1 Introduction into WSH

This chapter contains a brief introduction into the Windows Script Host (WSH). You will get a first impression of what you can do with this tool, how to install it, and how to use it.

What is the Windows Script Host?

Windows versions before Windows 98 provided almost nothing to automate certain tasks like backing up files, displaying user dialogs, and maintaining system administration tasks. Many system administrators and power users requested a tool to support these tasks.

> **NOTE:** Of course, you could always use the old MS-DOS batch (BAT-files) file to perform certain tasks like copying files. These BAT-files can be used within a Windows Command Prompt window (the window with the MS-DOS command prompt). However, this approach has several disadvantages like the fact that a BAT-file can only contain a simple sequence of MS-DOS commands (only simple branches and no real looping functionality among other things). Also, dialogs and messages boxes are not supported. In Windows 3.1, you could do a little more with macro recorder to record and play simple keystrokes and mouse clicks; however, the recorder is not available under Windows 9X and Windows NT and does not allow programming.

All this led users to seek out third party solutions (like Power Batch) or programming environments like Delphi, Visual Basic, and Visual C to handle these tasks. But these solutions were not acceptable for many Windows users that required a simple and free scripting language to solve their automation tasks.

Since Microsoft Office 97 provides Visual Basic for Applications (VBA) and since web authors also know scripting languages like VBScript and JavaScript, it was only a matter of time before Microsoft provided a scripting language for Windows operating systems. This tool is called the Windows Script Host (WSH), and falls under the umbrella of Microsoft's scripting technologies.

- **Windows Script Host (WSH):** This is a standalone host which enables you to execute a script file directly at the operating system level. You can use a command-line interface or you can double click a script file in the Explorer window, for example. WSH supports both languages – JScript and VBScript – because it uses the language engines provided by Internet Explorer 4.0/5.0. WSH comes in handy for many administrative tasks that require little or no user interface. It is far more powerful than old MS-DOS batch files, since JScript/VBScript provide a powerful scripting language with full access to WSH objects and to any other available automation objects.

- **Internet Explorer HTML scripts:** HTML pages may contain also JScript and VBScript scripts to automate some tasks. ActiveX controls included on a page extend the features of the scripting languages. With the objects exposed from WSH, it is also possible to access the Registry, the file system, and other objects within a HTML script. Because using ActiveX components in the Internet environment can be unsafe, many users deactivate ActiveX controls execution for the Internet zone within Internet Explorer. Since some scripts are only for personal local use, safety settings for ActiveX controls are unnecessary. So Microsoft intro-

duced **HTML Applications (HTA)**, HTML documents which contain scripts. With some extras you can do everything you can with HTML documents, and you can use any ActiveX control within your system (without being prompted by Internet Explorer to load the control). This allows you to create your own HTML applications which are similar to normal Windows programs.

Although web authors are the people that have primarily used scripting, it can also be used by power users and system administrators to tailor their personal Windows system to suit their needs. The current book deals primarily with the Windows Script Host, although you can use the same programming techniques within HTML scripts.

> **NOTE:** Microsoft recently has changed the official name from »Windows Scripting Host« to »Windows Script Host«. Within this book I will use both names and the abbreviation WSH.
>
> Besides JScript and VBScript, there are many other scripting languages as TCL, Perl, Python, and Rexx. I decided to focus the content of this book on JScript/VBScript, because these languages are supported directly from WSH.

What can we do with WSH scripts?

There are many possibilities for tailoring your personal Windows system using scripts. You can use power scripts to solve special tasks that can be processed with a double click from shortcuts added to Windows desktop or into the start menu, use schedulers to invoke script execution, or add the script to your startup files (so the script executes during each system start). Here a few examples of the possible tasks which can be automated using scripts:

- Back up or restore some files within your system (this comes in handy if you need to save only a few files from your machine to a network server).
- Shut down or restart Windows with a simple mouse click. A script allows you to add special shutdown tasks like to back up certain files after closing applications, etc.. Log a user's name after booting the system and more. (Windows NT provides many log features, but you can also record some events under Windows 9x using a script.)
- Integrate applications and their data even if they were not necessarily designed to interact. Office applications may be launched from a script. Then the script can load and process a document, print it and close the application.
- Write small but powerful applications (scripts) that can organize your access to common tasks and data.
- Manage system administrator tasks like adding, updating or removing user accounts under Windows NT. A WSH script can automate this whole task using the Active Directory Services Interface (ADSI) provided in Windows NT.
- Through suitable objects, directly access the Windows shell. This allows you to create shortcuts or map network devices (drives, printers).
- Read Environment variables or you can retrieve information about Windows. In addition, programs can be started and automation objects may be used.
- Show dialogs to inform the user about the program status or open dialog boxes to retrieve user input.

- Access the Windows shell and Windows API to control windows and other devices on your system. This can be done with a few tricks described within this book.

There are many other tasks which can be solved with WSH scripts. Reading this book will open your eyes to such solutions.

A few remarks about JScript/VBScript

The Windows Script Host which shipped with Windows 98 and Windows 2000 (and the downloadable version for Windows 95/NT 4.0) comes with two programming languages:

- **VBScript:** This language uses the same syntax as Visual Basic; it is actually a subset of Visual Basic (special commands like explicit data type declaration and Declare-statements are omitted). Or more precisely stated, Visual Basic (VB) is a superset of Visual Basic for Applications (VBA), which is a superset of VBScript. People who have programmed with Visual Basic or VBA, learn VBScript very quickly. With some experience, you can port VB or VBA-programs directly into VBScript (you need only remove the unsupported language constructs).

- **JScript:** This is Microsoft's implementation of ECMAScript. ECMAScript is the vendor-independent standard for a programming language based on the original work of Netscape's JavaScript. People who have worked with JavaScript, can transfer seamlessly to JScript.

