Your First VB4 App

Click & Retrieve Source CODE!

Put "Hello World!" to shame by building a fully functional text editor in less than five minutes.

by Chris Barlow

Hello World! is the traditional first application written by C programmers—they get a real satisfaction struggling through all it takes to write and compile that first application that displays the phrase "Hello World!" on the screen.

Visual Basic 4.0 makes that kind of application trivial:

```
Start Visual Basic.
Double click on the form.
Type MsgBox "Hello World!" in the Form_Load event.
Run the program.
```

Yawn! That's not even worth showing someone. But when I sit down with someone who is just getting started with Visual Basic I say, "How you'd like to build a text editor?" I can go from the Visual Basic autoload project to a finished text editor with eight steps in less than two minutes! Now that's an impressive demonstration.

TWO MINUTES AND COUNTING

Load Visual Basic (32-bit), click on the form, and change the form's Caption property to "TextEdit." Select the Menu Editor tool and type in a standard File menu with New, Open, Save, and Exit menu items (see Figure 1). Elapsed time after Steps 1 and 2: 30 seconds.

Before Windows 95 I used a TextBox control, but the new RichTextBox control has some real advantages. The TextBox provides multiline text, scrollbars, and built-in clipboard functions. The RichTextBox control, on the other hand, includes properties that let you change the font of selected text to bold, italic, underline, strikethrough, color, and even hanging indents.

It also has LoadFile and SaveFile methods to open and save files in both the RTF format and regular ASCII text format. Another neat feature is that you load the contents of an RTF file into the RichTextBox control simply by dragging and dropping the file directly onto the control. You can also print all or part of the text in a RichTextBox control using the SelPrint method.

Step 3: Draw a RichTextBox control on the form and set the Scrollbars property to "Both." Elapsed time: 45 seconds.

Now you need a way to identify the file to open and to save. The new CommonDialog control provides a standard set of dialog boxes for operations such as opening, saving, and printing files or selecting colors and fonts. It gives your Visual Basic program access to the same dialogs used by other Windows 95 programs so you can perform all the same file functions such as copy, rename, and quickview.

The CommonDialog control is even easier to use than the CMDialog control you are used to from VB3. It has ShowOpen, ShowSave, ShowPrinter, and ShowFont methods to open the various dialogs. The trickiest part is setting the Flags property to display and return the proper information. For example, you can set the Flags property so that the Open dialog restricts the user to a single directory (cdlOFNNoChangeDir), allows the File Name list box to have multiple selections (cdlOFNAllowMultiselect), or limits the user to entering only names of existing files (dlOFNFileMustExist). All these flags are documented in the Help file, so if the dialog is not doing what you expect, search the Help file and you'll probably find out how to make it do what you want (see Table 1).

Step 4: Draw a CommonDialog control on the form and right-click on it to display the Properties dialog. Set the DefaultExt to "txt" and set the Filter to "Text Files | *.txt | All files | *.*" (see Figure 2). Elapsed time: 60 seconds.

Now that you have the form, menu, and controls ready, add some code. Click on the File menu, then on the New menu item to display the click event. Type the code to clear the RichTextBox control by setting the Text property to an empty string:

