

O N E

# Windows Script Host Fundamentals

*The Windows Script Host, or WSH for short, is one of the most powerful and useful parts of the Windows operating system. Strangely enough, it is also one of least well known, and is used by only a small percentage of Windows users. This strange set of circumstances is due in large part to the fact that Microsoft has not publicized the WSH in the same way that other parts of Windows, such as its Web capabilities, have been publicized. It is also because DOS and Windows users, unlike the Unix community, have no history of a powerful scripting language and therefore usually don't even think to look for one. Trust me — once you learn how to use WSH, you'll wonder how you ever got along without it! In this first chapter we take an introductory look at WSH: what it is, how it works, and what you can do with it. Then we create a real, working script that you can put to use on your system right away.*

## Background

Computers are supposed to make your life easier, right? Generally they do, but this does not mean computers are immune from errors. Many errors could be avoided by automating the task at hand so that it runs with little or no need for user intervention. This general idea applies not only to the typical application programs that you run, such as word processing and graphics, but also to the computer's operating system itself. If some of the operating system



tasks can be automated, particularly those tasks that are repetitive and need to be performed regularly, that will be an additional increase in efficiency. Almost from the very beginning, when JCL (Job Control Language) and REXX were developed for mainframe computers, this has been a concern of operating system designers. Let's take a brief and incomplete look at the history of scripting languages.

## Unix Scripting

The Unix operating system, in its various implementations, has been around for a long time, and recently has experienced a sort of renaissance with the popularity of the open source code Linux version. From its beginnings, Unix provided a powerful scripting language that could be used to automate a wide variety of tasks. First came CScript, and then the more powerful and better known Perl and Tcl, both of which are full-featured programming languages with support for variables, iteration, regular expressions, and looping (I'll explain these terms soon). The things that could be done with these languages left many non-Unix users green with envy.

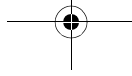
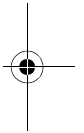
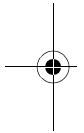
## The DOS Batch Language

When the DOS operating system was first introduced along with the first PCs, back in the early 1980s, it was provided with an extremely elementary scripting ability in the form of batch files. Batch files were simple text files that contained a combination of operating system commands and simple programming constructs. When you ran a batch file the effect was the same as if you had typed those commands at the command prompt. Likewise, if a batch file named autoexec.bat was present, it would be run automatically each time the system was booted.

Batch files were widely used to automate a variety of tasks, but they were extremely limited when compared with the scripting capabilities of Unix. These limitations are as follows:

- No support for looping or iteration (although some inelegant kludges could be used to implement limited functionality)
- Very limited variable support
- No regular expressions
- No direct support for structured programming (procedures)

Many DOS users managed to get a lot out of batch files, despite their inherent limitations. Not having experience with a "real" scripting language, they didn't know what they were missing!



## Windows Scripting

In the early days of the Windows operating system, the term "Windows scripting" was a contradiction in terms because there was no Windows scripting — not as part of the operating system at least. The old DOS batch file capabilities were carried over into Windows, but given the additional features and capabilities of the new operating system the limitations of batch files were even more problematic. The Perl scripting language was available as a third party product, but only a small percentage of users knew about it. This shortcoming continued until the mid 1990s, when Windows 95 was the latest version of the operating system. At this time, Microsoft introduced the Windows Script Host as a powerful, full-featured scripting language for command scripting with Windows 95. Finally, after too many years, users of PCs with the Microsoft Windows operating system had a true command scripting language at their disposal. There was no more need for Unix envy!

The initial release of WSH was configured as an add-on to Windows 95, rather than being an integral part of the operating system. In other words, you had to download the WSH files and install them before you could write scripts. With Windows 98, Windows NT 4.0 (with the NT option pack), Windows ME, and Windows 2000 the WSH is included as part of the operating system.

### Downloading WSH

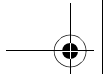
If you are running Windows 95 and do not already have the WSH components installed, you can download them from <http://msdn.microsoft.com/scripting/>.

## How WSH Works

I think you'll agree that WSH sounds intriguing, but you probably want to know more of the details. What can WSH do? How does it work? How do I create and run scripts? Most of this book is devoted to answering these and other questions in detail, but it's a good idea to have an understanding of the overall picture before getting to those details. That's what we do in this section.