These two programming languages are sufficient to enter the world of script programming. However, Microsoft designed an open interface for WSH; thus, third party vendors can integrate their own language engines to support other languages like Perl, TCL and Rexx.

What is the difference between VBScript and VB/VBA or between JScript and JavaScript?

If you have programmed already in VBA and/or VB or prepared scripts in VBScript and/or JScript for HTML-documents, then writing WSH scripts in VBScript and/or JScript should no be a problem. All you need to know are a few of the differences which exist between WSH-scripts and the areas mentioned above:

- Visual Basic provides a powerful development environment to create applications. The programs can be compiled into EXE-files. This functionality is not available under the Windows Script Host. All scripts must be kept in simple text files with file name extensions *.vbs* or *.js*. WSH uses Internet Explorer language engines to interpret the content of a script file directly. The advantage to this is that you can prepare your scripts using a simple text editor (like Windows Notepad).

- All language constructs supported in VB and VBA to declare external functions and procedures are not available. Also routines for extended run-time error handling are missing. In addition VBScript/JScript treat all variables with the data type *Variant*.

- Some internal objects and functions supported in the HTML/JavaScript Document Object Model cannot be used in WSH-scripts. For instance the *document.window* object isn't supported for user output.

Also, WSH does not support event handling (like *onClick*) like HTML scripts, because the WSH environment does not provide an extended user interface (only a few message boxes). I will discuss later the possibility to connect an event sink from an external application to WSH script. Further details about VBScript/JScript may be found in chapter 3 and 4.

How to install WSH?

The Windows Script Host was first shipped with Internet Information Server 3 for Windows NT. The Windows Script Host is also part of Microsoft Windows 98, and it will be part of Microsoft Windows 2000. For Windows 95 and Windows NT 4.0, Microsoft provides a download section on the Internet for the necessary files. Before you can use the Windows Script Host, you must install this feature.

How to check whether WSH is installed?

A check whether the Windows Script Host is installed may be done with a few mouse clicks. You need files using the file name extensions *.js* and . *vbs*.

Figure 1.1.
Icons and file name extensions for script files

Just browse a folder that contains a .js and a .vbs file (for example in Windows Explorer). If the icons shown for the files are equal to **Figure 1.1**, the Windows Script Host is (probably) installed already. If the files are shown with the icon for unregistered file types, proceed with the following steps to add the WSH feature to your operating system.

Install WSH in Windows 98

If the Windows Script Host is not already installed on your Windows 98 machine, perform the following steps to add this feature:

Figure 1.2.
Icons and file name extensions for script files

1. Open the Control Panel folder and look for the *Add/Remove Programs* icon.
2. Double click the icon and select the *Windows Setup* property page (**Figure 1.3**, left).
3. Check the *Accessories* check box and click *Details*.
4. Check the *Windows Script Host* check box and click *OK* (**Figure 1.3**, right).
5. Click *OK* (or *Apply*) to apply the new settings.

Windows 98 asks for the Win 98-Install-CD-ROM and then adds the WSH feature. Afterwards, you should see the icons for registered file types as shown in **Figure 1.1**.

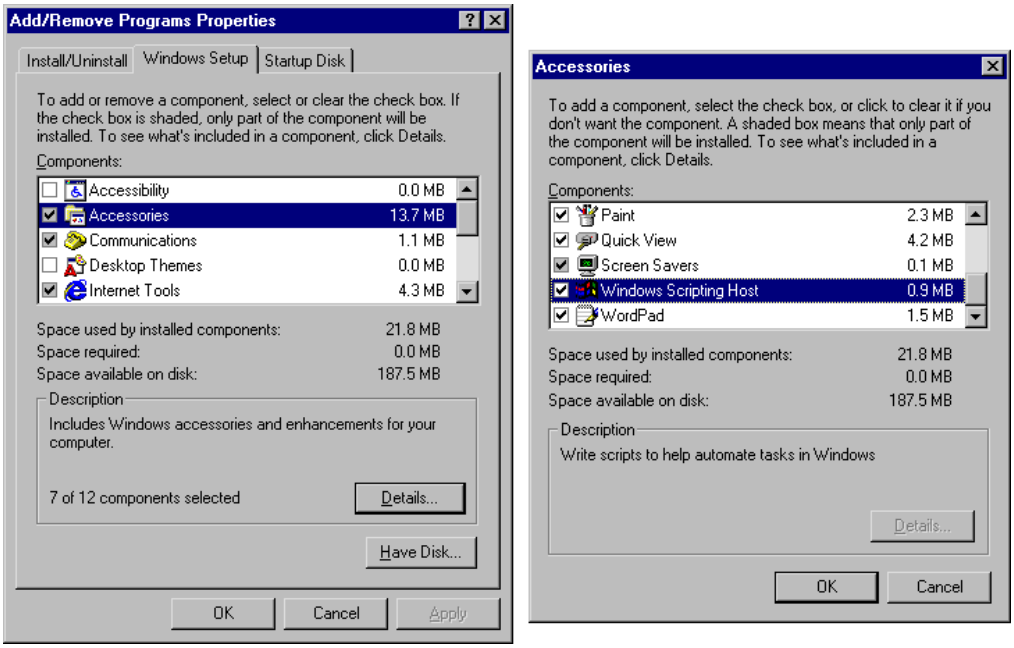

Figure 1.3.
Property pages to add Windows features

Windows 95/Windows NT 4.0

Windows 95 and Windows NT 4.0 are not shipped with the Windows Script Host. But Microsoft offers the Windows Script Host for free. You must download the required files from Microsoft's website.

Figure 1.4.
Internet Address of the Microsoft Scripting Website

1. Use the URL address shown in **Figure 1.4** to access Microsoft's web site. If you are a new visitor, Microsoft will ask you to register online first.

2. After you get access to the Microsoft Script Technologies website, select the items *Windows Script Host* and then *Downloads* in the upper left pane (see **Figure 1.5**).

