Convert and Edit Data

Write a VB program to handle data conversion, and learn to add custom editing features to it.

by Chris Barlow

WHAT YOU NEED

VB3, 4, 5, or 6

Click & Retrieve Source CODE!

Learning Visual Basic is like any other skill—you need practice. However, if programming with VB isn't your full-time job, you probably have difficulty finding practice applications. To develop your VB skills, you need to look for a problem that you can use as an opportunity to write a simple VB application.

For example, I recently purchased an electronic organizer for my daughter. Nothing fancy—no Windows CE or modem—just a pocket device for her to keep track of addresses and appointments. As I began to download names and addresses from my contact manager on my laptop, I ran into some problems. My contact manager allows me to enter names as "FirstName LastName" or as "LastName, FirstName," and it finds and sorts these contacts in the proper order. The organizer, however, is less sophisticated and sorts by the first character in the field. In addition, because the organizer's screen size is only 8-by-20 characters, I needed to break up some of the lines to make them readable. I realized I needed to juggle some of the data fields before downloading the records to the organizer.

Because the organizer's import file is stored as an ASCII file, I tried loading the file into Word and using the search-and-replace function. But when I saved the file of 500 names and tried to download it to the organizer, I got only one name. When I closely compared the modified file with the unmodified version that I saved (*always save your work!*), I found that Word inserted a carriage return and line feed after every partial line, but the organizer seemed to expect only a line feed. I decided this was a great opportunity to write a small Visual Basic program to handle the data conversion. In this column, I'll show you how to do the same.

First, start a new project in Visual Basic (I'm using VB5, but this code works about the same in VB3, VB4, or VB6). Add a TextBox control to the form by clicking on the icon in the toolbox and drawing the control on the form. Set the MultiLine property of the control to True so it displays all the lines in the file. Don't worry about the exact size of the textbox at this time because you can write code to adjust the dimensions in the Form_Resize event (see Listing 1).

Although much of the buzz nowadays is about accessing databases with VB, one of Visual Basic's real strengths is how easily it works with files. You need to specify the file mode when you open a file with the Visual Basic Open statement. The simplest modes open the file sequentially to either input or output a line of ASCII characters ending with a carriage return and line feed. If you're going to read data from the file, use Input mode. If you plan to write to the file, you have a choice of two output modes: Output mode erases any data in the file, and Append mode writes any new data at the end of the file.

In addition to normal se-

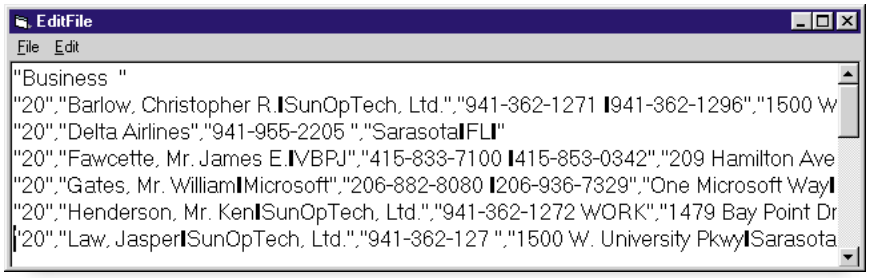

Figure 1 Custom Text Editor. This form shows the data from the test data file immediately after you execute the File | Open menu item. Notice the solid vertical bars interspersed with the text, indicating a control character.

quential input and output, VB recognizes two forms of binary file access—Random and Binary. Both modes allow you to read and write the entire range of ASCII characters, including control characters such as carriage return and line feed. In these modes, you work with a block of characters, or records, from the file. Random mode lets you specify a fixed number of characters for every read and write, and Binary mode allows you to read and write a variable number of characters with each statement. You should use Binary mode for this project.

Add a menu to your VB form by selecting the Menu Editor from the Tools menu. Create a simple File menu with menu items for Open, Save, and Exit. Click on the Open menu item from the File menu on your form to add some code. Using VB's Input function, you can read a given number of characters from an open file and return it to a TextBox control. Enter this code in the Open menu item's Click event:

```
Private Sub mOpen_Click()
Open "c:\windows\win.ini" _
   For Binary As #1
Text1 = Input(LOF(1), 1)
Close #1
End Sub
```

Note the use of the LOF function to return the length of the file in characters. Run this code and see your win.ini file appear in the TextBox control.

