Handling Files in Visual Basic

by Alex Allan

Introduction

The majority of Windows applications handle files. From Microsoft Word to Windows Explorer to Greasy Joe's Easy Accounts package.

But how do you do this in Visual Basic? Via the **Open** statement.

In this article, I'll be covering everything you need to know - from start to finish. Perhaps even from open to close, if I'm feeling funny.

Don't forget that I'd love to hear what you think about this article. Whether it's likes or gripes, post it on the bulletin board.

But for now, let's get started.

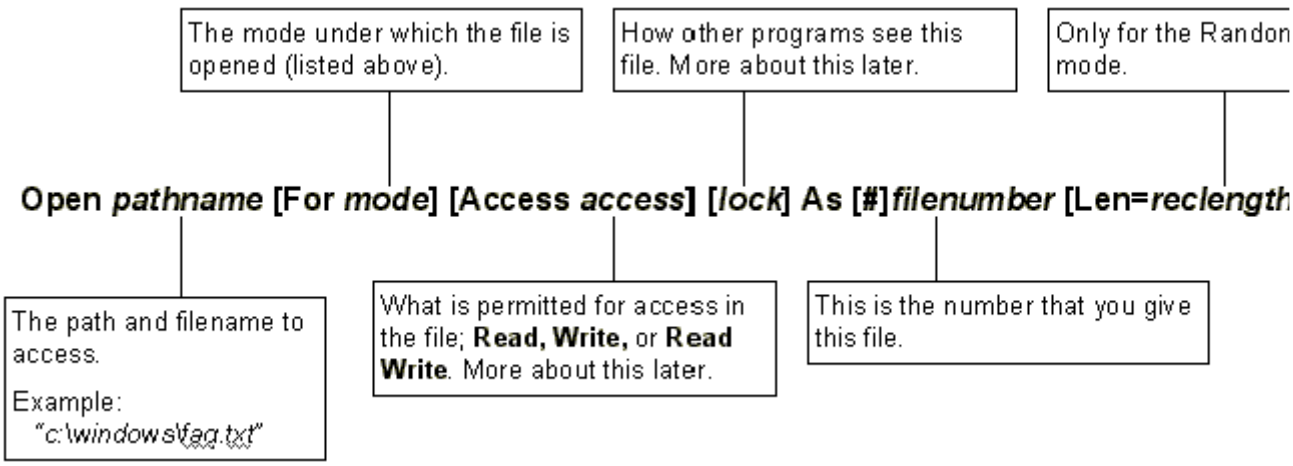
Getting Started

Before we delve into writing and reading files, let's ask ourselves a question; why would I want to write to a file?

Well, programmers often want to write data to a disk or retrieve data from it. And that data can be anything - from settings to plain text to pictures. Heck, you could even create your own mini word processor!

But don't confuse this with databases. If you're interested in accessing databases using Visual Basic, you'd better head down to Karl Moore's tutorial.

Now, we access files using the Open statement.



Here's an example:

```
Open "c:\windows\faq.txt" For Input As #1
```

Don't worry about the syntax for now, you'll get used to it. This example opens the file faq.txt for input (file reading). The file is referred to by the number 1.

Before we dive into the intricacies of using the Open statement, let's take a peek at three different modes available to us when accessing files:

- Input, Output and Append
- Random
- Binary

We'll cover each of these separately.

In, Out, Shake It All About

First off, let's take a look at the file modes input, output and append. You use these three types to read or write plain text - such as that found in .txt, .bat and .ini files.

But when should you use each mode? Well, it depends on what you want to do. Use the following list to help you decide...

- The **Output** mode creates a blank file and allows you to write information into it.
- The **Append** mode is similar to the Output mode but appends (adds to) an existing file.
- The **Input** mode opens a file for reading.

Top Tip: *You may hear these file modes being referred to as 'sequential files'. That's 'cause once you have read or written to a line, you can't go back to it unless you close and re-open. In other words, the modes are one way – sequential.*

So, for example, to open a file for **output**, you'd use the Open statement like this:

```
Open "c:\windows\faq.txt" For Output As #1
```

That's all fine and dandy, but how do you **use** each mode after you've **open**ed the file?

To **write** to a file we use (Output and Append only):

```
Print #filenumber, expression
```

To **read** from a file we use (Input only):

```
Input #filenumber, variablelist
```

This probably looks completely confusing at the moment, so let's figure out what it all means.