3. Click onto the *Download Microsoft Windows Script Host* hyperlink, and follow the advice given in the browser window.

Figure 1.5.
Microsoft's Website for Windows Script Technologies

> **NOTE:** The Microsoft Scripting website at *http://msdn.microsoft.com/scripting* may be changed in the future. During my first visits I was able to download a simple EXE-file containing the Windows Script Host for Windows 95 and Windows NT. Within my last visits the *Download Microsoft Windows Script Host* hyperlink guided me to the Windows 98 update page.
>
> **IMPORTANT:** Using the Windows Script Host under Windows 95 and Windows NT requires at least that you have installed Microsoft Internet Explorer 4.0. WSH uses several DLL's and the language engines for JScript/VBScript.

Some remarks about versions

The Windows Script Host was shipped as WSH 1.0 during the time I wrote this book. At the time I finalize this pages, there is a rumor that WSH 2.0 is around the corner (which is scheduled for Microsoft Windows 2000). I have read some documents which indicate to me that we will have significant changes within WSH 2.0, but this topic isn't covered within this book yet.

Besides the WSH version, you must know the version of the language engines currently used. Microsoft Internet Explorer provides these engines. Originally WSH was based on Microsoft Internet Explorer 4.0, which supports version 3.1 of JScript/VBScript. Since this time, Microsoft has released several upgrades to these languages engines. Version 3.1a was released to fix a few bugs (but this version still contains other bugs, which I will discuss within the following chapters). Visual Studio 6.0 was shipped with version 4.0 of the language engines. After installing Microsoft Internet Explorer 5.0, version 5.0 of the languages engines is available. Microsoft provides also version 5.0 of the JScript/VBScript language engines for download from their website (*http://msdn.microsoft.com/scripting*). These versions of the script engines can be used with IE 3.0, IE4.0, IIS 3.0, and IIS 4.0 and provides fixes to bugs found in the Version 3.0 engines. These engines extend also the language features. Although the first version of this book has been written originally using version 3.x language engines (and the samples were later tested under version 5.0), I recommend that you install the version 5.0 engines. The bug fixes and the extensions simplify the live of a script programmer.

How to create and use scripts

Is WSH installed on your machine? Do you have some script files that you would like to execute? Would you like to write your first script? This explains in principle how to create a simple script file and execute it.

Let's create our first (VBScript) script

Are you prepared to create and use your first script? I guess it's sufficient to create a simple program that displays just a dialog box with the message *Hello world* (**Figure 1.6**). We can use this script to demonstrate the basics of script programming, and how to launch a script.

Figure 1.6.
A simple dialog box

As you can see in the header of the dialog box, the script language will be VBScript. To create a simple dialog box according to **Figure 1.6** we need only one line in VBScript:

```
MsgBox "Hello world"
```

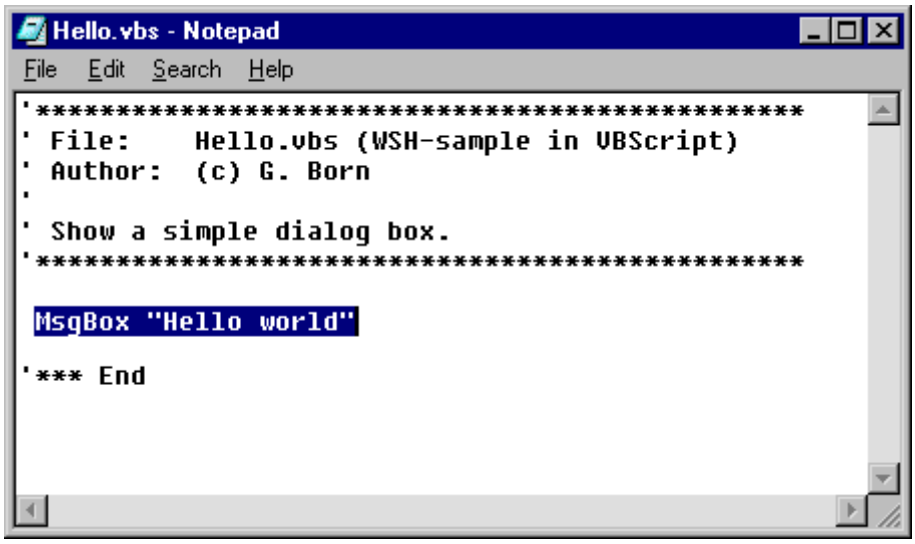

Figure 1.7.
Our first VBScript-program

But how do we create the script file? Not a problem, for your first script, you only need *Notepad.exe* (the Windows editor). OK, let's go:

1. Fire up your Windows editor (Notepad.exe) and enter the statements shown in **Figure 1.7**. The highlighted statement is the most important part of the code, because it creates the dialog box.

2. Save the contents of your Editor-window into a text file on your hard disk. You are free to choose any file name (like *Hello*), but the extension must be set to *.vbs*.

   NOTE: It is important to set the file name extension *.vbs* for VBScript programs (use *.js* for JScript) in the *save as* dialog. These file types are registered for WSH. If your script files uses a different file name extensions, WSH 1.0 will not recognize the script language. Double-clicking such a file with a »wrong« extension (*.vb* for example), opens the Windows *Open with*-dialog (because no application is registered for this file type). If you use the extension *.txt* instead, Windows invokes your editor to show the file's contents. If you try to force script execution using a command line in your Windows Command Prompt window, the program *cscript.exe* reports an error, because the file extension isn't registered.

