vertical line used to separate groups of other buttons (if the toolbar's Appearance property is set to Flat, otherwise the separator appears as a space in between the raised buttons)

Text: the label which appears under the button (or to its right if you wish); set the TextAlign property of the toolbar (not the button) to determine where the label goes

TooltipText: the text of the tooltip associated with the button

If the button is a ToggleButton, two other properties are available:

Pushed: the button is in its down (pushed) state

PartialPush: if this is True, then the entire button is hazed over and no longer acts like a toggle button but like a conventional push button

If the button is a DropDownButton, a drop-down menu can be attached to it. To do this, you have to add a Menu or ContextMenu control to the form and edit the menu in the usual way. You can then set the DropDownMenu property of the button to point to the new menu. Now, according to the MSDN documentation, a small arrow to the right of the button and pointing downwards will only be shown if the toolbar's DropDownArrows property is True. The user then needs to click the arrow to show the menu. In fact, there appears to be a bug in the beta2 of VS.NET; if DropDownArrows is False, the arrow isn't shown at design time but is still shown at run time.

One other point to note is that, as with the status bar panel objects, the buttons are added to the code as separate objects which are then added to the toolbar when it is initialised. This means that you can access the buttons directly in your code rather than having to go through the toolbar object.

3. Final design tweaks

There are a few other properties of the toolbar which are worth a quick look but it's easier to experiment with them than document them here. These are:

BorderStyle

Cursor

Divider

ContextMenu

ShowTooltips

Wrappable

4. Adding code

As with the status bar, the toolbar can generate a lot of events but the most useful ones are the ButtonClick and the ButtonDropDown events. The first occurs when a button is clicked, and the second when a DropDownButton (or its arrow) is clicked.

Both events receive a ToolbarButtonClickEventArgs parameter. This has a Button property which returns the button which was clicked; you can obtain the index of that button in the collection by using the IndexOf method of the Buttons collection in the toolbar.

An example makes this clearer. To add an event handler for the toolbar buttons, double-click the ButtonClick event in the Properties window. The .NET designer will add the following code:

```
private void toolBar1_ButtonClick(object sender, System.Windows.Forms.ToolBarButtonClickEventArgs e)
{

}
```

Now we can add a switch statement to select between the various possible buttons, so if we have (say) a New, Open, and Save buttons on the toolbar in that order, the code would look something like this:

```
private void toolBar1_ButtonClick(object sender, System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
switch (toolBar1.Buttons.IndexOf(e.Button)) {
case 0:
// create a new file
break;
case 1:
// open a file
break;
case 2:
// save the file
break;
}
}
```

This is all pretty straightforward. You would also need to add event handlers for your dropdown buttons, if you have any, but other than that the toolbar is now complete.

5. Example code

To test the toolbar control out, I will add one to Simple Editor, the test bed for this series of tutorials and how-tos. Because it's straightforward I won't go through the code here but you can download it from the link below to see how it works - it's all fully documented in the code. The code has also been tidied up a little since the previous version.

Download code for Simple Editor plus toolbar support (68K)