File Operations in Visual Basic 6

I'm frequently asked how to perform file operations in Visual Basic--for instance, how to delete a file or create a directory or folder.

In this month's article, I'm going to show you five basic file type operations using built in Visual Basic functions. These techniques are considered old style by programmers familiar with Object Oriented programming who prefer to use the File System Object (FSO). But a discussion of the F SO requires a knowledge of Objects and Collections---perhaps I'll discuss FSO in a future article, provided you promise to read my Objects book, Learn to Program Objects with Visual Basic 6.

But back to File Operations using the built in Visual Basic functions to which I alluded. We can categorize these operations in two ways: operations on files and operations on directories, or the newer term, folders.

Copying Files

When you copy a file you keep the existing file, and make a copy of the file with a new name. Visual Basic provides a statement for this called FileCopy.

Here's the syntax for the FileCopy statement

FileCopy *source*, *destination*

where source is the name of the file to copy, and destination is the name of the copied file.

You have several choices when it comes to specifying the file names here---you can use the full path names for the files, or you can just specify the name of the files.

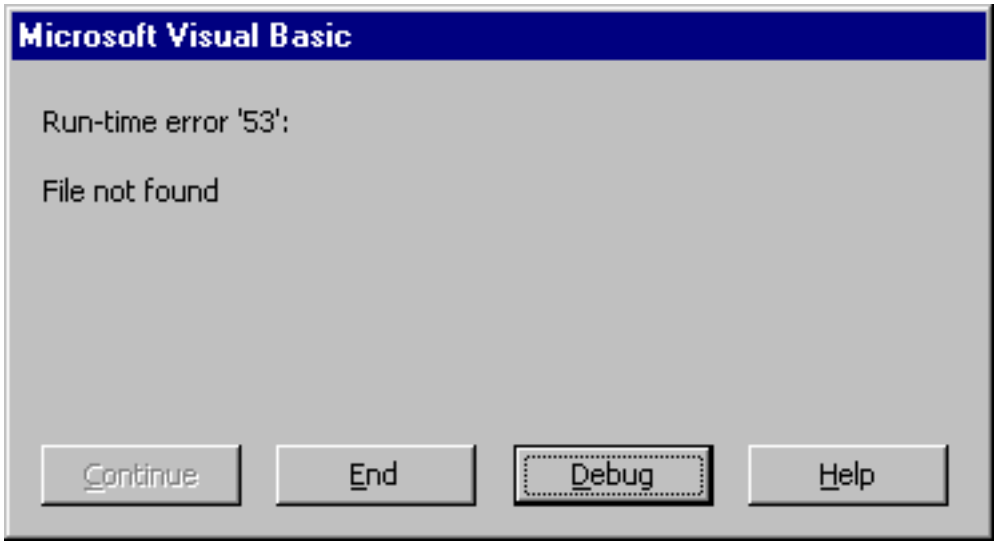
By way of background, Windows keeps track of something called the current drive and the current directory for us---these are basically pointers in the File System, and in the old days of DOS allowed us to perform mundane file operations without having to specify the name of the Drive and the Directory. These pointers still carry on in VB and Windows, so if we use this syntax in the Click Event Procedure of a Command Button

```
Private Sub Command1_Click()

FileCopy "a.txt", "b.txt"
```

```
End Sub
```

Windows looks for a file called "a.txt" in the current drive and current directory of our PC, and ***if*** the operating system finds it, copies the file as "b.txt", again in the default drive and directory of the PC.