After storing the script, there should be a file *Hello.vbs* on your hard disk with the source code shown in the following listing.

```
'*********************************************
' File:    Hello.vbs (WSH-sample in VBScript)
' Author:  (c) G. Born
'
' Show a simple dialog box.
'*********************************************

 MsgBox "Hello world"
'*** End
```

Listing 1.1.
The first VBScript-program

Maybe you are wondering why I have used 9 lines although I mentioned that we need just one command to invoke the dialog box. OK, you can create a simple script file with just one line:

```
MsgBox "Hello world"
```

However, I would like to encourage you to add some additional information as comments into the script file. The script engine ignores these comments, but it would be helpful, if your script file contains a few comments about the author, the purpose of the file and so on. If somebody gets this file, he/she will be able to read these comments. And, comments come in handy also for the script author, or do you always remember the purpose and contents of a script file after a few weeks? I don't! Therefore, I always add a comment header to my script files with further information about it's contents.

Now let's go back to the sample shown in the listing above. As agreed, we have chosen VBScript as the programming language. Thus, all statements must correspond to the syntax of this language. In VBScript comments are marked with a leading quote. So the following line:

```
' This is a comment
```

defines a valid comment. The whole line after the quote ' will be ignored by the VBScript language engine used within the Windows Script Host. You may comment whole lines or only the end of a statement. The statement:

```
MsgBox "Hello world"  ' Show a message
```

shows a dialog box with the text *Hello world*, but the comment at the end of the line will be ignored.

> **TIP:** You will find the ready-to-use sample file *Hello.vbs* in the folder *\Samples\Chapter01* of the samples files. These samples are distributed in a ZIP archive. After unpacking the ZIP file, the samples are located in several folders. For chapter 1 you will find the sample files in the folder *\Samples\chapter01*. To simplify creating scripts, you will find the template files *VBScript.vbs* and *JScript.js* in the folder *\Samples\Chapter01* of the sample files. You may use these files as templates to create a new script file. Load the template into your editor, add the new commands and save the contents into a new file. To load the template, right click onto the file's icon and select *Edit* in the shortcut menu.

Same sample, but now in JScript

Do you prefer JScript for your script programs? No problem, you can use the same steps as mentioned on the previous pages. You must enter however all statements in JScript syntax. The JScript example looks like the following listing:

```
//*********************************************
// File:    Hello.js (WSH-sample in JScript)
// Author:  (c) G. Born
//
// Show a simple dialog box.
//*********************************************

 WScript.Echo("Hello world");

//*** End
```

Listing 1.2.
Sample in JScript

> **NOTE:** Keep in mind to use the file extension *.js* within the *Save as* dialog. Otherwise WSH can't detect the script.

Because JScript uses a different syntax than VBScript, I would like to discuss a few of the differences. First of all, comments are marked in JScript with two leading slashes //. The sample shown above contains a comment header and trailer. The other difference comes from within the statement to open the dialog box. JScript doesn't supports a *MsgBox*-function. So I have used the *Echo*-method provided from W*Script* object (I will discuss later what are objects, methods and so on). The statement:

```
WScript.Echo("Hello world");
```

creates a simple dialog box containing the text *Hello world* and the *OK* button. Did you recognize the second difference? Another difference with VBScript is that JScript terminates all statements (excluding a few exceptions) with semicolons.

> **NOTE:** The sample *Hello.js* may be found in the folder *\Samples\Chapter01* of the samples.

How to execute WSH scripts?

Do you have WSH scripts stored in *.vbs* or *.js* files? You may execute the scripts either in Windows or on the Windows Command Prompt.

> **NOTE:** Behind the scenes, there are two files used to implement the Windows Script Host. W*script.exe* provides a Windows-based host for scripts while *cscript.exe* allows script execution from the Windows Command Prompt .The file *wscript.exe* is located in the Windows-folder for Windows 98, while *cscript.exe* may be found in the subfolder *\Command*.

Launch a script in Windows

Executing a script in Windows is simple: Just double-click on the script file's icon. Windows executes the applicable script (using the *Wscript.exe* host by default). This happens because during the Windows Script Host installation, the file types get registered for WSH.

For the first test, you can browse for the file in *\Samples\Chapter01* of the samples. Then double-click onto the script program to start the sample. The result of the script *Hello.vbs* is shown in **Figure 1.6**, while **Figure 1.8** shows the result of *Hello.js*. Both samples create a simple dialog box, which may be closed by clicking the *OK* button.

Figure 1.8.
Dialog created from Hello.js

> **NOTE:** Have a look at the title bar of the dialogs. In **Figure 1.8** we get the title *Windows Scripting Host*, because the JScript-program uses the *Echo*-method provided by WSH. The VBScript-program uses the internal *MsgBox*-function. This is the reason why we get the text *Visual Basic* in the title bar. There is also a possibility to define your own title text. I will come back to this issue within the next chapters.

Using the Windows Command Prompt to execute scripts

It is also possible to use the Windows Command Prompt window to launch a script. Executing scripts in the Windows Command Prompt window is supported by the program *cscript.exe*.

> **NOTE:** Cscript.exe is a Windows console application (runs in a Command window and sends output to STDOUT like native DOS/NT commands). It is intended primarily for non-interactive tasks.

To execute a script, you must enter the following command into the Windows Command Prompt window:

```
cscript path\script name [Host options] [Script options]
```

Script name is the placeholder for the script name including the drive and the path to the script file. Additional options for the Host and/or for the script may be appended in the command line (***Table 1.1***).