Let's imagine you've opened a file for output, like this...

```
Open "c:\groovy\myfile.txt" For Output As #1
```

...we now want to put information into this file using the Print statement, like this:

```
Print #1, "Hello World!"
```

This inserts the information you pass it direct to the file in #1.

If you'd opened a file for Input, like this...

```
Open "c:\groovy\myotherfile.txt" For Input As #1
```

...you can read information from the file, like this...

```
Input #1, MyVariableName
```

This reads information from the file in #1 and puts it into your variable.

Let's use an example - it's easier to explain that way!

Building a Sample

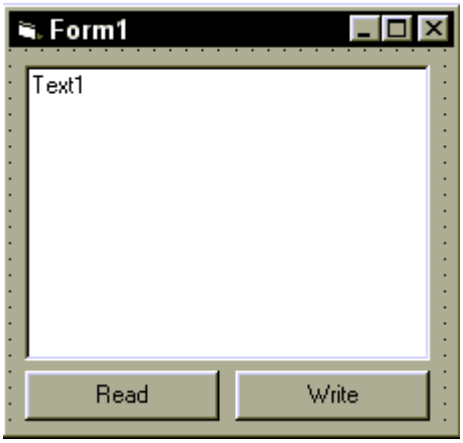
Let's build a sample to demonstrate accessing files:

Open Visual Basic and double-click on "Standard EXE"

You should be left with a blank form.

- Throw a simple Text Box onto the form
- Go to the Properties window and change MultiLine to True
- Create two Command buttons, setting the caption of the first to Read and the second to Write.

So far, it should look something like this:



Now, double-click on Read and insert the following code:

```
Private Sub Command1_Click()
    'Outline:       - Asks the user for a file.
    '               - Reads all the data into Text1.

    Dim FilePath As String
    Dim Data As String

    FilePath = InputBox("Enter the path for a text file", "The Two R's", _
    "C:\WINDOWS\WINNEWS.TXT")
    'Asks the user for some input via an input box.

    Open FilePath For Input As #1
    'Opens the file given by the user.

    Do Until EOF(1)
    'Does this loop until End Of File(EOF) for file number 1.
        Line Input #1, Data
        'Read one line and puts it into the varible Data.
        Text1.Text = Text1.Text & vbCrLf & Data
```

```
        'Adds the read line into Text1.
        MsgBox EOF(1)
    Loop

    Close #1
    'Closes this file.
End Sub
```

Read the comments (anything prefixed by an apostrophe).

Next, double-click on Write and insert the following code:

```
Private Sub Command2_Click()
    'Outline:       - Asks the user for a file.
    '               - Writes it all to a the file.

    Dim FilePath As String

    FilePath = InputBox("Enter the path for a text file", _
    "The Two R's")
    'Asks the user for some input via an input box.

    Open FilePath For Output As #1
    'Opens the file given by the user (for Output).

    Print #1, Text1.Text
    'Writes the data into the file number #1

    Close #1
End Sub
```

Once again, read all the comments. Do you understand what's happening?

Hit F5 to run your program!

Congratulations! You've just created a simple text editor. The Write button performs a simple 'Save As' whilst the Read button is the equivalent of 'Open'.

Didn't I Mention Those?

You may have noticed a few things in my code that I haven't told you about. Let's take a peek at a few geeky code words...

```
Line Input #filenumber, MyVariableName
```

- This is the same as Input but it reads the whole line instead of stopping at a comma (which can be useful - sometimes)

```
EOF(#filenumber)
```

- This outputs a true or false value, depending on whether it has hit the 'end' of the file. In my code, I used it in the Do…Loop for Read. When EOF=True, the loop ends.

```
Close #filenumber
```

- This closes the file. It allows other files to open this file.

And if you'll be getting real friendly with files in Visual Basic, here are a few other file writing

functions you may be interested in:

```
Write #filenumber, outputlist
```

This is similar to Print but it writes "screen formatted data" instead of raw data to a file. This means that values (numbers) have hashes ("#") put around them and strings have quote marks put around them. This makes the text less human readable but easier to read for your programs.

```
LOC(#filenumber)
```

The LOC function returns the current read/write position within an open file.

```
LOF(#filenumber)
```

The LOF function gives you the length of the file open.

```
FreeFile
```

This will give you next available file number.

Example:

```
MyFileNumber = FreeFile
Open "c:\windows\faq.txt" For Input as #MyFileNumber
Input$(number_of_chars_to_return, #filenumber)
```

This is the same as Input and Line Input except it has no limits (except the ones you set in number_of_chars_to_return). By using LOF - this can be used to get the whole file. We could replace all the stuff in the Do…Loop in the Read button, with:

```
Text1.Text = Input$(LOF(1),#1)
```

Why not have a play around and see what these do? Go on, have a go!