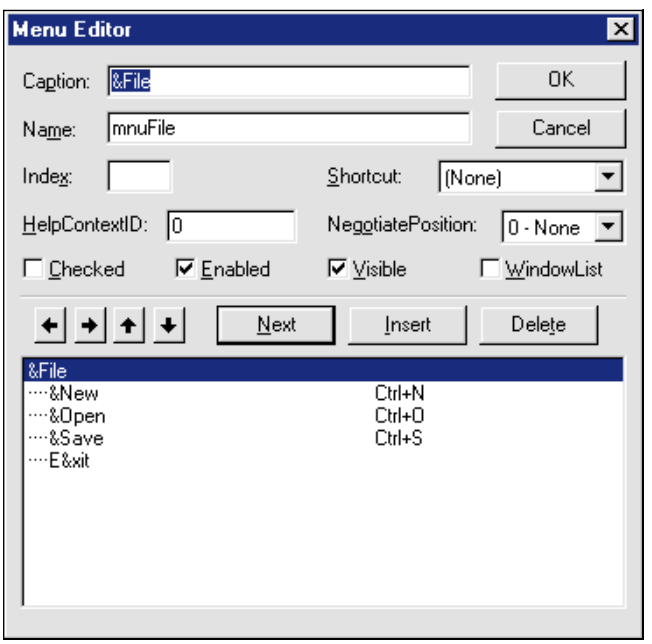


FIGURE 1 ***Get Started in 30 Seconds.*** *To start building your text editor, click on your form and change its Caption property to "TextEdit." Select the Menu Editor tool and type in a standard File menu with New, Open, Save, and Exit menu items.*

Chris Barlow is president and CEO of SunOpTech, a developer of manufacturing decision-support applications. 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.

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

Then click on the Exit menu item in the File menu and add the code to exit the program:

```
Private Sub mnuExit_Click()
Unload Me
End
End Sub
```

Elapsed time after Steps 5 and 6: 75 seconds. Now you're ready to use the CommonDialog control for the Open and Save events. Click on the File menu, then on the Open menu item and add the code to load a file into the RichTextBox. The ShowOpen method of the CommonDialog control displays the standard file-open dialog. Then the LoadFile method of the RichTextBox control loads and display the file:

```
Private Sub mnuOpen_Click()
CommonDialog1.ShowOpen
RichTextBox1.LoadFile (CommonDialog1.filename)
End Sub
```

Frequently Used Flags for the CommonDialog control Description	Constant
File Open/Save Dialog Box Flags	
Causes the Save As dialog box to generate a message box if the selected file already exists.	cdlOFNOverwritePrompt
Sets the current directory to what it was when the dialog box was invoked.	cdlOFNNoChangeDir
Causes the dialog box to display the Help button.	cdlOFNHelpButton
Allows the File Name list box to have multiple selections.	cdlOFNAllowMultiselect
User can enter only valid path names.	cdlOFNPathMustExist
User can enter only names of existing files.	cdlOFNFileMustExist
Sets the dialog box to ask if the user wants to create a file that doesn't currently exist.	cdlOFNCreatePrompt
Use Long file names (Windows 95 only.)	cdlOFNLongNames
Color Dialog Box Flags	
Entire dialog box is displayed, including the Define Custom Colors section.	cdlCCFullOpen
Dialog box displays a Help button.	cdlCCHelpButton
Fonts Dialog Box Flags	
Dialog box lists available screen and printer fonts.	cdlCFBoth
Dialog box enables strikeout, underline, and color effects.	cdlCFEffects
Printer Dialog Box Flags	
Returns or sets state of All Pages option button.	cdlPDAllPages
Returns or sets the state of the Pages option button.	cdlPDNoPageNums
Disables the Selection option button.	cdlPDNoSelection
Returns or sets the state of the Pages option button.	cdlPDPageNums
Displays the Print Setup dialog box rather than the Print dialog box.	cdlPDPrintSetup
Returns a device context for the printer selection value returned in the hDC property of the dialog box.	cdlPDReturnDC
Returns default printer name.	cdlPDReturnDefault

TABLE 1 ***Grand Old Flags?*** *The trickiest part about using the new CommonDialog control is setting the Flags property to display and return the proper information. Here's a list of Flags that I use frequently.*

By the way, I've gotten used to the Visual Basic code window in the Full Module View so I can see all the procedures for the form or module. After using VB3 for so many years, I never thought I would make the change. But now I like the ease of copying code from one procedure to another. Try it yourself: select the Options menu item from the Tools menu, click on the Editor tab, and select Full Module View (see Figure 3). If you do this the next step will be easy. Just copy the code above to the mnuSave_Click event and change the ShowOpen method to ShowSave and the LoadFile method to SaveFile:

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

Steps 7 and 8 complete with a total elapsed time of 90 seconds. At this point we've got a fully functioning text editor that:

- Supports Windows 95 Open and Save As dialogs.