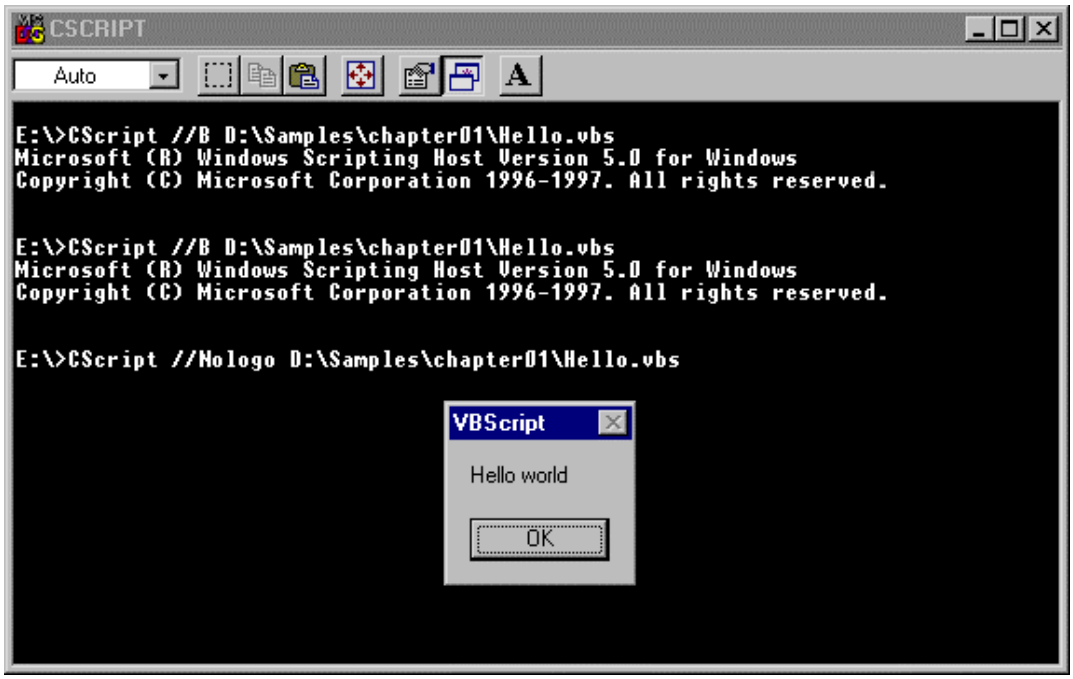

Figure 1.9.
MS-DOS commands to call the Windows Script Host

The Host options enable or disable Windows Script Host functions. Two slash characters always precede these Host options //. The following table contains a survey of all Host options:

Option	Remark
//I	Interactive Mode: allows display of user prompts and script errors (this is the default, and the opposite of //B).
//B	Batch Mode: suppresses command-line display of user prompts and script errors. If a script uses the *Echo* method all user prompts will be disabled.
//T:nn	Set a Time-out value – the maximum time in seconds a script may run.
//logo	Displays a banner (the text shown in Windows Command Prompt window after execution of a script, see upper lines in **Figure 1.9**). This is the default option, //nologo is the opposite).

//nologo	Prevents the display of an execution banner.
//H:Cscript //H:Wscript	Registering *Cscript.exe* or *Wscript.exe* as the default application for running scripts. If this options isn't specified, *Wscript.Exe* is assumed as the default scripting host.
//S	Saves the current command-line options for this user.
//?	Displays a help page with the host options.

Table 1.1.
cscript.exe *host options*

Figure 1.9 shows a Windows Command Prompt window with a sample command line calling the script *Hello.vbs* with several options.

Set the script properties

According to my explanations given above, you may set several options to execute a script under the Windows Command Prompt. In Windows you may also use the *Run* dialog to pass parameters to a script.

In most cases a user prefers to start a script with a double-click (either on the script file itself or on the shortcut file if parameters must be submitted). However, you have the possibility, to define additional script properties under Windows.

1. Right-click onto the script file. Select *properties* in the shortcut menu.

2. Select the requested properties on the *Script* property page (**Figure 1.10**).

The check box *Stop script after specified number of seconds* and the associated up-down control *seconds* allow you to set a Time-out value. If a script is still executing when the time-out is reached, Windows will terminate the script.

As soon as you close the property page using the *OK* button, Windows creates a new file with the name of the script file and a *.wsh* extension. Double-clicking this file forces Windows to execute the script with the properties set on the property page. *.wsh* files are simple text files similar to *.ini* files. You can open such a *.wsh* file with an arbitrary text editor (like Notepad). Below the contents of such a *.wsh* file is shown:

```
 [ScriptFile]
Path=C:\WINDOWS\Samples\WSH\showprop.vbs

[Options]
Timeout=10
DisplayLogo=1
BatchMode=0
```

Listing 1.3.
A sample WSH file

The (absolute) Path statement in the *[ScriptFile]* section identifies the script file to be executed. The keywords in the *[options]* section determine the run-time properties. Timeout is set to the time

the user has specified on the Script property page. *DisplayLogo=1* forces the display of the logo in the Windows Command Prompt window (**Table 1.1**).