Make Code More Generic

Now that you get the idea of how file access works, you need to make a few changes to make your code more generic. First, you should not hardcode the number 1 for the file number because you might want to have another file open in your program. If your program tries to open two files as number 1, you get an error. The Visual Basic function FreeFile returns the next available file number as an integer. Change your code to look like this:

```
Private Sub mOpen_Click()
Dim fn As Integer
fn = FreeFile
Open "c:\windows\win.ini" For Binary As #fn
Text1 = Input(LOF(1), fn)
Close #fn
End Sub
```

You want to let the user select the file to open, rather than hardcode your procedure to open win.ini. The CommonDialog control lets you add the same File

VB5 | Code for the Custom Editor

```
Option Explicit
Private Function Replace(sF$, sR$) As Integer
Dim pos&, iCnt
Do
  pos = InStr(pos + 1, Text1, sF)
  If pos Then
    Text1 = Left(Text1, pos - 1) & sR & Mid( _
      Text1, pos + Len(sF))
    iCnt = iCnt + 1
  End If
Loop While pos
Replace = iCnt
End Function

Private Sub Form_Resize()
Text1.Height = Me.Height - 400
Text1.Width = Me.Width - 100
End Sub

Private Sub mExit_Click()
Unload Me
End Sub

Private Sub mLF_Click()
Dim pos&
pos = Text1.SelStart
Text1 = Left(Text1, pos) & vbLf & Mid(Text1,
pos + 1)
Text1.SelStart = pos
End Sub

Private Sub mOpen_Click()
Dim fn As Integer
fn = FreeFile
CommonDialog1.ShowOpen
Open CommonDialog1.filename For Binary As #fn
Text1 = Input(LOF(1), fn)
Close #fn
End Sub

Private Sub mRemoveMr_Click()
Dim iCnt As Integer
iCnt = Replace("Mr. & Mrs.", "")
MsgBox "Removed " & CStr(iCnt) & " Mr. & Mrs."
iCnt = Replace("Mr.", "")
MsgBox "Removed " & CStr(iCnt) & " Mr."
iCnt = Replace("Mrs.", "")
MsgBox "Removed " & CStr(iCnt) & " Mrs."
iCnt = Replace("Ms.", "")
MsgBox "Removed " & CStr(iCnt) & " Ms."
End Sub

Private Sub mReplace_Click()
Dim sF$, sR$
sF = InputBox("Enter text to find")
sR = InputBox("Enter replacement text")
MsgBox Replace(sF, sR) & " replacements"
End Sub

Private Sub mSave_Click()
Dim fn As Integer
fn = FreeFile
CommonDialog1.ShowSave
Open CommonDialog1.filename For Binary As #fn
Put #fn, , Text1.Text
Close #fn
End Sub
```

Listing 1 This source is written for VB5, but works with only minor changes in VB3, VB4, or VB6. You might want to add error-checking code in the mOpen and mSave procedures to make sure the user has not clicked on the Cancel button on the common dialog.

Open and File Save dialogs used by most Windows programs. Add a CommonDialog control to your form and change your code to look like this:

```
Private Sub mOpen_Click()
Dim fn As Integer
fn = FreeFile
CommonDialog1.ShowOpen
Open CommonDialog1.filename _
   For Binary As #fn
Text1 = Input(LOF(1), fn)
Close #fn
End Sub
```

Notice that the CommonDialog control's ShowOpen method changes to the folder where the file is located, so you need to use only the FileName property to open the file.

