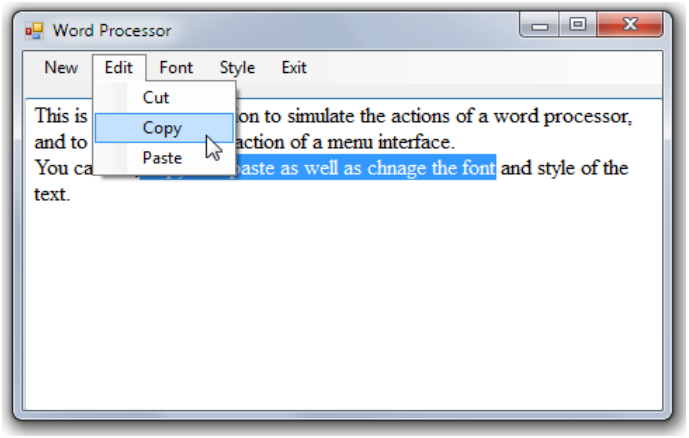
Project 5 – Write on!

The user interface in any application is critical for ease of use. One of the most widely used interfaces is to offer the user a menu of choices. In this project we will prepare a simple menu driven word processor.



Task: Enter text, and be able to change its font, copy, cut and paste, or clear it.

Objects: Text box and menu bar.

Events: Mouse click on menu items to carry out set tasks.

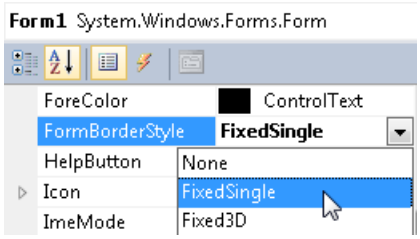
1. In VB begin a new project.

2. a Resize the form to about one third screen size.

   b Set the form's *Text* property to *Word Processor*.

   c Change the form's *FormBorderStyle* to *FixedSingle.*

   By making the border fixed the user will not be able to re-size it when the program is running. In this way we can control the size of the word processor window on screen.



3. a Add a text box to the form and name it *WPText.* TextBox

   b Set the *MultiLine* property for the text box to *True*. (This will enable word wrap.)

   c Re-size the text box to nearly fill the form. There should just be room for the menu bar at the top. (We will add this shortly.)

   d Set the *Font* for the text box to *Times New Roman, Regular* style*, 12pt* size.

4. At this point save the project to a suitable folder and run it.

   You will be able to type in the text box. Note how the word wrap works. As writing reaches the end of the line, whole words are "wrapped" onto the next line.

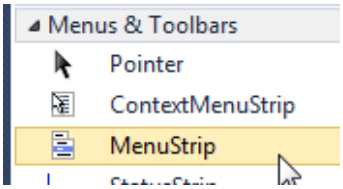
   Note also what occurs as typing spills off the bottom of the text box.

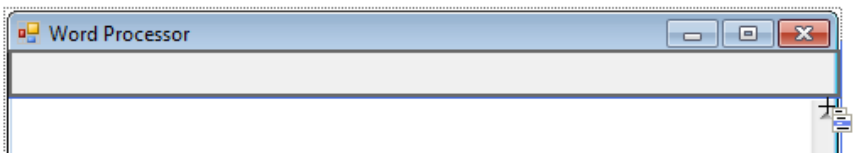
   To fix this, return to VB, stop the application, and change the *ScrollBars* property to *Vertical*. This will place a vertical scroll bar on the side of the text box for when the user types outside of the visible section. (Why do we not need a horizontal scroll bar?)

   Note also that the mouse pointer changes to an I beam in the text box. (The cursor shape can be set in the properties inspector for the text box.)

5. We are now ready to add our menu bar.

   a From the toolbox select *MenuStrip* from the *Menus & Toolbars* group.





Click and drag across the top of the text box.

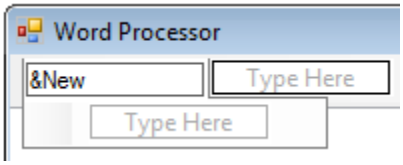
 © Kevin Savage 2011

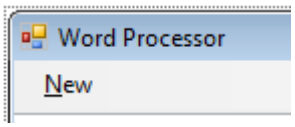
We will place our menu items in this strip.

(The reason we set *FormBorderStyle* to *FixedSingle* in step 2c above is so that the user could not resize the window horizontally, and hence leave the edge of the menu strip hanging. This is part of a process known as *user proofing* a program.)

b Next we will add the menu commands.