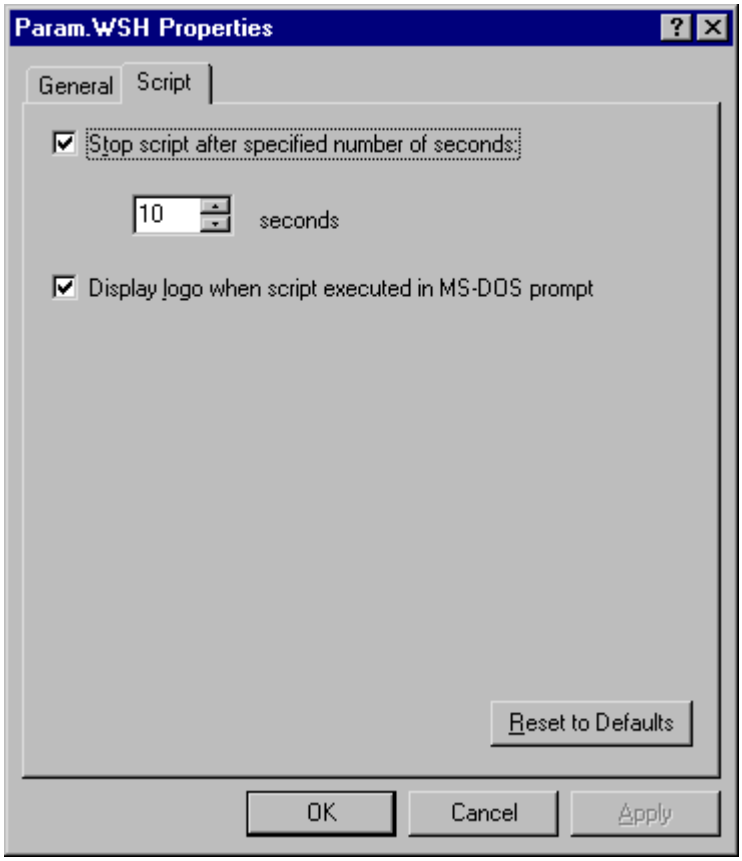

Figure 1.10.
The Script property page

> **NOTE:** The file *Param.wsh* shipped with the samples probably causes an error on your system because of the wrong path settings. Copy the script files *Param.vbs* and *Param.js* to a local folder on your hard disk. Then, create a new WSH-file (see the section above **Listing 1.3**).
>
> **TIP:** The entry *BatchMode* can not be set within the *Script* property page and defaults to 0. This enables user prompts and error messages during script execution. If you set this value to 1 (using an editor), the script will be subsequently executed in batch mode (all messages are suppressed). This comes in handy, if you want your script to run in the background (perhaps during a system start).

How to submit parameters to a script?

The samples mentioned above do not require any parameters. There are however surely cases, in which the user submits the name of a file to the script. The following VBScript demonstrates how to display all parameters submitted to the script (**Figure 1.11**).

Figure 1.11.
Displaying submitted parameters

This script requires a few additional statements. For those who know VBScript, I like to show the code to examine the submitted parameters in the listing below.

```
'*********************************************
' File:     Param.vbs (WSH-sample in VBScript)
' Author:  (c) G. Born
'
' Show all parameters submitted to the script in a dialog.
'*********************************************

 text = "Arguments" + vbCRLF +vbCRLF

 Set objArgs = Wscript.Arguments    ' create object
 For I = 0 to objArgs.Count - 1     ' loop for all arguments
  text = text + objArgs(I) + vbCRLF ' get argument
 Next

  Wscript.Echo text ' show arguments using Echo
'*** End
```

Listing 1.4.
Program to show all submitted parameters (arguments)

The script obtains the *Arguments* collection of the W*Script* object and shows all parameters contained in this collection (I will discuss the details later on).

Let's assume you have an existing script file that you would like to submit some parameters to. At this point, we only need to know how to submit these parameters. A double-click on the script file does not meet our requirements, because it just executes the script.

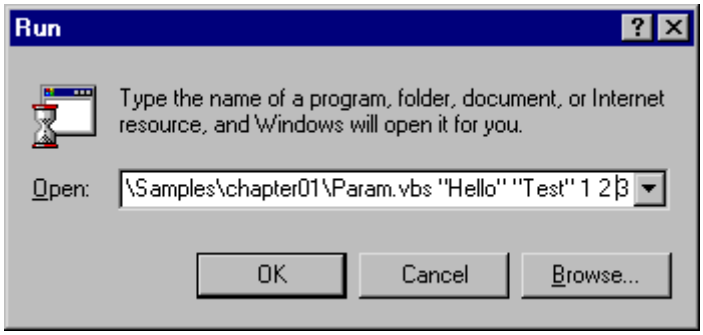

Figure 1.12.
Submitting parameters to a script using the Run-*Dialog*

To submit parameters to a script, you may use the *Run* dialog shown in **Figure 1.12** (use the *Run* command in the start menu). You must enter the drive, the path and the name of the script file in the *Open* text box. The parameters are appended to the command line, and they must be separated with blanks. The script mentioned above reads the submitted parameters and displays the results within a dialog box (see **Figure 1.11**).