Now you have a generic procedure that displays a common dialog to open any file and display it in a TextBox control. The file-saving procedure is quite similar. You can use the Put statement to write to a binary file from the Text property of the TextBox control. Use the CommonDialog control's ShowSave method to let the user select the file name, and the FreeFile function to let the user select the file number:

VB recognizes two forms of binary file access: Random and Binary.

```
Private Sub mSave_Click()
Dim fn As Integer
fn = FreeFile
CommonDialog1.ShowSave
Open CommonDialog1.filename For _
   Binary As #fn
Put #fn, , Text1.Text
Close #fn
End Sub
```

Add these lines of code to resize the TextBox control to the form size and to end your application when the user clicks on the Exit menu item on the File menu:

```
Private Sub Form_Resize()
Text1.Height = Me.Height - 400
Text1.Width = Me.Width - 100
End Sub

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

You now have a working application to load a file, type in changes, and save the file. Now let's add some custom editing features to the files to import to the organizer.

Away With the Appellations

There were several edits I wanted to make to the import file: I wanted to be able to insert an ASCII 10-line feed character anywhere

in the file, and I wanted to search for certain company names and replace them with a shorter version. My contact manager adds an appellation, such as "Mr.", in front of each name, but I don't need appellations in the organizer. VB makes it easy to make these kinds of changes.

First, add an Edit menu to your Visual Basic form with three menu items: Insert Linefeed, Replace, and Remove Appellations (see Figure 1). Click on the Insert Linefeed menu item from the File menu on your form to add some code.

The SelStart property of the TextBox control contains the location of the insertion point within the text. Think of the text within the TextBox control as a long string of characters. You can use the built-in string manipulation functions to manipulate the text in the control just as you can manipulate any string. To insert a line-feed character, take all the text to the left of the insertion point using the Left function, concatenate the special constant vbLf, and concatenate the remainder of the text using the Mid function:

```
Private Sub mLF_Click()
Dim pos&
pos = Text1.SelStart
Text1 = Left(Text1, pos) & vbLf & _
   Mid(Text1, pos + 1)
Text1.SelStart = pos
End Sub
```

Replacing text is similar to inserting text. The only difference is that you must search for a text string, insert the replacement text, and concatenate the remaining text starting at the end of the found string. Use the InStr function to search for a text string and locate its starting point. This function has an optional parameter that indicates the starting point of the search, which you can use to loop through the text string until you find and replace all occurrences of the text string. Write a generic function, and pass in the found string and the replacement string as parameters:

```
Private Function Replace(sF$, sR$) _
   As Integer
Dim pos&, iCnt
Do
   pos = InStr(pos + 1, Text1, sF)
   If pos Then
      Text1 = Left(Text1, pos - 1) & _
         sR & Mid(Text1, pos + _
            Len(sF))
      iCnt = iCnt + 1
   End If
Loop While pos
Replace = iCnt
End Function
```

Notice how you can increment a counter each time through the loop and return the count in the function. Click on the Replace menu item on your Edit menu, and add the code to let the user enter the find and replacement strings using the InputBox function. Use the MsgBox function to display the number of replacements:

Although much of the buzz nowadays is about accessing databases with VB, one of Visual Basic's real strengths is how easily it works with files.

```
Private Sub mReplace_Click()
Dim sF$, sR$
sF = InputBox( _
   "Enter text to find")
sR = InputBox( _
   "Enter replacement text")
MsgBox Replace(sF, sR) & _
   " replacements"
End Sub
```

You can reuse this generic replace function to remove the appellations from the file by replacing the appellation with a Null string. Click on the Remove Appellations menu item on your Edit menu and insert this code:

```
Private Sub mRemoveMr_Click()
Dim iCnt As Integer
iCnt = Replace("Mr. & Mrs.", "")
MsgBox "Removed " & CStr(iCnt) & _
   " Mr. & Mrs."
iCnt = Replace("Mr.", "")
MsgBox "Removed " & CStr(iCnt) & "
Mr."
iCnt = Replace("Mrs.", "")
MsgBox "Removed " & CStr(iCnt) &
"Mrs."
iCnt = Replace("Ms.", "")
MsgBox "Removed " & CStr(iCnt) & "
Ms."
End Sub
```

Easy, isn't it? I used this little program to reformat the import file and get the name and address data into my daughter's organizer. I'm sure you can find a similar problem just waiting to be solved by one of your new Visual Basic applications. 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, type **VBPJ0199** into the Locator+ field at the top right of the VBPJ home page. (You first need to register, for free, on DevX.) The free code for this article includes all code listings, plus the simple file editor application.

To get the code for this article only, available to DevX Premier Club members, type **VBPJ0199GS** into the Locator+ field.