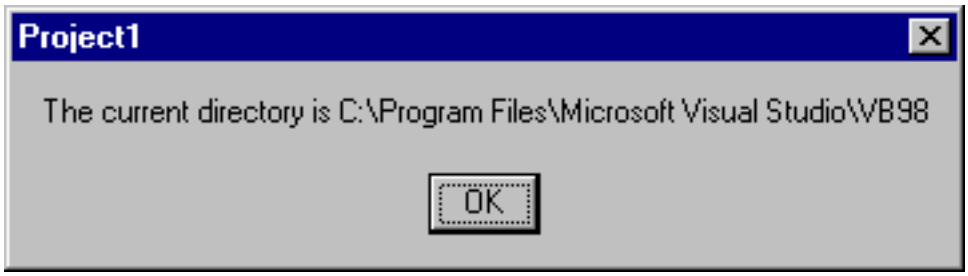
The problem here is that if the file is not found, your program bombs, just like this…



The Current Drive and Current Directory

As it turns out, Visual Basic has a function that can be used to determine the current directory called the CurDir function …

```
Private Sub Command2_Click()

MsgBox "The current directory is " & CurDir

End Sub
```



Changing the Current Drive and Current Directory

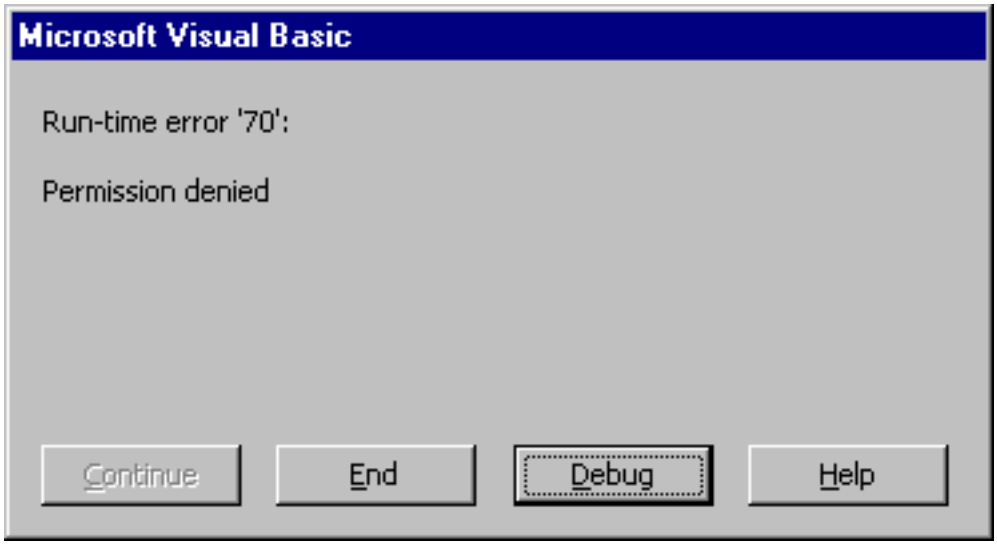
Once you know the current directory, you can then use the ChDir and ChDrive functions to change either the current drive or the current directory, like this…

```
Private Sub Command3_Click()

ChDrive ("d")
ChDir "\vbfiles"

MsgBox "The current directory is " & CurDir

End Sub
```

Now if you are like me, you may not want to leave anything to chance, in which case, you can use the full path name with the FileCopy statement, like this…

```
Private Sub Command1_Click()

FileCopy "c:\vbfiles\a.txt", "c:\vbfiles\b.txt"

End Sub
```

I should mention that here that if you attempt to copy a file that is opened, you'll receive this error message…



Does a file exist?

There's no confirmation that the copy was successful, but you can determine if a file exists by using the Visual Basic Dir$ function. The Dir$ function requires just a single argument representing the file name (as was the case with the CopyFile statement, you can specify just the file name or the full path name). If the file is found, then Dir$ returns the file name (not the full path). If the file is not found, then Dir$ returns an empty string. Let's see how we can use the Dir$ function to determine if a file exists before we copy it.

```
Private Sub Command1_Click()

Dim retval As String

retval = Dir$("c:\vbfiles\b.txt")

If retval = "b.txt" Then
  MsgBox "b.txt exists--no need to copy it..."
Else
  FileCopy "c:\vbfiles\a.txt", "c:\vbfiles\b.txt"
End If

End Sub
```

If we now run the program, and click on the command button…

We receive a message saying that the file already exists---Dir$ has done its job.

By the way, you'll discover that there's a Dir function as well---Dir returns a variant return value and Dir$ returns a string.