That's about it for the Input, Output and Append modes. If you're thirsty for more, check out John Percival's article on Adding Lines to Autoexec.bat.

The Random Mode

Now, scrap all you know about writing to text files. Yep, the methods of accessing random files are wildly different.

This method of writing to files I personally find a lot more useful. Instead of storing data in lines, random mode files are stored in records, just like an Access database stores its information.

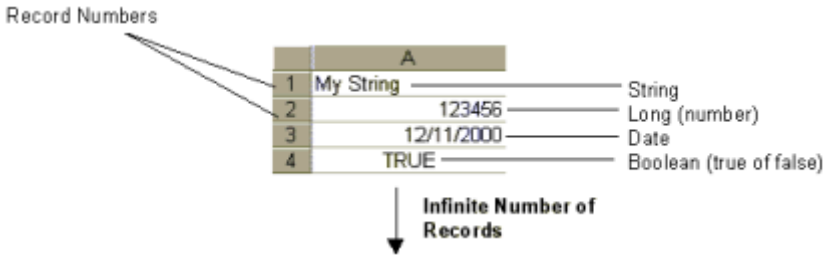
Let's look at a sample of how we can open a file for Random access:

```
Open "c:\myfile.ran" For Random As #1 Len=1000
```

Here you can see that the file myfile.ran is being opened as a random file under the number 1. However, the more observant of you will notice there is an extra bit at the end - Len= - I'll explain what that does in a short while.

Firstly, we have to get the Random file format clear in our heads. You need to think of it as being

like an Excel spreadsheet with only one column (the "A" column) and expanding for an infinite number of rows down. As with Excel each of these boxes can hold its own type of data. Let's take a good ol' peek at the diagram below:



This is how random files are stored and how you, as the developer, will access data in these files.

Next, we need to actually know how to read and write to these random files. For this we used the two commands:

```
Get #filenumber, recnumber, varname Statement
Put #filenumber, recnumber, varname Statement
```

You use **Get** to read data from your file and **Put** to write data to your file. For #filenumber you use the number that you used in the Open statement (e.g. #1), for the recnumber you put the record you'd like to access and the varname is the variable that you want to read into (*Get*) or write into (*Put*).

Once again, it's probably easier to demonstrate this with another sample!

And Another Sample!

Go through the following steps in order to build a sample application using the Random mode:

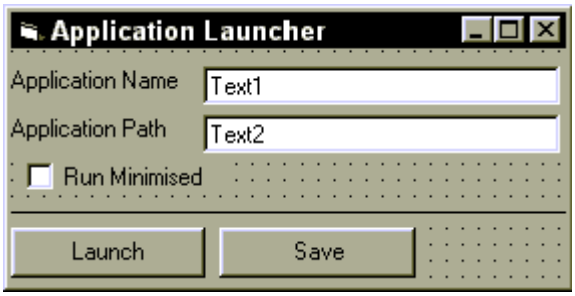
Open Visual Basic and double-click on "Standard EXE"

You will be left with a blank form.

Add two Text Box controls, two Command Buttons and a Check Box onto the form

Don't worry about naming conventions for now.

<Alex grins insanely>



Next, insert the following code into the (Declarations) code section of the Form.

```
Option Explicit
'Forces us to declare our varibles.
```

```
Private tmpText1 As String
Private tmpText2 As String
Private tmpCheck1 As Boolean
'Declares temp varibles.

Private Sub Command1_Click()
    'Outline:   Launches this program.

    If MsgBox("Click OK to run  " & Text1.Text, _
        vbOKCancel + vbQuestion, "The Open Statement") = _
        vbCancel Then
        'If the user clicked on 'Cancel'
        Exit Sub
    Else
        'If the user clicked OK
        Shell Text2.Text, IIf(Check1 = True, _
        vbMinimizedFocus, vbNormalFocus)
    End If
End Sub

Private Sub Form_Load()
    'Outline:   - Opens the test file
    '           - Opens records 1-3 into the controls.

    Open "c:\test.ran" For Random As #1

    If FileLen("c:\test.ran") = 0 Then CreateBlankFile
    'If this file has just been created then run
        ' the sub 'CreateBlankFile'

    Get #1, 1, tmpText1
    Get #1, 2, tmpText2
    Get #1, 3, tmpCheck1
    'Gets all the values and puts them into
        ' the temp varibles.

    Text1.Text = tmpText1
    Text2.Text = tmpText2
    Check1 = tmpCheck1
    'Sets the values in the controls to the temp values.


End Sub

Private Sub Command2_Click()
    'Outline:   Saves the current values.

    tmpText1 = Text1.Text
    tmpText2 = Text2.Text
    tmpCheck1 = Check1
    'Sets the temp values to the values in the controls.

    Put #1, 1, tmpText1
    Put #1, 2, tmpText2
    Put #1, 3, tmpCheck1

End Sub

Private Sub CreateBlankFile()
    'Outline:   This creates a blank file.

    tmpText1 = ""
    tmpText2 = ""
    tmpCheck1 = False
```

```
    'Clears the temp varibles.

    Put #1, 1, tmpText1
    Put #1, 2, tmpText2
    Put #1, 3, tmpCheck1
    'Writes the varibles into the file
End Sub
```

