Execute a VB6 Program with Command Line Arguments

I frequently receive queries from readers asking if Visual Basic supports Command Line Arguments. The answer is 'Yes', and in this article, I'll show you how to use them.

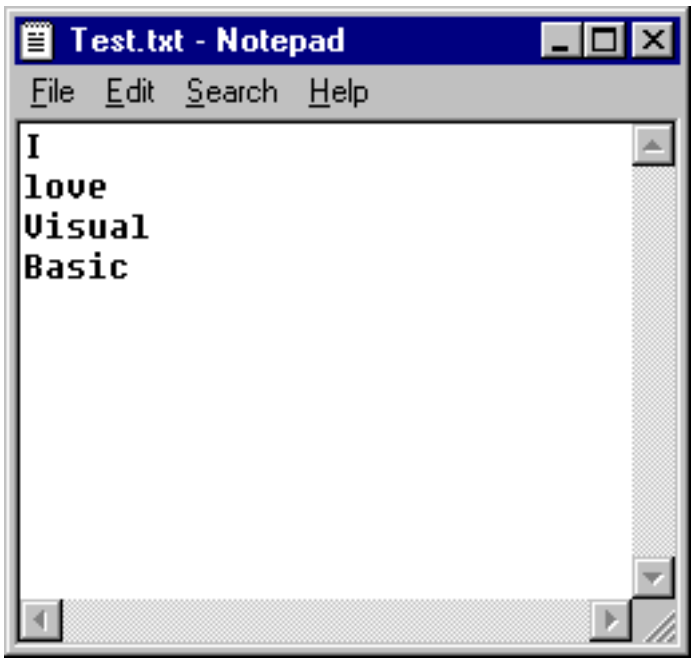
For those of you not familiar with the concept, programming languages, such as C and C++, allow you to execute a compiled program with one or more Command Line Arguments, which are then used to modify the behavior of the program at run time.

For instance, suppose you write a program to read a text file and process the records in that file. Suppose 99 times out of 100 that file is located in a specific directory on your Local Area Network, and so you 'hard code' the file path into your program code. However, that 1 time out of 100 that the file is not where your program expects it to be, the program unceremoniously bombs. This would be a perfect use for a Command Line Argument, which, if supplied to the program, can override the 'hard coded' file path. Let me illustrate by first writing the code to open a file, read the records, display them on the form, and then close the file. Let's place that code in the Click Event Procedure of a Command Button…

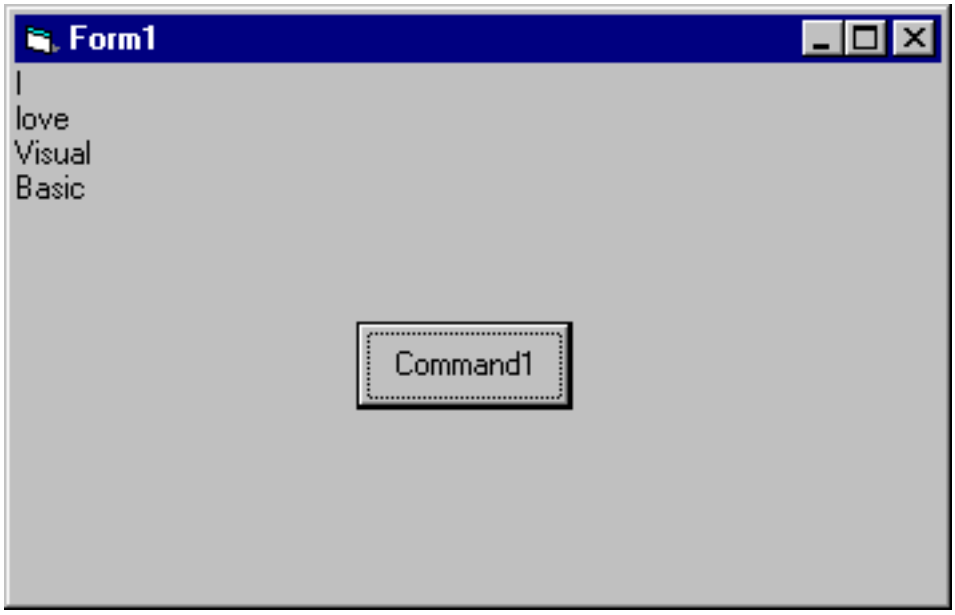
```
Private Sub Command1_Click()

Dim strValue As String

Open "c:\vbfiles\august2000\test.txt" For Input As #1

Do While Not EOF(1)
  Input #1, strValue
  Form1.Print strValue
Loop

Close #1

End Sub
```

What we're doing here is opening a file called 'test.txt' contained in the "\vbfiles\august2000" folder on my hard drive, reading each line into a variable called strValue, and then using the Print Method of the form to display that line on the form (for those of you not familiar with this code, check out my first book, Learn to Program with Visual Basic 6.)