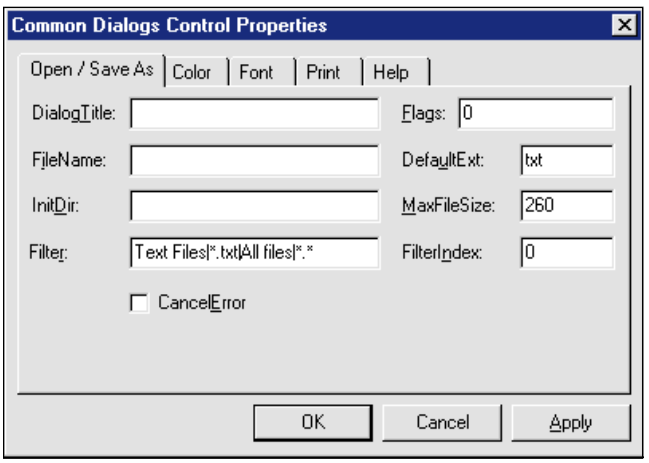


FIGURE 2 ***Identify the File.*** *The new CommonDialog control provides a standard set of dialog boxes for operations such as opening, saving, and printing files or selecting colors and fonts.*

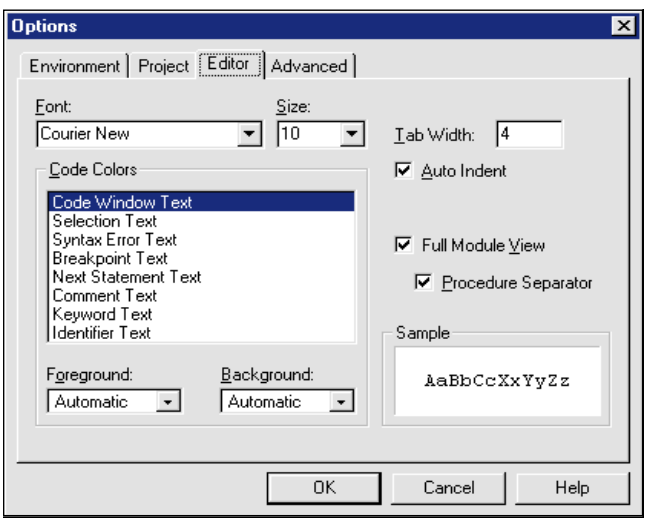


FIGURE 3 ***Want to See It All?*** *If you select Full Module View on the Editor tab in the Options dialog, you can see all the procedures for the form or module. This gives you the ability to easily copy code from one procedure to another.*

• Supports clipboard cut (Ctrl-X), copy (Ctrl-C), and paste (Ctrl-V) functions.
• Supports one level of Undo (Ctrl-Z).
• Supports file-open with drag-and-drop from Explorer, Word for Windows, etc.

Try to run your application, use the Windows Explorer to drag a text or RTF file and drop it on your form, make some changes, then save it using the Save menu item in the File menu.

DON'T STOP NOW...

You've completed your text editor, but the RichTextBox control gives you some more easy-to-use capabilites. Because you've still got some time, try expanding it a bit to allow printing and font changes and even a simple Find function. Go back to the Menu Editor tool, insert a separator below the Save menu item, and add menu items for Font and Print. Then click on the Font menu item and add code to use the CommonDialog control to display a dialog to change the Font including the name, size, color, bold, italic, strikethrough, and underline properties. You'll need to use the Flags property to display both screen and printer fonts (cdlCFBoth) and to show the full set of font properties (cdlCFEffects):

```
Private Sub mnuFont_Click()
CommonDialog1.Flags = cdlCFBoth + _
   cdlCFEffects
CommonDialog1.ShowFont
```

