

Handling Files in Visual Basic

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!

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.

Input, Output and Append mode

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 **opened** 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.

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 variable 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*).

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