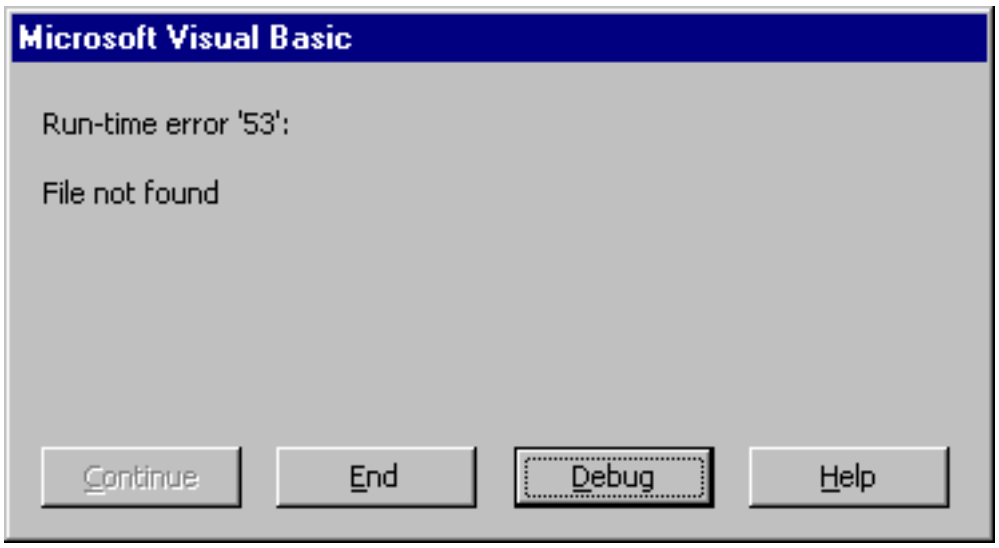
Next, let's create the text file…



Now let's run the program, click on the Command Button and see what happens.



As we expected, when the Command Button is clicked, the file is opened, each line is 'read' into the variable called strValue, and then the value of strValue is displayed on the form using the Form's Print Method. Now let's see what happens if by chance the file is not located where our program is expecting it. First I'll stop the program, and then I'll rename the file "text. txt" to "text.old". Now let's see what happens when I run the program and click on the Command Button…



Within the Visual Basic IDE, we receive this nasty message. If we had compiled this program into an executable, and our user had run it from the Start Menu, the results would be even nastier---the program would bomb with an error message.

Now this is where a Command Line Argument can come in handy to let our program know that the file is in a different location. What we can do is write our program in such a way that if a Command Line Argument is **NOT** passed to it, the program presumes the file is in the normal location. However, if a Command Line Argument is passed, the program can take the value of that argument, and use it as the alternate file path name.

The Command Function

How are Command Line Arguments passed in Visual Basic? And how are they detected by the running program?

Let's answer the last question first.

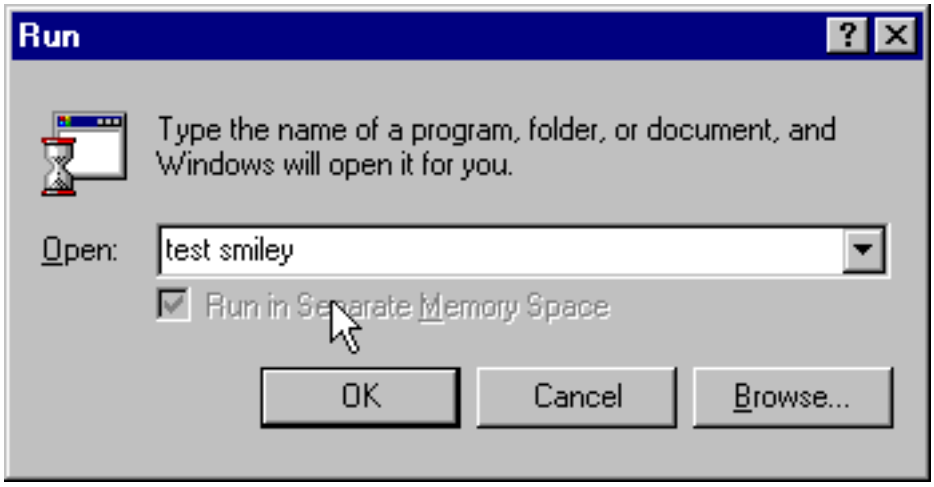
Command Line Arguments, if passed to an executable, are stored in a special System Variable, and can be returned to our program using the Command Function. For instance, if a Command Line Argument of "smiley" is passed to an executable, this code

```
Form1.Print Command
```

would result in 'smiley' being displayed on the Form.

Passing the Command Line Argument is pretty easy---you just type the name of your executable at the Run Menu, followed by a space, followed

by the Command Line Argument. For instance, if our program's name was Test, and we wanted to pass a Command Line Argument called 'smiley', this is what the Command Line would look like…



Take note here that Command Line Arguments can only be used with Compiled Programs---that is, programs that have been compiled into an . EXE file.

It is possible to test a program that accepts a Command Line Argument within the Visual Basic IDE however (more on that in a little bit)---but bear in mind, you can only execute a program using Command Line Arguments if it has been compiled into an EXE.