### What Can WSH Do?

An obvious question to ask at the start is what can be done with WSH? To be honest, it might almost be easier to list the things you *cannot* do with WSH! It is a very powerful and flexible tool, and in effect provides you with most of the functionality of a full-featured programming language combined with the



capabilities that are built into the Windows operating system. If you have some familiarity with the old DOS batch files, it is definitely a mistake to think of WSH as simply a souped-up version of batch files. You might as well think of a Ferrari 360GT as a souped-up tricycle! Here are just some of the tasks that can be easily and efficiently performed using WSH:

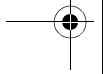
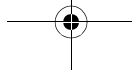
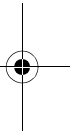
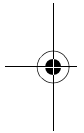
- File management tasks, such as moving, copying, deleting, and renaming files and folders
- Reading data from, and writing data to, text files
- Scheduling scripts to run at specified times
- Administering user logins, operating system settings, and file short-cuts
- Automating the administration of Microsoft SQL Server and Internet Information Server
- Accessing the capabilities of the programs in Microsoft Office
- Working with information stored in databases
- Leveraging the power of ActiveX components
- Automating messaging tasks

This is only a partial list, but I think you get the idea. WSH was designed from the outset to impose as few limitations as possible, and to put the maximum power and convenience in the hands of the users.

## WSH Operation

At the heart of WSH is a set of *scripting engines*. These are the same scripting engines that are used by the Microsoft Internet Explorer browser to execute client-side scripts, and by Microsoft Internet Information Server (Microsoft's Web server software) to execute Active Server Page scripts. WSH provides a different environment in which these scripting engines can run, an environment designed for local user execution of scripts.

You may have noticed that I said "engines." Is there more than one scripting engine for WSH? There is indeed. WSH was designed from the outset to support different scripting languages, with each language supported by a separate scripting engine. As supplied by Microsoft, WSH supports two languages (as detailed in the next section) but the WSH specification is open to the development of other engines by third-party vendors. As an analogy, think of a mail-order computer store that has telephone representatives who can speak English, Spanish, and Chinese. Depending on the language you want to speak, your order will be taken by a different person (analogous to a scripting engine), but regardless of who takes your order, your order will be packed and shipped by the same people in the warehouse (analogous to WSH).



All WSH scripts are stored as plain text files, in a file whose extension depends on the specific scripting language in use. When you execute a script, here's what happens, in a somewhat simplified form:

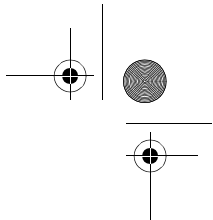
1. Based on the extension of the script file, WSH searches the Windows registry for information on the scripting engine that is associated with the script language.
2. Using the information obtained in (1), an instance of the scripting engine is created. At the same time, an instance of the script object (the host) is also created.
3. WSH establishes two-way communication between itself and the scripting engine.
4. The script file is read and its commands carried out.

## Scripting Technology

It would be an error to think of WSH as some sort of independent "extra" that has been added to the Windows operating system. There is much integration within the operating system, and nothing is truly independent. When you create and use scripts, there are really several technologies coming into play:

- Windows Script Host provides the full-system scripting environment and object model that provides access to system and application functionality.
- The VBScript and JScript engines permit script authors to take advantage of the power of these two modern and full-featured scripting languages.
- COM (Component Object Model) is a standard for creation of reusable software components that expose an automation interface, and therefore can be controlled by scripts. COM is central to many aspects of Windows and Windows applications.
- Windows Script Components (previously called Server Scriptlets) supply the capability to provide COM objects in script code, which in turn permits the creation of reusable script objects.

You can see from these descriptions that the term "Windows Script Host," technically speaking, refers to only part of the picture. From the perspective of the script programmer, however, these distinctions are largely irrelevant, just like the details of a car's engine and transmission are irrelevant to most drivers. For the remainder of this book I use the term "Windows Script Host," or WSH, in the broad sense of referring to the full range of technologies that are utilized by scripts.



## Two Scripting Utilities

The WSH actually offers two scripting utilities. Under the surface they are essentially identical. They differ in the interface between the utility and the user. Cscript is the command line WSH scripting utility, which communicates with the user by means of the DOS command line. In Windows, of course, this is a "DOS box," which is identified on the Start menu as "MS-DOS Prompt." Wscript is the Windows interface version of the script utility, which communicates by means of screen windows.

Let's take a look at these two approaches to scripting, using a very simple script (Listing 1-1). Use any text editor, such as Notepad, to create a file containing the code shown in the listing. Save the file using the name List0101.vbs. You can use another name if you prefer, as long as you be sure to use the .VBS extension.

