Understanding The Windows Scripting Host

Automating common tasks for users is relatively easy with the Scripting Host built into Windows.

By Julian Moss

One of Windows's most notable deficiencies when compared with other operating systems is its lack of a batch language for automating tasks. Although the underlying MS-DOS supports batch files, they are of little use in the Windows environment: under Windows 3.1 you can't even launch Windows programs from batch files.

Support staff wishing to automate tasks for their users have been forced to use third-party tools like Wilson WinBatch or JP Software's Take Command. Because these tools are not part of Windows, they must be separately installed before they can be used. Licensing of these tools means that additional costs are often incurred, which, if the budget is not available, becomes a problem.

This limitation of Windows was removed with the advent of the Windows Scripting Host (WSH). Available since the release of Internet Explorer 3.0, Microsoft originally kept quiet about the facility. This is no longer the case, and the WSH is one of the standard accessories to be included in Windows 98. It can be also be downloaded from www.microsoft.com/scripting to install on Windows 95 and NT 4.0 [*and is lso on this month's CD-ROM - Ed.*]. Documentation for the Microsoft script engines can be viewed online at http://www.microsoft.com-/scripti ng/start.HTM.

```
' Hello world! Example script
MsgBox("Hello world!")
```

Figure 1 - The simplest VBS script.

Scripting Architecture

Internet Explorer 3.0 introduced the capability of client-side scripting, using script code embedded in HTML pages. Instead of making this a dedicated feature of Internet Explorer, Microsoft designed the facility in such a way that scripting could be incorporated into other applications as well. The application - in this case Internet Explorer - is the scripting host, and exposes various COM objects that a script can manipulate.

In the WSH, the objects that are provided allow scripts to control certain features of the operating system. Microsoft produces one other scripting host: Internet Information Server (IIS), which uses the same script language for server-side scripting. Other developers may also write applications that are scripting hosts since Microsoft has published the specification for doing so.

Microsoft has also documented the interface for script engines. This will allow developers to write engines for other script languages. Once the engines are registered with the system these script languages can be used in any scripting host. Microsoft has produced two script languages, VBScript and JavaScript. Engines for other popular script languages like Perl and Rexx will doubtless soon appear. In this article we will focus on using VBScript, but all of the examples given here could be implemented equally well using JavaScript.

```
' Example showing how to obtain command line arguments
Dim Args, ArgList
Args = WScript.Arguments
For i = 0 to Args.Count - 1
    ArgList = ArgList & i & ": " & Args(i) & chr(13)
Next
MsgBox "No. of arguments: " & Args.Count & chr(13) & ArgList
```

Figure 2 - ARGS.VBS - Command line arguments.

Independence

The key point about the scripting architecture is that script engines and scripting hosts are independent of one another. Script engines may incorporate some functionality of their own, but the greater part of the power of scripting comes from the script's ability to use the objects exposed by the host. Scripts can also create OLE Automation objects of other applications on the system such as Microsoft Word and Microsoft Excel, which allows them to control these applications.

Scripting is a simple and very powerful way to automate tasks under Windows. You can create scripts using a text editor such as Notepad, and save them to disk as plain text files. When you run a script, the file extension of the script file tells the Windows Scripting Host what language has been used. VBScript scripts must have the extension .VBS. JavaScript scripts use the extension .JS.

The default Open action associated with script files is to run the script. To run a script you (or a user) needs simply to double-click its icon. To have a script run automatically when Windows starts just create a shortcut to it in the StartUp group. To make changes to a user's system you could send the script to the user as an attachment to a mail message and have them run it by opening the attachment. Scripts can even be run from batch files or a command prompt by running the program CSCRIPT.EXE with the script file name as the first parameter to the command.

Writing Scripts

Scripts can be very simple. Just like a batch file, a script can consist of a sequence of commands to be executed. The simplest possible script is shown in Figure 1 which, following the time-honoured tradition for programming texts, displays the message "Hello world!" The example is twice as long as it needs to be because the first line is a comment.

As you can see from this example, scripts need no additional structure commands. Script commands are executed sequentially from start to finish. However, within a script you can define BASIC subroutines and functions using the standard syntax Sub/End Sub or Function/End Function. These subroutines and functions are ignored on the first pass through the script file, and only executed when they are called by script commands.

```
Dim Dict                   'Create an instance
Set Dict = CreateObject("Scripting.Dictionary")
Dict.Add "a", "Athens"     'Add some keys and items
Dict.Add "b", "Belgrade"
Dict.Add "c", "Cairo"
Dict.Add "d", "Doncaster"
Dict.Add "e", "Eastwood"
key = InputBox("Enter a letter a - e")
MsgBox "Item selected: " & Dict.Item(key)
```