If no Command Line Argument has been passed to the program, then the value returned by the execution of the Command Function will be an Empty String, or "".

Let's modify the code we've written in the Click Event Procedure of the Command Button to take into account the passing of a Command Line Argument. Here's the code…

```
Private Sub Command1_Click()

Dim strValue As String

If Command = "" Then
  Open "c:\vbfiles\august2000\text.txt" For Input As #1
Else
  Open Command For Input As #1
End If

Do While Not EOF(1)
```

```
    Input #1, strValue
    Form1.Print strValue
Loop

Close #1

End Sub
```

All we've added here is an 'If statement' to interrogate the value of Command. If it's an Empty String

```
If Command = "" Then
```

we know that no Command Line Argument has been passed and we can open the file as usual

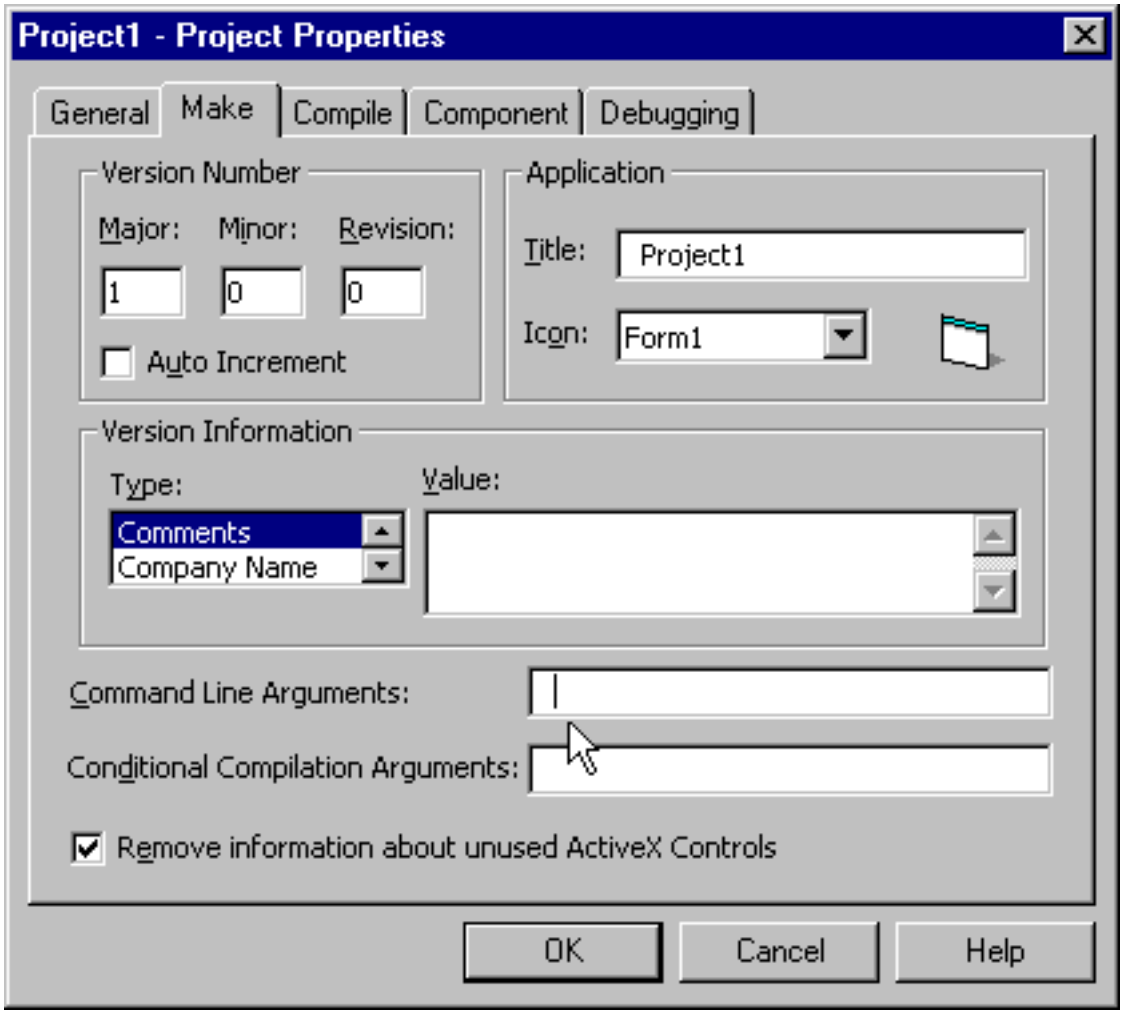
```
Open "c:\vbfiles\august2000\text.txt" For Input As #1
```

However, if the value of Command is anything other than an Empty String, then we know that the program has been executed with a Command Line Argument, and we use the value of Command as the file name argument in the Open Statement…