Read all the comments here. Can you figure out what's happening?

Hit F5 to run the program!

Well done - you've just created a program launcher! Try it out. Does it work?

Top Tip: *If you're using Visual Basic 4 or below, it's probably time to upgrade. Oh, and you'll also have to declare any variable and constants before writing them to a file (using the Dim statement).*

What's What?

Let's run over a few particulars of the code that'll be of interest. Well, maybe.

- You may notice that I used a few temporary variables (tmpText1, tmpText2 and tmpCheck1) to store data in before I read or wrote data into the controls. This is because Get and Put are very picky about the data they read or write. Data for Get and Put must not come directly from form controls or user defined types.

```
FileLen(#filenumber)
```

FileLen outputs the file size in bytes. So, if LenFile(1) in our code equals 0 then the Open statement has just made the file. Therefore, we will need to create this file using our CreateBlankFile subroutine.

```
IIf(expression, truepart, falsepart)
```

See John Percival's IIf Tip function

But what about that 'Len=' mystery I promised to unravel for you?

Let's think about this in terms of an Excel Spreadsheet. Think of each cell having a maximum size. This maximum size is set (in bytes) in the Len= section of the Open statement (by default it's 256 bytes).

Unfortunately, as with everything else in life, size matters. So there is a drawback in setting a huge length using 'Len ='. Try out the following code:

```
Open "c:\test.ran" For Random As #1 Len=10240
Put #1,1000,"Hello"
Close #1
```

The results aren't pretty. You will end up with a file which is about 10MB. Why? Because you created a record at the point 1,000. Now, Visual Basic must leave space to put records before record 1,000 it will create a bundle of blank records from 1 to 999, each taking up 10,240 bytes. So, this leaves a file that is:

10,240 bytes x 1,000 records = 10,240,000 bytes

so, that's *10,240,000 / 1048576* (that's how many bytes there are in a megabyte) which equals *9.77MB*

So be careful - you have been warned!

But anyway, enough calculations. Let's think about how we can take our work one step beyond.

I've made a module for use with VB that treats random mode files in sets of data. Each data set has a name. This works perfectly with the improved example mentioned above so you can spend less time coding monotonous Get and Put commands.

Which Should You Use?

So we've seen the Random file access mode in use. And it's pretty darn cool.

But what are the advantages when compared with the sequential Input/Output/Append modes?

Advantages:

- It's easy to change the data already saved in records - you just write a Put command and it will overwrite the old data.
- You don't need to open and close files to change how you read the files (i.e. random does both input and output).

Disadvantages:

- If you want to do anything except simple retrieval of settings or basic data storage, you will need to do a lot of calculations which can be frustrating. Trust me.
- You can't read the data into a text editor, though a simple record editor can be created with a moderate amount of coding

And now for something completely different; binary file mode...

Binary File Mode

The binary file mode is the closest you get to writing direct to disk in Visual Basic. It allows you to directly write to a file at a byte-by-byte level.

To be honest, this mode is pretty limited in its uses, but there are times when it could be helpful.

As with the last mode, we use the **Get** and **Put** statements to manipulate files. This time we have the syntax:

```
Put #filenumber, bytenumber, varname
Get #filenumber, bytenumber, varname
```

You will notice that the difference between using Get and Put for random files is that instead of using a recordnumber as in Random mode, it uses a bytenumber. This "byte number" is the number of bytes into the file that you want to start reading from.