Figure 3 - DICTIONARY.VBS - Using a Dictionary object.

```
' update one file with another
' if modification time is later
Function Update( source, target )
  Dim f1,f2,d1,d2,c1,c2
  if fs.FileExists( source ) then
  ' source file is accessible
    set f1 = fs.GetFile( source )
    d1 = f1.DateLastModified
    c1 = Year(d1) * 10000 + Month(d1) * 100 + Day(d1)
    if fs.FileExists( target ) then
      set f2 = fs.GetFile( target )
      d2 = f2.DateLastModified
      c2 = Year(d2) * 10000 + Month(d2) * 100 + Day(d2)
    else
      c2 = 0
    end if
    if c1 > c2 then
      ' overwrite local copy with new version
      f1.Copy target,True
    end if
  end if
End Function
' begin script execution
Dim fs
set fs = WScript.CreateObject("Scripting.FileSystemObject")
s = "\\Server\Server_c\AVP\sigfile.dat"
t = "C:\AVP\sigfile.dat"
update s, t
```

Figure 4 - UPDATE.VBS - Update local file from server

Omissions

The VBScript language will be familiar to anyone who has used Visual Basic or Visual Basic for Applications (VBA.) However it is not a complete implementation of VBA. Microsoft has made a number of simplifications in order to produce a script language that is both lean and fast. For example, there is no GOSUB or WITH statement, and the On...Goto construct is not supported. Arrays must be zero-based. There is no Static keyword. The DDE Link commands are not supported, and you cannot access the Clipboard from a script.

Additions

VBScript has several features of its own that are not found in VBA. There is a special group of functions for formatting numbers, currency and date values, and a group of constants for obtaining the current date and time. New string processing functions include Join and Split for converting between arrays and strings of delimiter-separated values. To help you avoid compatibility problems there are functions for obtaining the version of the script engine in use.

VBScript is not an alternative to full-blown programming languages, even if you are prepared to live with the fact that scripts are written in plain text for all to see and tinker with. You cannot create forms or dialog boxes, so the possibilities for creating scripts that interact with the user are limited. You can display message boxes with Yes/No or OK/Cancel buttons, and a rather clunky input box that displays a prompt and allows the user to enter some text. If your application requires a greater degree of interaction than that you will need to fire up Visual Basic.

Functionality specific to the Windows Scripting Host is accessed using its primary object WScript. This object is always present, and so does not have to be created. It provides several important methods such as CreateObject which is used to create instances of other objects, and Quit which allows you to terminate a script before the end of the script file.

The WScript object has an Arguments property which can be used to obtain the command line arguments passed to a script. This is illustrated in Figure 2, which displays the arguments in a message box. Command line arguments for a script can be specified following the script filename in the Target field of a shortcut, or when running a script from a command prompt using CSCRIPT. Unfortunately scripts are not treated by Windows as an executable file type so you can't pass filenames to them dynamically by dragging files and dropping them on a script icon or shortcut.

VBScript provides two types of object, defined by the built-in Scripting object, which you are likely to find useful in your scripts. Dictionary objects can be used to hold indexed lists of data in a manner similar to a Perl associative array. Figure 3 gives a brief taste of how they are used.

File Access

The FileSystemObject object is one of the most important objects available to your scripts. It allows them to carry out file management activities like testing whether files or folders exist and copying, deleting and moving files and folders. FileSystemObject objects have methods that themselves return objects representing drives, folders and files. These objects can be used to get information such as the attributes of a folder or the size and modification date of a file.

FileSystemObject has too many properties and methods to cover in detail here, but an example of its use is shown in Figure 4. This script, or a derivation of it, could be launched from a user's StartUp group, and will run silently in the background when the PC starts. It compares the modification date of a file on a network server with the modification date of the same file on the local hard disk. If the server copy is newer it is copied to the local system. A script working along these lines could be used to update the signature files for a workstation's anti-virus software from copies kept on the server.

Walk-Through

A brief explanation of the code in Figure 4 may be beneficial. Note that the code for updating the file has been written as a function. This makes it a generic function that could be called multiple times with different arguments to update a number of different files if you so wished. Execution of the script begins by declaring and creating the FileSystemObject object which is needed in order for many of the functions used by the Update function to become available. Variables s and t are initialised to the source and target filenames. This isn't necessary: the filenames could be specified as constants when the function "update" is called.

The function first checks to see if the source file exists. If it cannot be found for any reason - even if the server is inaccessible - the FileExists function simply returns False and nothing more is done. If the file is found then a file object f1 is created for it. This is used to obtain the file's modification date. Because this is returned as a text string in d1 it must be converted to an integer value c1 for comparison. As we are only interested in whether one file is newer than another, and not by precisely how many days, a simple conversion to the form yyyymmdd is used.