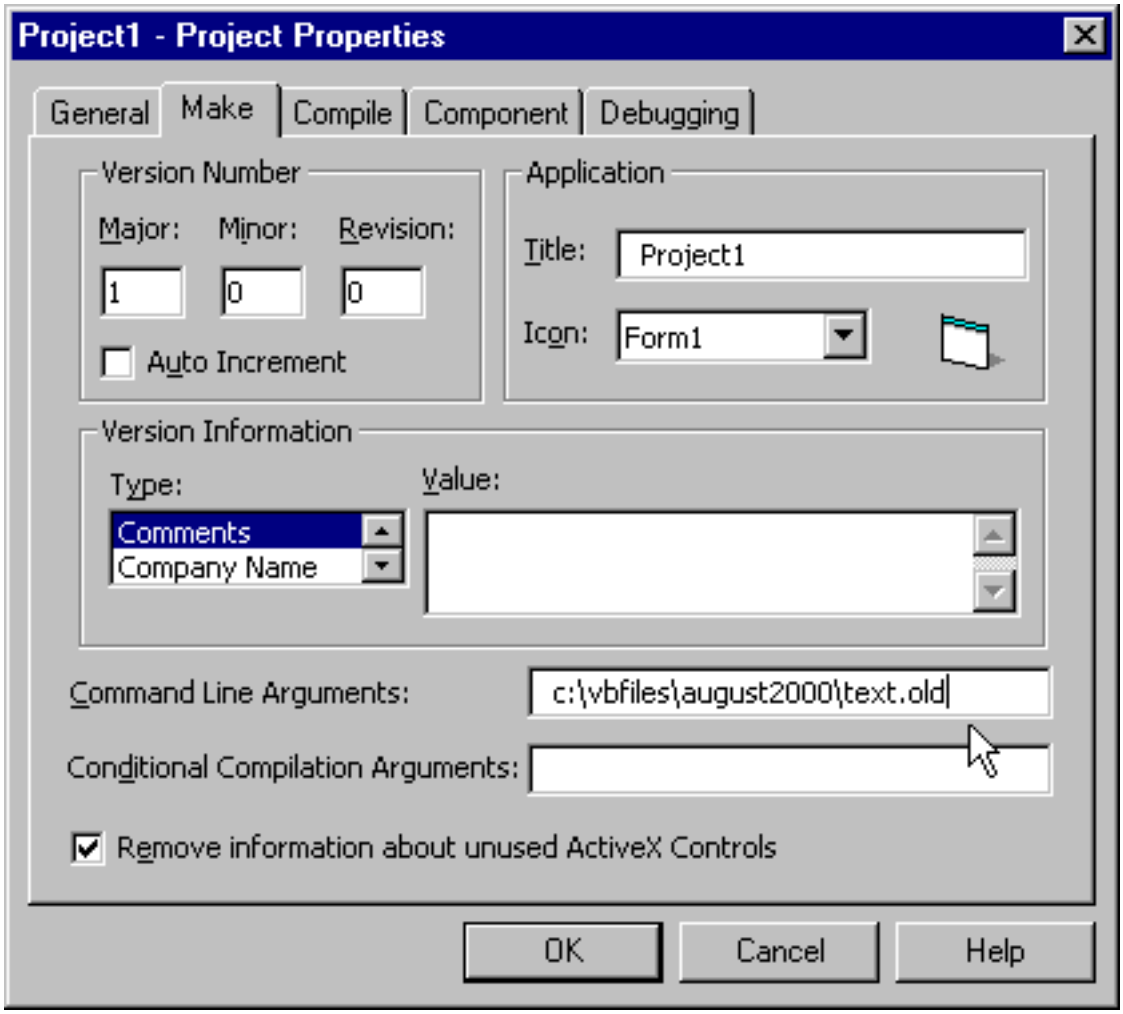
```
Open Command For Input As #1
```

Testing the Command Function in the Visual Basic IDE

As I mentioned earlier, the only way to pass a Command Line Argument is when you execute an executable version of your program in the Run Window, and that means you must compile your program. However, it is possible to test your program to see how your Command Line Argument logic is working within the Visual Basic IDE. You can simulate the passing of a Command Line Argument within the IDE by bringing up the Properties Window for your project and selecting the 'Make' Tab…



You can see from the screen shot above that there is a Textbox labeled 'Command Line Argument'---an value placed in this Textbox simulates the program running as an executable with a Command Line Argument. For instance, if we place the file path of our re-named text file in this Textbox…



and then run the program within the IDE, our program will detect the alternate location of the file.