Renaming Files

Renaming files is similar to copying them--this time we use the Visual Basic Name statement. Here's the syntax.

Name oldpathname As newpathname

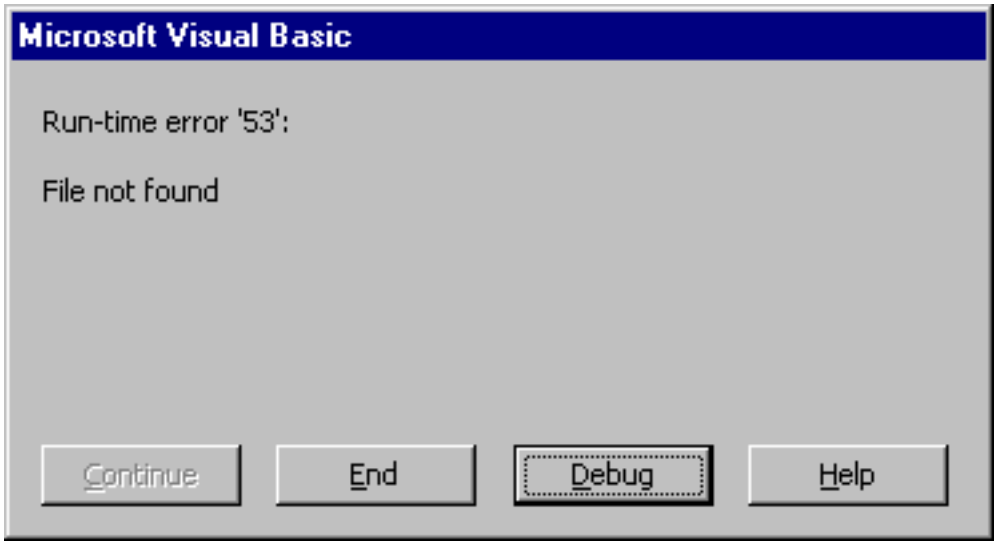
As was the case when we copied files, we can choose either to specify a file name or to include the full path name---once again, I advise the full path name…

Private Sub Command1_Click()

Name "c:\vbfiles\b.txt" As "c:\vbfiles\newb.txt"

End Sub

This code will result in the file 'b.txt' begin renamed to 'newb.txt'. Once again, don't expect a confirmation message telling you that the rename was successful--the only message you'll receive is if the file does not exist



Deleting Files

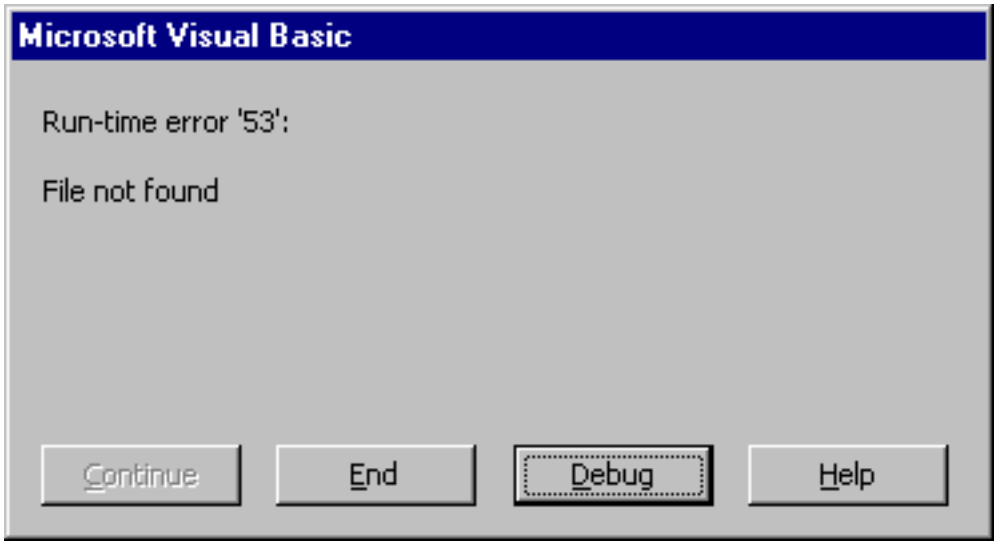
The final file operation I'll discuss in this article is that of deleting a file. Visual Basic provides us with the Kill statement which will delete a file (and dangerously, a wildcard selection of files) of our choosing. Here's the syntax…