The modification date d2 for the second file is found in a similar way. If

```
Dim WSHShell, fs
Set WSHShell = WScript.CreateObject("WScript.Shell")
Set fs = WScript.CreateObject("Scripting.FileSystemObject")

Function MakeDesktopShortcut( name, target )
  Dim Shortcut,DesktopPath,StartupPath
  DesktopPath = WSHShell.SpecialFolders("Desktop")
  Set Shortcut = WSHShell.CreateShortcut(DesktopPath & "\" &
                                       name & ".lnk")
  Shortcut.TargetPath = target
  StartupPath = fs.GetParentFolderName( target )
  If fs.FolderExists( StartupPath ) then
    Shortcut.WorkingDirectory = StartupPath
  End If
  Shortcut.Save
End Function

MakeDesktopShortcut "Shortcut to Notepad", "C:\Windows\-
                                            Notepad.exe"
```

Figure 5 - SHORTCUT.VBS - Create a shortcut on the desktop.

```
Dim WSHNet, fs
Set WSHNet = WScript.CreateObject("WScript.Network")
set fs = WScript.CreateObject("Scripting.FileSystemObject")

Function MapDrive( letter, UNCname)
  If fs.DriveExists( UNCname) Then
    WSHNet.MapNetworkDrive letter, UNCname
  End If
End Function

MapDrive "E:", "\\DARWIN\DARWIN_C"
MapDrive "F:", "\\DARWIN\DARWIN_D"
```

Figure 6 - MAPDRIVE.VBS - Map network drives.

the file does not exist at all the value for comparison c2 is set to 0. The two values are then compared, and if c1 is greater than c2 file f1 is copied to the target filename, overwriting the previous version if it existed.

Two other methods of the FileSystemObject object which we won't go into here, OpenTextFile and CreateTextFile, create TextStream objects. These objects have methods that allow you to read from and write to text files in a serial manner. It is not difficult to see how you could enhance the functionality of the code in Figure 4 by using TextStream objects to read a file containing a list of the files to be updated or write a log of updates carried out.

A by-product of this capability is that it would be possible to write scripts that create or modify other scripts. Whilst this is unlikely to be generally very useful, it does seem to make the possibility of script file viruses at least technically possible.

Shell Object

Besides the objects that are provided by VBScript and hence available to scripts running under Internet Explorer or IIS, two important objects are provided by the WScript object of the Windows Scripting Host. The Shell object has a number of useful functions. It allows you to find out the location of special folders such as the Windows desktop or Start Menu, to obtain the contents of MS-DOS environment strings, to run programs, pop up message boxes (a function that is more or less duplicated by the VBScript MsgBox function), create shortcuts and read from, write to or delete keys in the Windows Registry.

Figure 5 illustrates some of the features of the Shell object. It contains a generic function MakeDesktopShortcut which, as the name suggests, creates a shortcut on the user's desktop. The function takes two arguments: name, which is the name of the shortcut, and target, which is the filename or URL which is launched when the shortcut is double-clicked.

The function uses the SpecialFolders method of the Shell object to obtain the location of the Windows desktop for the current user. It then creates a shortcut object using the Shell method CreateShortcut. The shortcut's TargetPath property is set to the filename or URL. Two FileSystemObject methods, GetParentFolderName and FolderExists, are used to set the working directory to the folder containing the target. If this is an Internet URL the result of GetParentFolder is invalid, so the WorkingDirectory property is only set if the folder actually exists. The Save method of the Shortcut object must be used to actually create the shortcut file on the hard disk.

Network Object

The other object provided by WScript is the Network object. This lets you find out information like the user name, computer name and organisation name. It also lets you connect and disconnect network drives, add and remove printer connections and set the default printer.

Figure 6 is a short script that maps network drives if they are accessible. In some situations this may be preferable to the normal Windows mechanism which displays a warning message if it is unable to connect to a drive. However, the real value of the example is to illustrate how it can be done so that you can create scripts that map drives using a degree of intelligence.

The function MapDrive uses the FileSystemObject method DriveExists to test whether the network volume is accessible using its UNC name. If the drive is found, the MapNetworkDrive method of the WScript.Network object is used to make the connection. Without the test, the script would fail with an error message if the network volume was not accessible when it tried to map it.

OLE

As mentioned earlier, scripts can create and use automation objects exposed by any Windows application that supports OLE Automation. This includes all the major Office suites and many other applications that claim to be Microsoft Office compatible. Though many of these applications may have script languages of their own, it may often be more convenient to script them using a standalone, application-independent script language. Figure 7 shows a very trivial illustration of how to automate Microsoft Word by creating a new document, inserting some text in it, saving the document and closing it. The purpose of the example is simply to show that it can be done, using exactly the same commands as would be used in Visual Basic or VBA.

Conclusion

The Windows Scripting Host is the tool that Windows has needed since its inception. It fulfils admirably the function of a batch language and automation tool, whilst its ability to automate other applications extends its power tremendously. Support professionals will undoubtedly find it to be a valuable tool. But many advanced users may also enjoy the facility to write scripts that make their time at the keyboard more productive.

PCSA

```
Dim MSWord, WSHShell
Set WSHShell = WScript.CreateObject("WScript.Shell")
Set MSWord = WScript.CreateObject("Word.Basic")
MSWord.FileNew("Normal")
MSWord.Insert("The quick brown fox jumps over" & Chr(13))
MSWord.Insert("the lazy dog.")
MSWord.FileSaveAs(WSHShell.SpecialFolders(12) & "\Test.doc")
MSWord.FileClose
```

Figure 7 - WORD.VBS - Automating Microsoft Word

The Author

Julian Moss is a freelance technical writer and software developer. He can be contacted by email as jmoss@cix.co.uk.