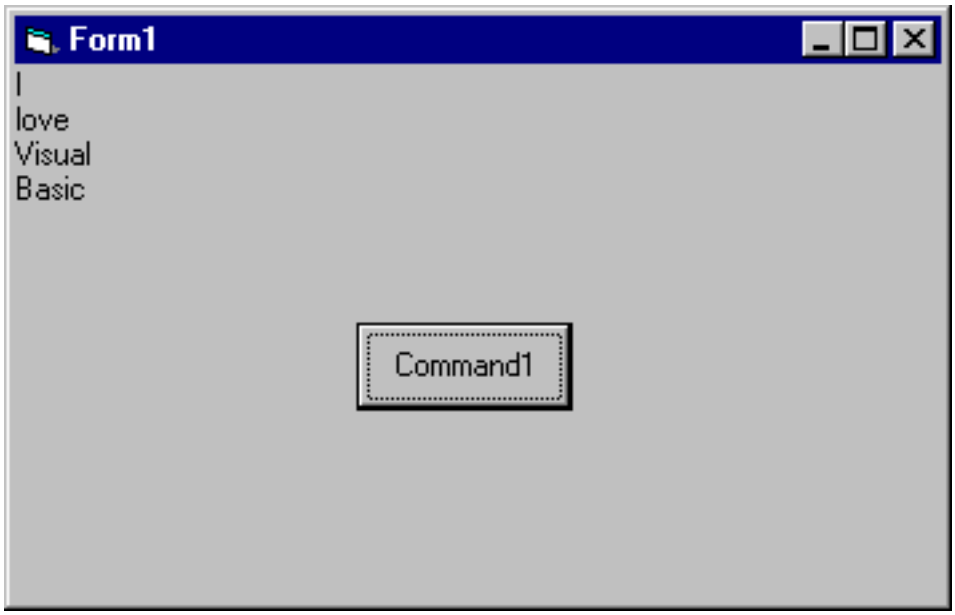
Before we run the program within the IDE, let's add a line of code to the Click Event Procedure to display the value of Command in a Message Box…

```
Private Sub Command1_Click()

Dim strValue As String

MsgBox Command

If Command = "" Then
  Open "c:\vbfiles\august2000\text.txt" For Input As #1
Else
  Open Command For Input As #1
End If

Do While Not EOF(1)
```

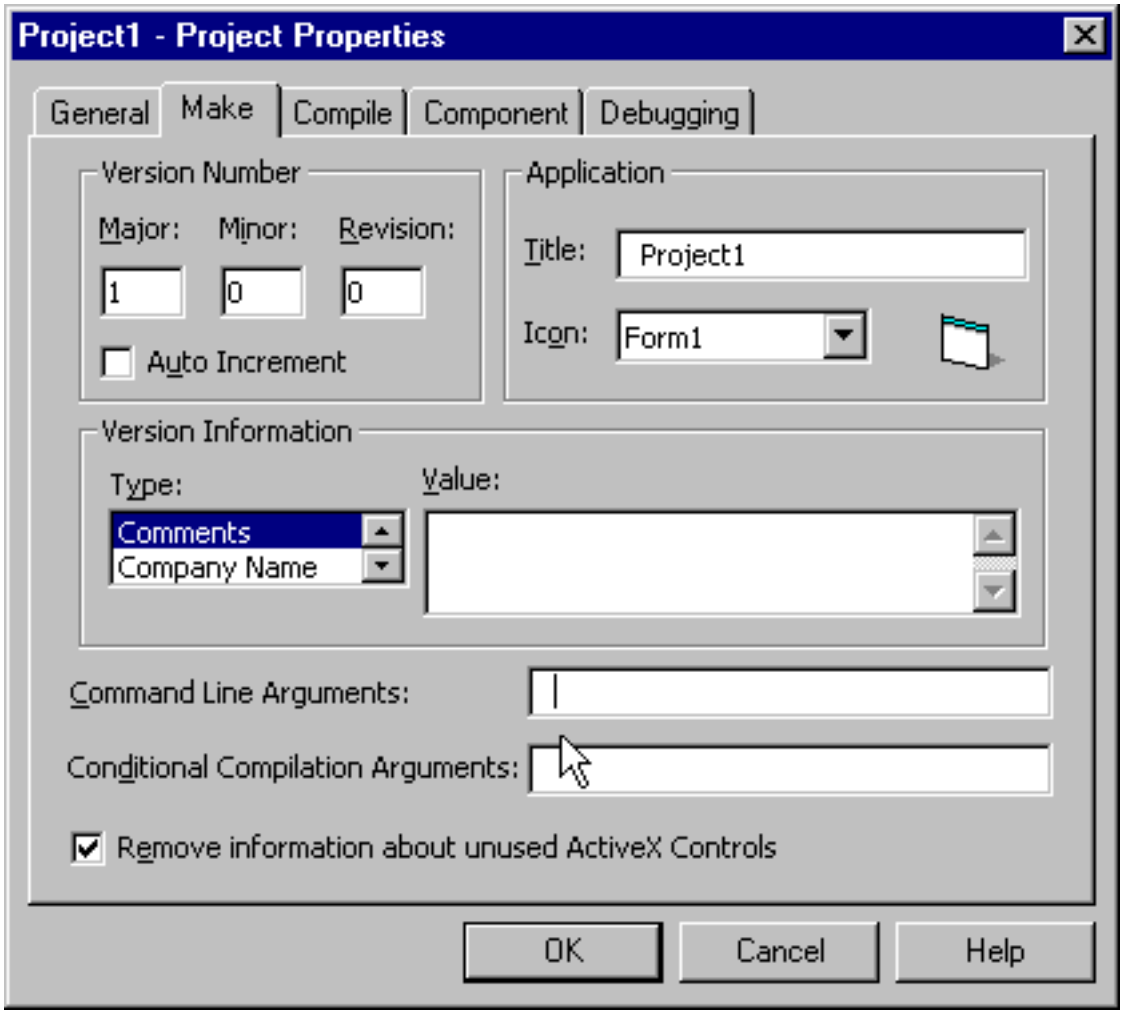
```
    Input #1, strValue
    Form1.Print strValue
Loop

Close #1

End Sub
```