Kill *pathname*

Let's say that we wish to delete the file "newb.txt' file that we created just a few minutes ago. This code will do the trick…

```
Private Sub Command1_Click()

Kill "c:\vbfiles\newb.txt"

End Sub
```

Again, there will be no confirmation message, only an error message if the file we are attempting to delete does not exist.



As I mentioned, you can also use wildcards as an argument to the Kill statement (WARNING: Don't attempt this at home!!!). For instance, this code will delete EVERY file in the \VBFILES directory that has a file extension of *.txt…

```
Private Sub Command1_Click()

Kill "c:\vbfiles\*.txt"

End Sub
```

Most dangerously, this code will delete EVERY file in the \VBFILES directory…

```
Private Sub Command1_Click()

Kill "c:\vbfiles\*.*"

End Sub
```

Be careful when using the Kill statement---when issued through Visual Basic, there's no going back. There is no Undo statement, and files deleted in this way are NOT moved to the Windows Recycle bin.

Moving Files

There is no explicit Visual Basic statement to move a file. To simulate a move of a file, all we need to do is combine the FileCopy and Kill statements that we've already seen. For instance, to move the file a.txt from C:\VBFILES to C:\VBILES\CHINA, we can execute this code…

```
Private Sub Command1_Click()

FileCopy "c:\vbfiles\a.txt", "c:\vbfiles\china\a.txt"

Kill "c:\vbfiles\a.txt"

End Sub
```

Again, don't expect any confirmation messages---only errors if the files you reference do not exist.

That's it for Visual Basic actions that we can take against files---now it's time to turn our attention to Directory or folder operations.

Creating a Directory (Folder)

Creating a Folder is something that we're used to doing using Windows Explorer, but Visual Basic gives us the capability of creating folders within our program using the MkDir statement. Here's the syntax…

MkDir *path*

where path is either the name of a folder to be created, or (better yet!) the full path name of the directory or folder that you wish to create.

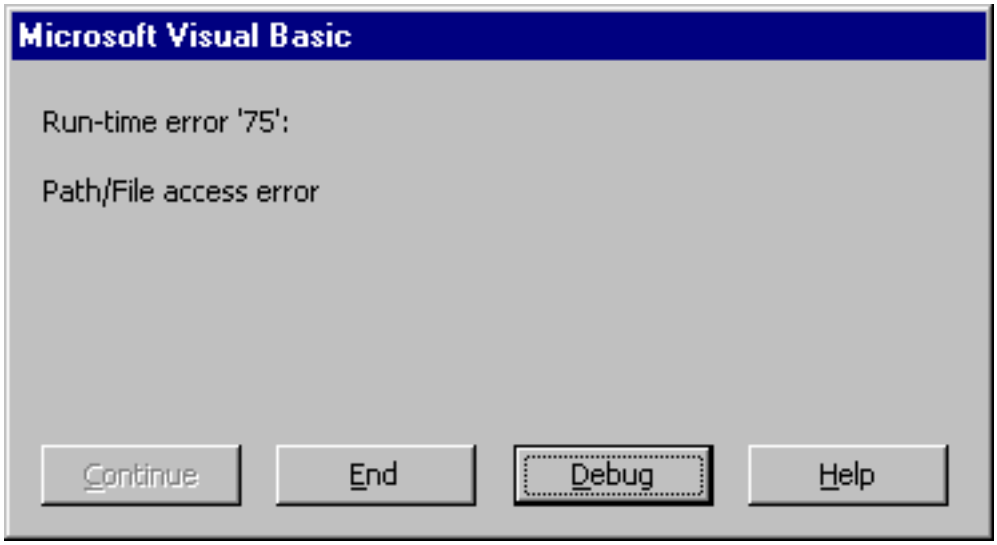
Specifying just the folder name to be created can be dangerous---if you are not aware of the current drive and directory, you may wind up creating a folder ***somewhere*** on your hard drive, with no real idea where it went. Better to be sure and specify the full path name, like this

```
Private Sub Command1_Click()

MkDir "c:\vbfiles\smiley"

End Sub
```

This code will create a folder called 'smiley' within the folder 'vbfiles' on the C Drive. Once again, you'll receive no confirmation message if the folder is created, but you will receive an error message if the folder creation fails.