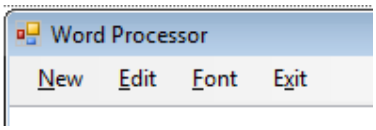
Click once on the menu strip and type *&New* where is says *Type Here*.





c Repeat for *&Edit, &Font* and *E&xit.*

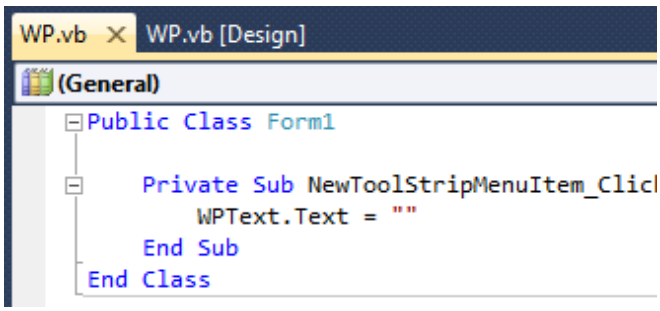
d Re-save the form and run it to see how the menu looks (it will not work yet).



6. We will now add code to the *New* and *Exit* commands.

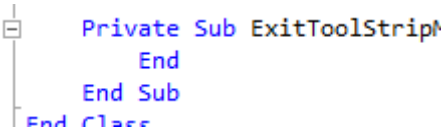
a On the menu strip double click on the *&New* menu item to go to the code editor.

At the cursor enter *WPText.Text = " "*



This will clear all writing from the text box, that is it will put nothing (" ") in place of what is there.

b Return to the menu strip, double click on the *E&xit* button, and enter *End* at the cursor:



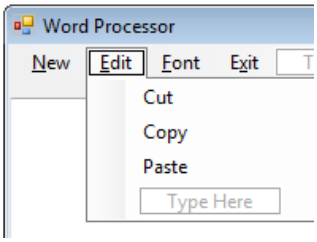
c Again save and then run the project. Note how you can now type, and then use the menu to clear text and to exit.

7. We are now ready to add the *Edit* sub-menu items of cut, copy and paste. We will use the Windows clipboard to do this.

a In the menu strip click on *&Edit* and then on the *Type Here* below it.

At this sub-menu level add three commands for *Cut, Copy* and *Paste.*

b By double clicking on each item one at a time, in turn, add the following code in the appropriate places. (If you use the autocomplete option to save typing make sure the correct word is chosen.)




```
Private Sub CutToolStripMenuItem_Click(ByVal sender As Sys
    Clipboard.Clear()
    Clipboard.SetText(WPText.SelectedText)
    WPText.SelectedText = ""
End Sub

Private Sub CopyToolStripMenuItem_Click(ByVal sender As Sy
    Clipboard.Clear()
    Clipboard.SetText(WPText.SelectedText)
End Sub

Private Sub PasteToolStripMenuItem_Click(ByVal sender As S
    WPText.SelectedText = Clipboard.GetText
End Sub
```

 © Kevin Savage 2011

SelectedText refers to text selected with the mouse.

Cut and *Copy* first clear the Windows clipboard then add the selected text to it. *Cut* in addition clears the text after copying it.

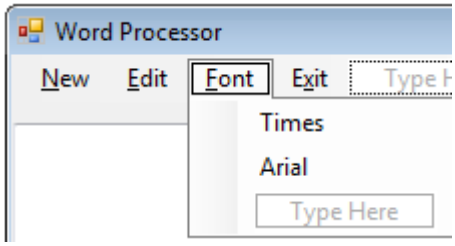
Paste gets text from the clipboard and inserts it at the cursor.

c Save and run to test the commands.

Note: at the moment cut and copy will cause an error if nothing is selected. Handling errors such as this is beyond what we will cover at this stage.

8. Now to repeat for the *&Font* menu.

a Add the *Times* and *Arial* commands as sub-menu items under *&Font*.



b Code these items as:

```
Private Sub TimesToolStripMenuItem_Click(ByVal sender As System.Object, E
    WPText.Font = New System.Drawing.Font("Times New Roman", 12)
End Sub
```

```
Private Sub ArialToolStripMenuItem_Click(ByVal sender As System.Object, E
    WPText.Font = New System.Drawing.Font("Arial", 12)
End Sub
```

c Save and run to test again.

9. Add another menu item called *Style* with three sub-menu items, *Plain, Bold* and *Italic*. Write the code for each of these.

Hint: to make the text bold use:

WPText.Font = New System.Drawing.Font(WPText.Font, FontStyle.Bold)

The font style can be regular, italic or bold.

10. Close the project saving it to a suitable folder.

Follow up

In this project we have seen how VB can interface directly with the Windows operating system by accessing the clipboard. What is not so obvious is the method we have gone about preparing our application. We have been following a process called module testing.

Module testing basically means prepare a small part of your application and then test it is working correctly before going on to the next part.

In this project we firstly created the form with a text box, saved it, and then tested that the form opened and we could write in the text box. Next we added the menu bar and again tested to see if everything was okay. Only at this stage did we then start adding code, and this we did in sections, checking each time to see the last part added worked correctly.

By building the application part by part (module by module) we ensured that any errors that might have arisen only occurred in the part most recently written. These errors could then be isolated (they must have been in a change just made) and corrected.

While such a process is not vital in a small program such as this, it can be critical in preparing larger applications. Locating errors in a complex program with many inter-related parts can be very complicated. By using module testing the occurrence of errors can be greatly reduced.

 © Kevin Savage 2011