Now let's run the program and see what happens. If our code is correct, because of the presence of a Command Line Argument in our Project's Properties Window, the program should detect a value returned from the Command Function, and properly open the alternate file. The first thing that happens when we click on the Command Button is that the program displays the value of Command in a Message Box…

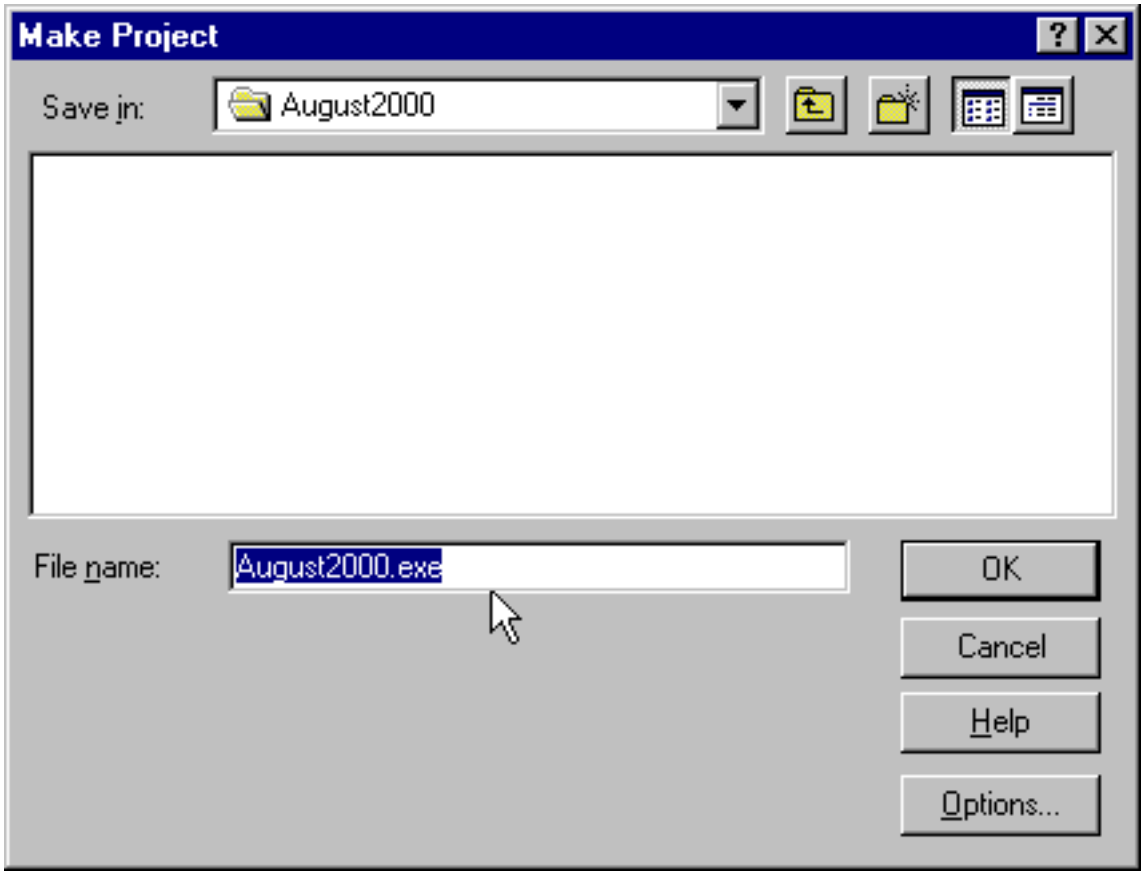
and then when we clear the Message Box, it opens the alternate file and displays its records on the form…