### LISTING 1-1

*A simple script to display a screen message.*

```
' List0101.vbs  
' A very simple script!  
  
Wscript.echo "Welcome to the Windows Scripting Host!"
```

Let's take a quick look at the code in this very simple script. The first two lines are *comments*, which are not executed by WSH but serve only as a way for the programmer to include comments and notes in a script file. The third line is the only line of executable script code in this file. It uses the `Wscript.echo` command to display text on the screen.

### Create a scripts folder.

It is easier to keep your scripts organized if you keep them all in one place. For example, you might create a folder named `C:\Windows\Scripts` to keep your scripts.

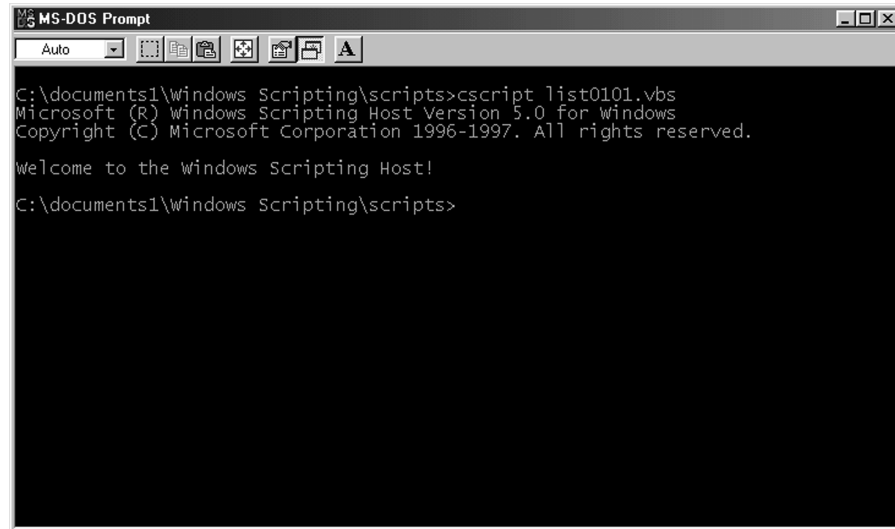
Once the script file has been created, here's how to run it using the Cscript command line utility:

1. Open an MS-DOS window by selecting MS-DOS Prompt from the Windows Start menu.
2. At the prompt, enter the command to change to the folder where your script file is stored. For example, if your script file is in the folder `C:\Windows\Scripts`, then the command will be

```
cd c:\windows\scripts
```

3. Enter the following command to start the Cscript utility and execute your script:

```
Cscript list0101.vbs
```



```
MS-DOS Prompt
Auto
C:\documents1\Windows Scripting\scripts>cscript list0101.vbs
Microsoft (R) Windows Scripting Host Version 5.0 for Windows
Copyright (C) Microsoft Corporation 1996-1997. All rights reserved.

Welcome to the Windows Scripting Host!
C:\documents1\Windows Scripting\scripts>
```

**FIGURE 1-1**

*The output when the sample script is run with Cscript.*

4. Examine the output, which is shown in Figure 1-1.

When you examine the output of the script, you'll note several things (your output may look slightly different depending on the folder where you put the script file).

1. The first line is where you entered the command to run the script.
2. The second and third lines are the version and copyright notices displayed by the Cscript utility.
3. The fourth line is the output created by the script.
4. The fifth and last line is the command prompt, which returns after the script terminates.

Now let's try out the Wscript utility. You could run it using the same steps as just described, replacing the Cscript command in step 3 with Wscript. However, it does not make a lot of sense to use the MS-DOS command line to run a Windows interface program. Here's a better way to run a script using Wscript:

1. Open the Windows Explorer.
2. Navigate to the folder where the script file List0101.vbs is located.
3. Double-click the name of the script file.

### Script file actions

How does Windows know to run a script file with Wscript when you double-click the file name in Explorer? Windows keeps a list of file extensions and the actions that are associated with them. When WSH is installed, the action associated with opening a file with a .vbs or .js extension is set to "execute with Wscript." A file is opened in Explorer by double-clicking it or by selecting Open from its context (right-click) menu. Other actions are available on a file's context menu, such as Edit, which, for script files, opens the file in Notepad for editing. You can view and edit the actions for various file extensions as follows:

1. Open My Computer.
2. Select Folder Options from the View menu.
3. In the dialog box, select the File Types tab.

The output of the script as created by Wscript is shown in Figure 1-2. You can see that this is a lot more appropriate for the Windows environment, consisting of a dialog box with the message and an OK button.