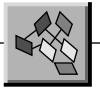
Then you need to set the properties of your RichTextBox control to match the properties of the CommonDialog control. Use the new With/End With statement to make the code easier to read and more efficient:

```
With RichTextBox1
   .SelFontName = _
      CommonDialog1.FontName
   .SelFontSize = _
      CommonDialog1.FontSize
   .SelBold = CommonDialog1.FontBold
   .SelItalic = _
      CommonDialog1.FontItalic
   .SelStrikethru = _
      CommonDialog1.FontStrikethru
   .SelUnderline = _
      CommonDialog1.FontUnderline
End With
End Sub
```

Notice that these statements will change the properties of the selected text within the RichTextBox control. If no text is selected, however, all text will be set to these properties.

Printing is almost as easy. The only trick is to recognize that the RichTextBox control does not print directly to the printer, but sends the formatted text to the device context of a device that can print text. I don't expect you to fully understand things like "hDC: handle to a device context." You just need to know that the Printer object has an hDC property and you need to hand that property to the RichTextBox control's SelPrint method. Because the CommonDialog control has a ShowPrinter dialog, you can use it to select the proper device context. Set the CommonDialog control's Flag property so that it returns the device context in its hDC property:

```
Private Sub mnuPrint_Click()
CommonDialog1.Flags = cdlPDReturnDC + _
   cdlPDNoPageNums
```

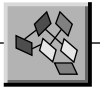


Then, if no text is selected in the RichTextBox control, preselect the All Pages radio button. Otherwise preselect the Selection radio button:

```
If RichTextBox1.SelLength = 0 Then
   CommonDialog1.Flags = CommonDialog1.Flags + cdlPDAllPages
Else
```

```
Option Explicit
Public sFind As String

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
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
```

LISTING 1 ***Fastest Text Editor in the West.*** *Demonstrate the power of Visual Basic and build a powerful text editor in less than five minutes.*




```
   CommonDialog1.Flags = CommonDialog1.Flags + cdlPDSelection
End If
```

Finally, call the ShowPrinter method of the CommonDialog control and then the SelPrint method of the RichTextBox control, handing the device context from the CommonDialog control as an argument:

```
CommonDialog1.ShowPrinter
RichTextBox1.SelPrint CommonDialog1.hDC
End Sub
```

Now you've got a powerful text editor that certainly beats Notepad in what—about three minutes?

TRY TO FIND IT

Why don't you add one more feature to your text editor: the ability to Find with Ctrl-F and Find Next with the F3 function key. Go back to the Menu Editor tool and add a top-level Edit with Find and Find Next menu items. Be sure to set the shortcut keys to Ctrl-F and F3. Because you'll want to "remember" what you are searching for and share that string with both the Find and Find Next functions, you need to define a Public variable, sFind, at the top of the code in your form. Then click on the Find menu item in the Edit menu, and use the InputBox function to let the user set the sFind variable. Pass that variable to the Find method of the RichTextBox control:

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

If the Find method is successful, the found text will be selected. In the Find Next function you need to turn off this selection, move one character past this selected text, and force the Find method to start at this point and search to the end of the text. You can use the SelStart and SelLength properties of the RichTextBox control to move past the prior selection. Then you can pass three arguments to the Find method of the RichTextBox control—the sFind variable as the search parameter, an empty second argument to start at the current insertion point, and the length of the text:

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

I can get to this point in about five minutes—that's usually enough to convince anyone to give Visual Basic a try. The code discussed in this column is also available on the VB-CD Quarterly and in the Magazine Library (#3) of the *Visual Basic Programmer's Journal* Forum (GO VBPJ) on CompuServe (see Listing 1).

Can you think of some more quick, easy, and useful features to add? Download my code, and see how you can enhance it, and send me your best results. Don't forget to include your time estimates. ×