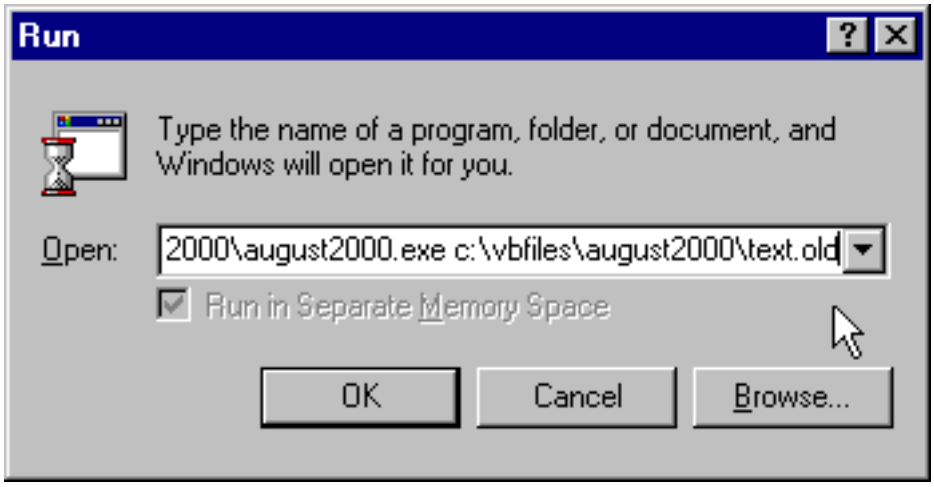
What remains now is to clear the Command Line Argument from the Project's Properties Window…



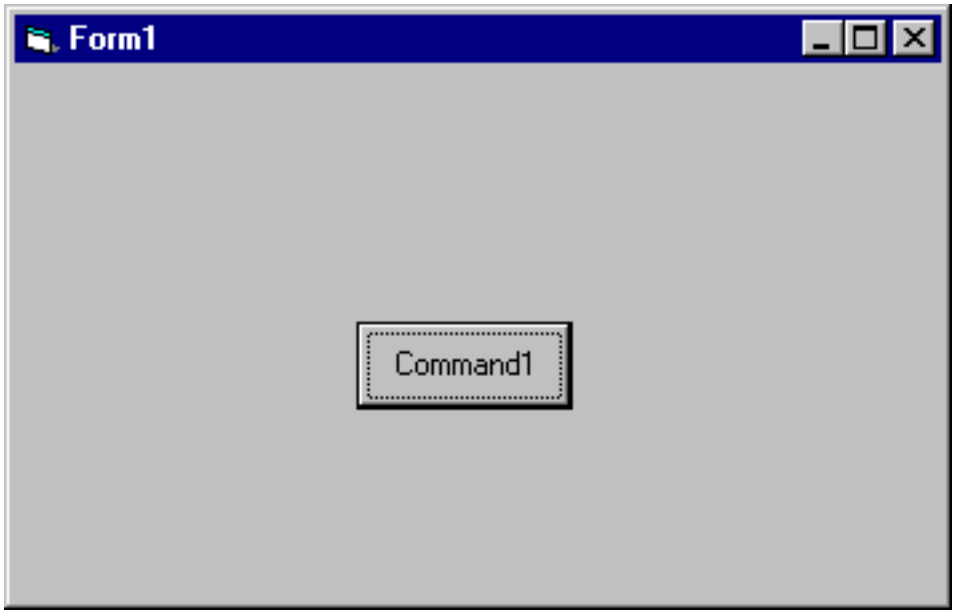
and then to compile this project into an executable by selecting File-Make from the VB Menu Bar. Let's call the executable August2000…



With the program compiled, let's exit out of Visual Basic, and test the execution of the program with a Command Line Argument…