Now let's take a look at another script, the one in Listing 1-2. This script does exactly the same thing as the one in Listing 1-1, but it is written in a different scripting language — JScript rather than VBScript. The two scripts look quite similar, but you'll note some differences:

1. The JScript comment is enclosed between `/*` and `*/` rather than being preceded by an apostrophe.
2. The text to be displayed by the `WScript.Echo` command is enclosed in parentheses as well as double quotes.
3. Statements in JScript end with a semicolon.
4. JScript scripts are saved in files with the `.JS` extension.
5. It is not obvious from the listing, but JScript is case sensitive. Thus, `WScript.Echo` must be entered exactly as shown; if you try `wscript.echo` or some other capitalization variant, it will not work.



**FIGURE 1-2** The script file output produced by Wscript.



**LISTING 1-2***Another version of the simple script.*

```
/* List0102.js  
A very simple script! */  
  
WScript.Echo("Welcome to the Windows Scripting Host!");
```

If you want to try this JScript example, follow the instructions for the VBScript example that were given earlier. Since the file will have the .JS extension, the scripting engine will know which language it contains and process it appropriately. The output will be exactly the same as it was for Listing 1-1 whether you use Cscript or Wscript.

**Two JScript comment styles**

JScript supports two styles of comments. As mentioned in the text, anything between `/*` and `*/` is a comment. This style can span multiple lines. Also, a single line that begins with `//` is a comment. To use this style with multiple lines, each line must begin with `//`.

## WSH Languages

As you have seen, WSH as provided by Microsoft has support for two different scripting languages: VBScript and JScript. VBScript is based on Microsoft's Visual Basic and Visual Basic for Applications languages, which are used in the Visual Basic programming environment and the Microsoft Office applications suite, respectively. VBScript is not identical to Visual Basic but is very similar, with most syntax and language elements in common. JScript is Microsoft's implementation of the JavaScript language. Both JScript and JavaScript are, as the names imply, based on the Java language, although there are significant differences.

Do you need to know Visual Basic or Java to use WSH or to get the most out of this book? Definitely not. If you do happen to know either of these languages then you will certainly pick up VBScript and JScript more quickly, but that's all. You'll find all of the information you need to master both of these languages in this book.

Which language should you use? From a practical standpoint, it does not really matter as neither of the languages is really better than the other. I believe that VBScript is a little easier to learn and use, but that is a personal opinion, and other experienced WSH programmers may feel differently. If you are just going to write scripts for yourself and some friends, you can probably get away with learning just one of the languages. If, however, you will be doing WSH development at the professional level, I strongly suggest



that you develop at least a reasonable familiarity with both languages. Even though you may write all your own scripts in your preferred language, you never know when you may be called on to modify or debug a script that someone else has written in the other language.

## WSH and Objects

WSH development is closely based on the concept of *objects*. An object is a self-contained tool that has been designed to perform a specific task. The Windows operating system itself makes use of objects in most of the things it does, as do most Windows applications programs such as Excel and PhotoShop. Most of the scripts you write will use objects to carry out their tasks. The objects that are available to a script fall into three general categories:

- Objects provided by WSH itself
- Objects that are part of the Windows operating system
- Objects that are part of applications programs, such as Microsoft Office (ActiveX objects)

The beauty of working with objects is that the code required to perform most of the actions you need to do has already been written, tested, and tucked inside an object ready for you to use it when needed. For example, one of the objects that is part of WSH is the `FileSystemObject`. This object already "knows" how to perform all the file management tasks that a script might need, such as copying and renaming files and folders. When your script needs to perform a file management task, all you need do is write the VBScript or JScript code to send the proper commands to the `FileSystemObject`. You'll learn a lot more about working with objects in Chapter 2, and the details of the various objects available to a script will be presented as needed throughout the book.

## Overview

The Windows Scripting Host is Microsoft's answer to the command scripting needs of the millions of users of the Windows operating system. Compared with other operating systems, such as Unix, scripting is a relatively new addition to Windows. It is, however, a very powerful addition, providing multiple language support, a powerful object model, access to existing ActiveX objects, and a powerful and flexible syntax. WSH puts you in command of your operating system and can simplify many tasks and save you time as well.

