VB6: Using Files to save Structures

Tutorial 01

By:
Hugo Ferreira

September 3rd, 2001
Version 1.0
Best viewed with Microsoft Word for Windows 97 or better.

Web Page:
http://unitek3000.tripod.com/

Any Doubts?
Lotsjunk@hotmail.com

Length of Tutorial:
You should take one hour off to complete this tutorial.

Necessary Software:
You need Visual Basic 6 to complete this tutorial.

Files:
This tutorial has no accompanying files.

Message:

These Tutorials take time to make. The authors took all the necessary steps to ensure the Tutorials are accurate and fault-free.

Even so, once in a while, errors get through. It is, in my opinion, unavoidable.

So, to all of you reading this, in case you detect an error in the tutorial, please email us so we can correct it.

Also, be sure to email the authors with your doubts and questions. These questions will enable them to write better and better tutorials.

And, any suggestion you have for us… well, you know where to reach us.

The team thanks your cooperation.

Files… who doesn't need them?

I don't care which programming language I'm learning.
Soon enough I'm gonna need file handling capabilities.
If you're here, reading this, I guess you want too.

This is one of my first tutorials, so I'm still getting the hang of this.

I think there are many tutorials out there explaining how to write text ("hello world"?) to a file in VB, but that isn't what we are here for.
We want to write raw binary data, and full, unrestricted access to everything inside that file.

When I'm trying to learn something, I like the teacher to talk to me as if I was a 3 year-old, and that's how I think other people like to be taught.

hey boys hey girls here we go (homage to the Chemical Brothers)

First I'm gonna teach you structures under Visual Basic.
Do define a structure, you use the Keyword "Type", and you write the variables you want inside the structure.

```
Public Type AlphabetStructure
   Alpha As Long
   Beta As String
   Lambda As Boolean
End Type
```

Defining the structure's inner layout isn't enough, you need to declare variables that will take on the form of that structure.

```
Public Alphabet As AlphabetStructure
```

Ok, now we have a variable "Alphabet" that uses the TYPE we declared earlier.

To alter any one of the variables inside Alphabet, you just type "Alphabet" followed by the biblical dot:

Ok, now you have access to all the variables inside the Alphabet structure.

What does this got to do with files?
Well, housekeeping file formats can get really ugly really fast, so, IMHO, we should access a file's contents using a structure.

Ok, so lets now jump onto the topic of Files!

Here's a code dump:

```
Public Sub WriteAlphabet()
Dim FHandle As Integer

FHandle = FreeFile

Open "File.Ext" For Binary Access Read Write As FHandle
Put FHandle, , Alphabet
Close FHandle

End Sub
```

Let's examine this closely. FHandle is an integer type variable, which gets assigned to FreeFile, but what the heck is FreeFile?
Well, in the software world, whenever you request a certain service, it is usual to receive a handle. FreeFile is a VB internal function, which checks if there is a free file handle, and returns it, in the form of an integer.

You need that handle to open one file, work on it, and close it.
One handle per file, that’s the rule.

Next comes the Open statement itself. As I stated in the beginning of this tutorial, we would be working without restrictions, and that's what we get with Binary file access type.

```
Open "File.Ext" For Binary Access Read Write As FHandle
```

When working on files, you must declare if you want to read them, write them, or have permission to do both things.
In our Open statement I declared the "Access" to be "Read Write", this will allow us to both read and write to the file, giving us complete access.

Next to "Open" comes the file name. This should be obvious.

Lets now look at the Put statement:

```
Put FHandle, , Alphabet
```

Alphabet is our structure. We inserted all the values we wanted into it. Now comes the time to save our work in a file, for posterity.

So, for the purpose of dumping structures onto files, we have the Put statement, which will require our file handle, and a structure (duh!).

After this statement is executed, the information inside our structure will be saved on the file.

The next part is just simple common house keeping procedures, in this case, closing the file.

```
Close FHandle
```

Here we see the file handle in action again, for the last time.

Ok, so our work is now safely saved inside a file. How do we retrieve it? Take a look at another code dump:

```
Public Sub ReadAlphabet()
Dim FHandle As Integer

FHandle = FreeFile

Open "File.Ext" For Binary Access Read Write As FHandle
Get FHandle, , Alphabet
Close FHandle

End Sub
```

See the difference? No? Let me spot it out for you, it’s the "Get" statement. Get does the inverse of Put, it reads data from a file and inserts its contents into a structure. Pretty useful, no?

After the Get statement gets executed, the data you saved onto the file will be inside the structure again.

Well, we've traveled full loop.
I hope now you can go out and start doing some file saving of your own.

There is a FAQ below, but I say goodbye now, email me if you want a tutorial on another aspect of VB.
See you all on cyberspace,

Hugo Ferreira
Lotsjunk@hotmail.com

FAQ - Frequently Asked Questions (email us for more questions)

Q1: How do I know if a file already exists?
A: Here's a code dump, check it out:

```
Public Function FileExists(FName As String) As Boolean
Dim FH As Long
FH = FreeFile

On Error GoTo errorhandler
Open FName For Input As FH
Close FH

FileExists = True
Exit Function

errorhandler:
FileExists = False
End Function
```

Let's go slowly on this one. FH will store a temporary file handle. This function accepts as input a file name, and returns either true (file already exists) or false (it does not).

On Error GoTo errorhandler
This tells VB "hey, if anything weird happens go to the place I defined".

Next, we try to open the file, if it does not exist, or anything else weird happens, VB will jump to "errorhandler", else it will normally continue.

If no error occurs, we close the file we just opened, assign the function as true, and get out,

Otherwise,

We just ended up on "errorhandler", we assign the function as false (yep, no file with this name exists, or there is something major wrong going on), and we get out.

This code dump also lets you see basic error trapping in action.