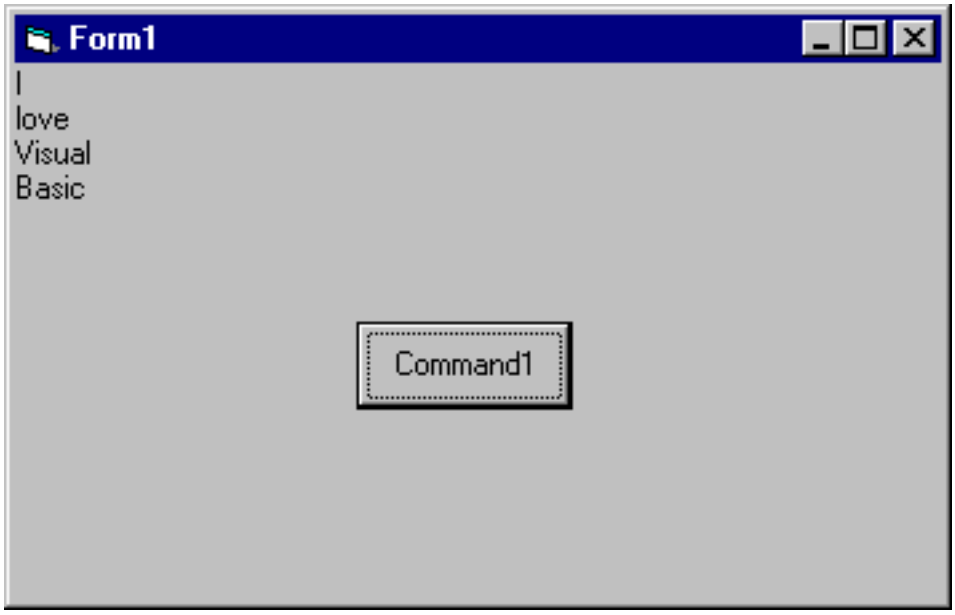


The program's form appears…



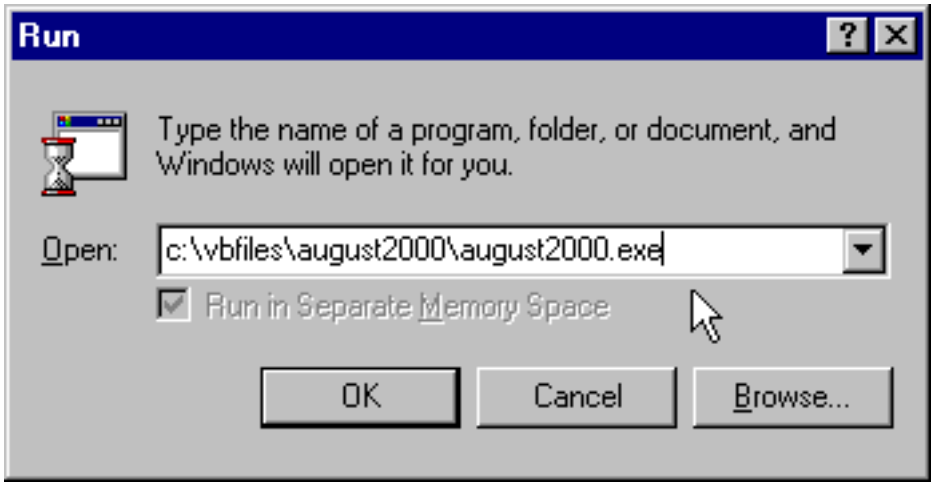
and when we click on the Command Button, the program displays the value of our Command Line Argument in a message box …

and then properly opens and displays, on the form, the records in the alternate file …



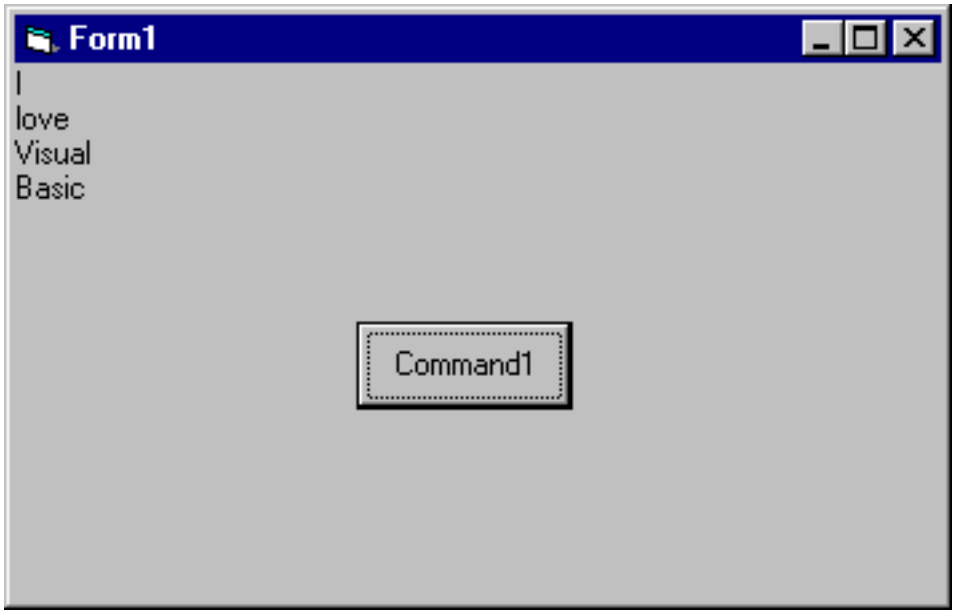
Now let's stop the program by clicking on the Form's Close button, and

using Windows Explorer, change the name of the alternate file back to its default value of 'text.txt'. Finally, let's run the program WITHOUT a Command Line Argument…



When we do so, once again the program's form is displayed, and when we click on the Command Button, our program displays, in a Message Box, the return value of the Command Function which is an Empty String--since the program was executed without a Command Line Argument…

and then properly opens and displays, on the form, the records in the default file …



More than one Command Line Argument?

I'm sure that some of my C and C++ programmers are wondering if it's possible to supply more than one Command Line Argument to the program, as it is in C or C++. The answer is a qualified 'Yes'.

Unlike C and C++, where Command Line Arguments are passed as an array of values to the executable program, in Visual Basic whatever follows the name of the executable program in the Run Window is passed as a single String value. What that means is that it's possible to pass more than one Command Line Argument, but 'parsing' (separating) the arguments is your responsibility---Visual Basic will not do it for you.

Hopefully you paid careful attention to my chapter on String Manipulation in my first book. For instance, if you pass multiple Command Line Arguments in your Visual Basic program separated by spaces, you'll need to use String Manipulate to detect the space character and 'extract' the argument to the left and to the right of it.

Summary

I hope if you were one of the many people who wrote to me asking for an explanation of Command Line Arguments in VB, that I've answered your questions. If you had never thought of using them before, they can be the solution to the problem of modifying the behavior of your program at runtime.