There are two potential errors when executing the MkDir statement.

First, if you attempt to create a folder that already exists, you'll receive this error message

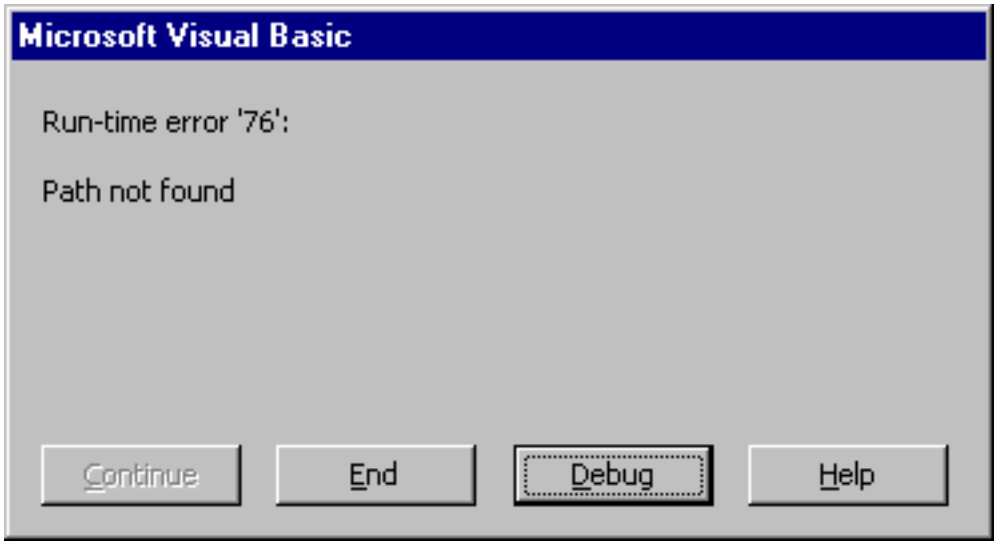


The error message is not explicit enough for my liking, but that's what it means---the folder 'smiley' already exists.

A second possible pitfall is attempting to create a folder within a folder that itself does not exist. For instance, in this code

```
Private Sub Command1_Click()

MkDir "c:\vbfiles\smiley\one\two"

End Sub
```

if the folder 'one' does not yet exist within 'smiley', you can't create the folder 'two'--and you'll receive this error message…



Removing a Directory (Folder)

Removing a directory is similar to removing a file---in this case, we use the Visual Basic RmDir statement. Here's the syntax…