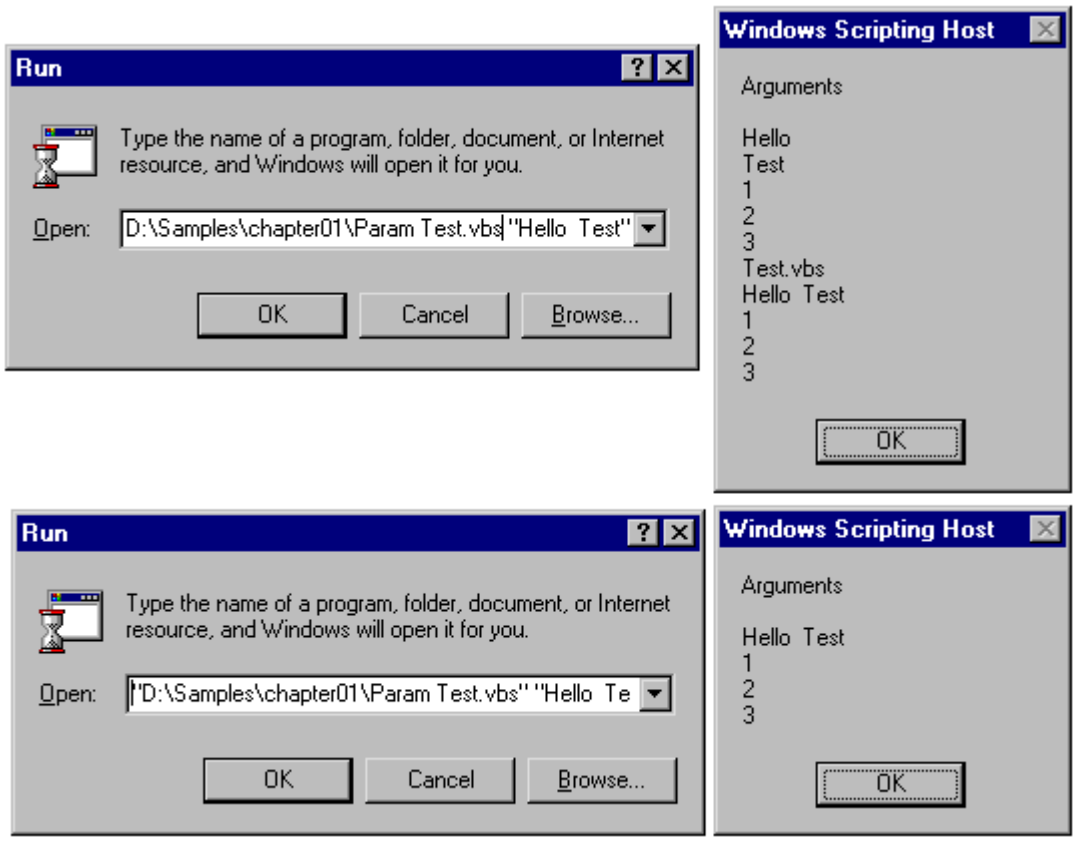

Figure 1.13.
Examples with wrong parameter selection

Some remarks about possible problems

At this point I would like to discuss some problems with submitting parameters. In **Figure 1.12** I have enclosed some parameters in double-quotes. This is necessary, if you submit text which contains blanks as a parameter. Let's assume you would like to submit the name »Hello world« as one parameter to your script. If you use the command:

I:\Samples\Chapter01\Param.vbs Hello world

the script shows two parameters, because WSH interprets both sub strings »Hello« and »world« as separate parameters. To submit a string which contains blanks as one parameter, you must enclose the string in double-quotes. The command given above must be written as shown below:

I:\Samples\Chapter01\Param.vbs "Hello world"

Unfortunately I must mention another problem which may occur during submitting parameters to a script. If you use long filenames for your script files, then you need to be aware about how Windows processes command lines. **Figure 1.13** shows two cases using different commands. The left part of the figure shows the content of the *Run* dialog with the executable command. The right part of the figure shows the results recognized within the script. While the dialog box in the lower right corner shows the submitted arguments in the required order, the dialog box in the upper right corner displays some curious results – the parameters are shown twice. This was because a long file name containing a blank is used in the command line. In the first example (see the *Run* dialog in the upper left corner in **Figure 1.13**) only the parameter was set in double-quotes. So WSH gets »confused« and the script shows a wrong number of arguments. The solution is simple: If the name of your script file consists of a long filename, you must enclose this name also in double-quotes. This is used in the *Run* dialog shown in the lower left corner in **Figure 1.13**. As you can see the corresponding dialog box shows the parameters in the right order.

You may test this behavior. The folder *\Samples\Chapter01* of the samples contains the script files *Param.vbs* and *Param.js* (which show the submitted parameters). The files *Param Test.vbs* and *Param Test.js* contain identical statements (I have renamed the original files to be long filenames). If you execute the scripts using the *Run* dialog, you might get the results shown in **Figure 1.13**.

Probably you feel uncomfortable submitting parameters using the *Run* dialog. If the parameters don't change, you can use a trick and create a shortcut to your script file.

1. Right-click onto the script's file icon.
2. Select *Create Shortcut* in shortcut menu.
3. Right-click onto the shortcut file and select *Properties* in the shortcut menu.
4. Set the command and the required parameters in the text box *Target* in the *Shortcut* property page (see **Figure 1.14**).

After closing the property sheet using the *OK* button, Windows saves the parameters. These parameters will be submitted automatically to the script if the user double-clicks the shortcut file.

> **NOTE:** According to my explanations in the preceding sections, you must enclose parameters and file names (which contain blanks) in quotation marks.

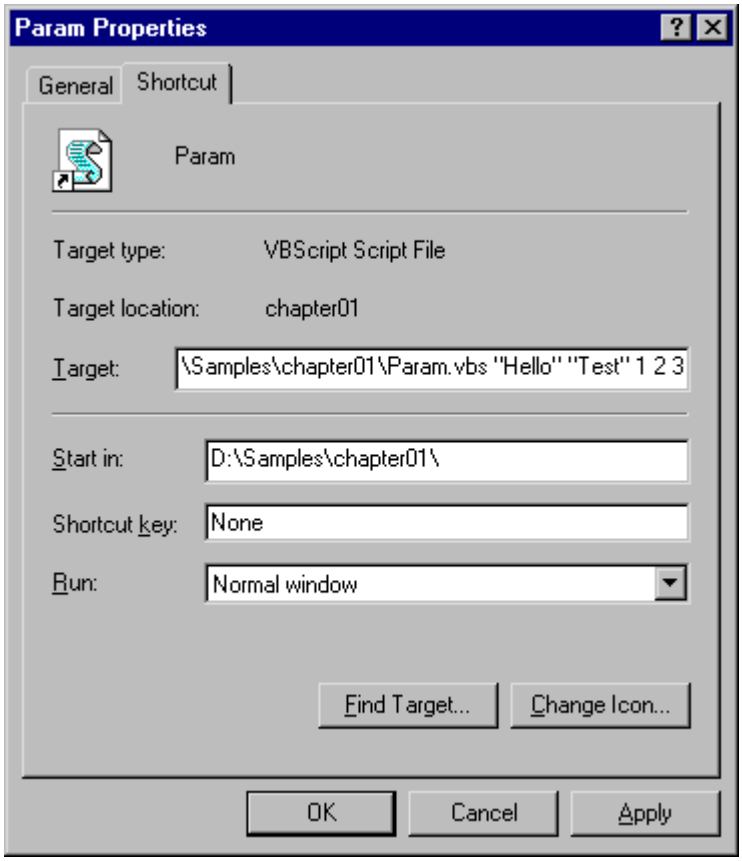

Figure 1.14.
Shortcut properties for a script file

Passing arguments in *Cscript.exe*

Does your script request some parameters (for instance a file name)? Besides the Host options, which are marked with two preceding slashes, you can submit certain parameters as additional script options to the script. A simple slash / can be placed in front of the appended script parameters. The sample already mentioned can then be called under the Windows Command Prompt as follows:

```
cscript D:\Samples\Chapter01\Param.js //S /"Hello world" 1 2 3
```

Here I assumed that the drive letter D: is used for the script file. You can also dispense with the slash if you set the Host options before the name of the script file. **Figure 1.15** shows the commands executed in a Windows Command Prompt window.