Byte Number	01	02	03	04	05	06	07	08	09	10	11
String Data	H	E	L	L	O		W	O	R	L	D

You can see from the diagram above that every byte can be given a byte number. So, the command...

```
Get #1, 8, MyVar
```

...will place data from byte number 8 into the variable called 'MyVar'. The amount of data put into 'MyVar' depends on the defined size of the variable.

Now there are two ways you can define the length of a variable:

```
Dim varname as String * 50
```

Here, varname is the name of the variable and the number after the asterisk (*) is the length you want the string to be. However, once set, these variables have been declared (Dimmed) cannot be changed in length.

```
varname = String(lengthofstring, 0)
```

Here, varname is the name of the variable you want to set the length of, lengthofstring is the length that you want to set the string. 0 means character zero which is a "null" character.

Going back to the Get statement, using the above diagram you should be able to see that...

```
Get #1, 8, MyVar
```

... would return 'ORLD' to our 4-byte MyVar variable.

Make sense? Once again, the best way to demonstrate this is via example. So fire up Visual Basic and off we go!

Just One More Example

It's now time to check out the Binary file access mode in action!

Open Visual Basic and double-click on "Standard EXE"

You will be left with a blank form.

Now insert the following code:

```
Const ChunkSize = 1024
'This is the amount (chunk) of data to take each
' time the file is written to.
'The larger the quicker but more memory intensive.
'Here I have set it to 1024 bytes which equals 1KB.

Dim Data As String

Public Sub Form_Load

    Open "c:\source" For Binary As #1 'Source
    Open "d:\destination" For Binary As #2 'Destination

    Do Until LOF(1) = Loc(1) Or EOF(1)
    'Will do this loop until the end of the source file.
            Data = ""
            'Resets the size of the Data string.
            MsgBox LOF(1) & vbTab & Loc(1) & _
```

```
            vbTab & LOF(1) - Loc(1)
            If LOF(1) - Loc(1) < ChunkSize Then
            'If there is not enough data left for one chunk.
                Data = String(LOF(1) - Loc(1), 0)
                'Sets the length of the string; Data.
            Else
            'If there is enough data left for one chunk
                Data = String(ChunkSize, 0)
            End If
            Get #1, , Data
            Put #2, , Data
        Loop

End Sub
```

You won't find any strange new commands in this code, but the concepts may be a little difficult to understand at first. Just remember...

LOF(1) - Loc(1) - Remember that LOF(#filenumber) tells you the length of the file and Loc (#filenumber) is the last read byte. So, by simple maths when you take away the current location (which will be how many bytes have been read) from the length of the file, you will get how many bytes of the file is left.

Chunks - I have read data in as chunks, the size of each chunk is set in the constant ChunkSize. The reason I have taken data in as 1 kilobyte chunks is because if you try to make a variable the size of the source file you end up running out of memory.

Anyone for Scraps?

And now for all those other bits I almost forgot.

So far, we've used the Open statement like this...

Open *pathname* For *mode* As *#filenumber* [Len=*reclength*]

... however there are also a couple of other extra widgets that I've not yet mentioned. The official way of using the Open statement is like this...

Open *pathname* For *mode* [Access *access*] [*lock*] As *#filenumber* [Len=*reclength*]

In other words, you have two extra optional arguments there - Access and Lock. Let's explain those now.

[Lock] controls how other programs 'see' your file. [Lock] can be replaced with; Shared, Lock Read, Lock Write, and Lock Read Write, depending on what you want:

Lock Mode	What it Does
Shared	Allows other programs to read and write to this file.
Lock Read	Allows other programs to write and not read to this file.
Lock Write	Allows other programs to read and not write to this file.
Lock Read Write	(Default) Does not allow other programs to read or write to it.

[Access] controls if your program can write, read, or read & write to your file. [Access] can be

replaced with; Read, Write, or Read Write.

Access Mode	What it Does
Read	Allows your program to read from the file
Write	Allows your program to write to the file
Read Write	(Default) Allows your program to both read and write to the file.

[Access] is useful to stop you from corrupting files that you mean to read from (i.e. typing Get instead of Put). It is a similar practice to Option Explicit which forces you to declare variables.

As an example, this statement...

```
Open "TESTFILE" For Binary Access Read Lock Read As #1
```

... opens 'TESTFILE' in Binary mode for reading; other applications cannot read the file due to our lock.

Great, huh?

Conclusion

In this tutorial, you've discovered how to read and write files three separate ways. Talk about value for money!

Sure, it might not seem terribly exciting, but this is a very useful skill often overlook by modern VB'ers.