RmDir *path*

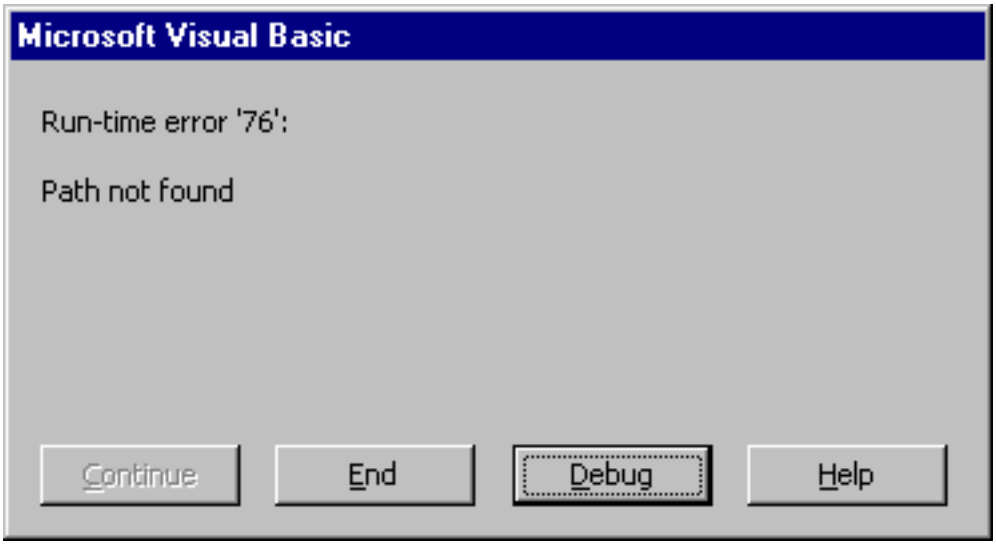
As was the case with the MkDir statement, path can either be a file name or the full path of a file name (once again, my recommendation). This code will remove the folder 'smiley' that we just created …

```
Private Sub Command1_Click()

RmDir "c:\vbfiles\smiley"

End Sub
```

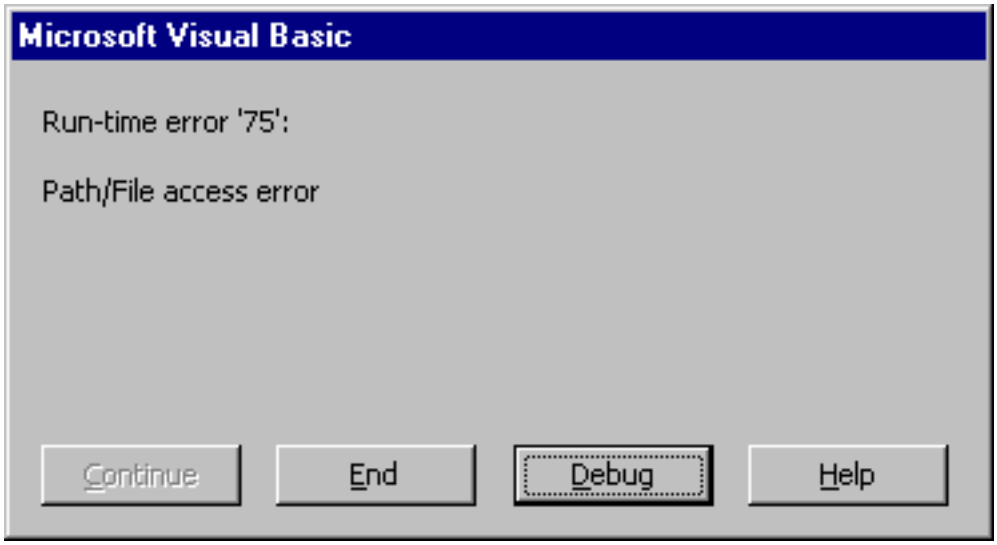
It should come as no surprise to you that there's no confirmation message generated for a successful removal of the folder.

Possible error messages from RmDir?

There are two pitfalls. First, as we've seen all along, if you attempt to remove a folder that does not exist, you'll receive this error message…



A second possible error can occur if you attempt to remove a directory or folder that contains files. If you try, you'll receive this error message…



You must first use the Kill statement to remove every file from the folder before executing the RmDir statement (that's where the wildcard for the Kill statement comes in handy!)

Moving a Directory (Folder)

As was the case with moving files, there is no explicit Visual Basic statement that will do this for you.. To move a folder (and everything along with it), you'll first need to create the new folder, then use FileCopy to copy all of its files to the new folder, then delete all the files in the old folder using the Kill statement, and finally remove the old folder.

Summary

The need to work with directories (folders) and files can arise during your Visual Basic programming career---I hope this overview of the Visual Basic file and folder statements will help you.

As I mentioned at the beginning of the article, the File System Object (FSO) can also be used to do everything that you've seen here--but it's available only in Visual Basic 6, and it requires a comfort level with Objects and Collections that you may not yet have.

If there's a demand for it, I'll be glad to address it in an upcoming article.