> **NOTE:** Since the individual parameters are optional, at least the name of the script file must be specified in the command line. Also the slash (requested in the Microsoft Windows Script Host documentation) to mark a parameter isn't needed. And the // options may be mixed in with the arguments to be passed to the script.

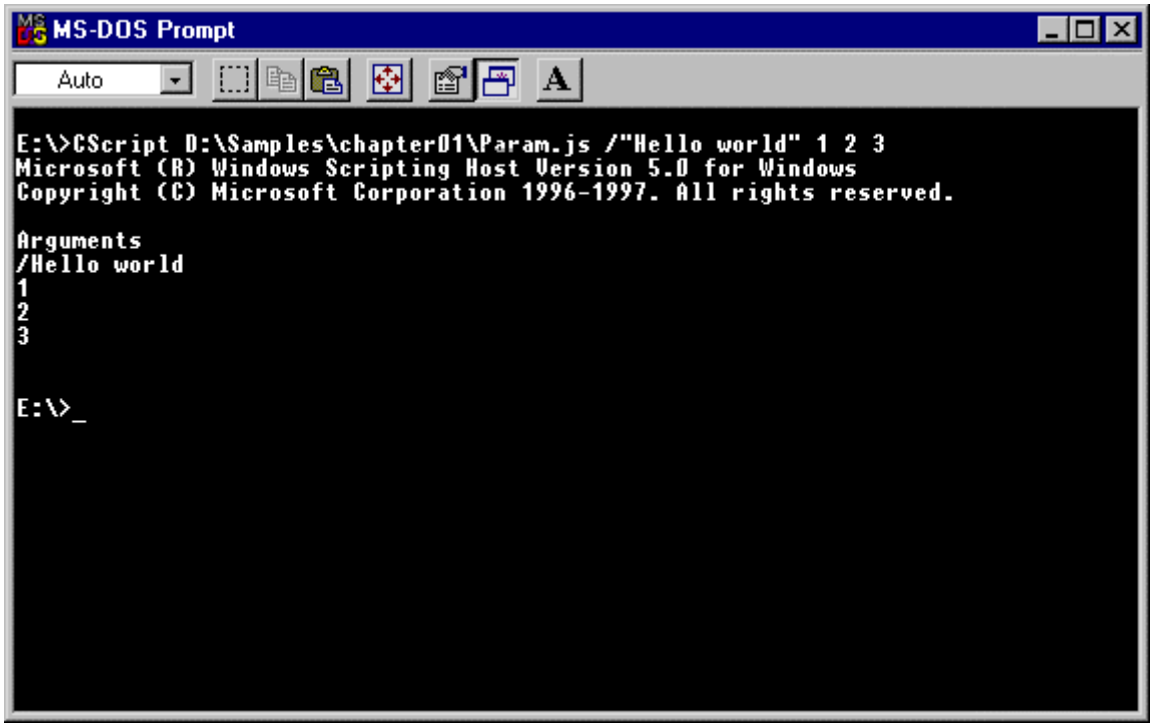

Figure 1.15.
Call a script with arguments in a Windows Command Prompt window

I have used the JScript sample script in **Figure 1.15**. This version uses the *Echo* method provided by the W*Script* object to show the submitted parameters. Because we use *Cscript.exe* within a Windows Command Prompt window, the *Echo* method sends all output to Windows Command Prompt window (which is standard behavior for a Windows console program).

> **TIP:** You can use this method together with I/O-redirection to send your script output into a file. The command *Cscript D:\Test.vbs "Hello" 1 > D:\test.tmp* creates the file *D:\test.tmp* and directs all script output into this file.

Details about the implementation of the JScript program may be obtained from the following listing.

```
//*********************************************
// File:    Param.vbs (WSH-sample in JScript)
// Author:  (c) G. Born
//
// Display the script arguments in a dialog box.
//*********************************************

 var objArgs;
 var text = "Arguments \n";

 var objArgs = WScript.Arguments;       // create object

 for (var i=0; i < objArgs.length; i++) // loop for all arguments
```

```
  text = text + objArgs(i) + '\n';      // get argument

  WScript.Echo (text);  // show arguments using Echo-method
  WScript.Quit();
//*** End
```

Listing 1.5.
Accessing parameters in JScript

> **NOTE:** You may find the sample program *Param.js* in the folder *\Samples\Chapter01* of the samples. Further information about this script will be discussed later.

Remarks about the scripting samples shipped with Microsoft Windows 98

Windows 98 comes with a few scripting samples. These sample files may be found after installing WSH in the Windows folder *\Samples\Wsh*. The table below contains a few notes about the function of these samples.

File	Remarks
Chart.vbs *Chart.js*	Demonstrates how you can access Microsoft Excel (Chart-Function) via the Windows Script Host.
Excel.vbs *Excel.js*	Creates an Excel-Table and writes some Windows Script Host properties into this table.
Network.vbs *Network.js*	Uses the W*SHNetwork* object, reads the network properties (user name and computer name), sets a network connection and lists the mapped network drives. Afterward the network connection is terminated.
Registry.vbs *Registry.js*	Shows how you can access the Windows registry to read and write entries.
Shortcut.vbs *Shortcut.js*	Uses the W*SHShell* object to create a shortcut on Windows desktop.
Showvar.vbs	Collects some system information and lists the Environment variables on a machine.

Table 1.2.
WSH samples in Windows 98

> Additional information may be found in the help file *wshadmin.hlp*, which is located in the folder *\tools\reskit\scripting* on Windows 98-CD-ROM.