2 Development Tools

This chapter introduces tools and techniques to simplify script development. I will introduce some editors, some techniques to get further information about ActiveX controls and some debugging tools.

A few tips to edit scripts

Below I will discuss how to create and edit script files. I will introduce some techniques, which simplifies this step.

How to create a script?

A script can be prepared directly using the Windows Editor (Notepad). Launch Notepad and enter the statements using the syntax of the selected script language. I have discussed this technique already in chapter 1.

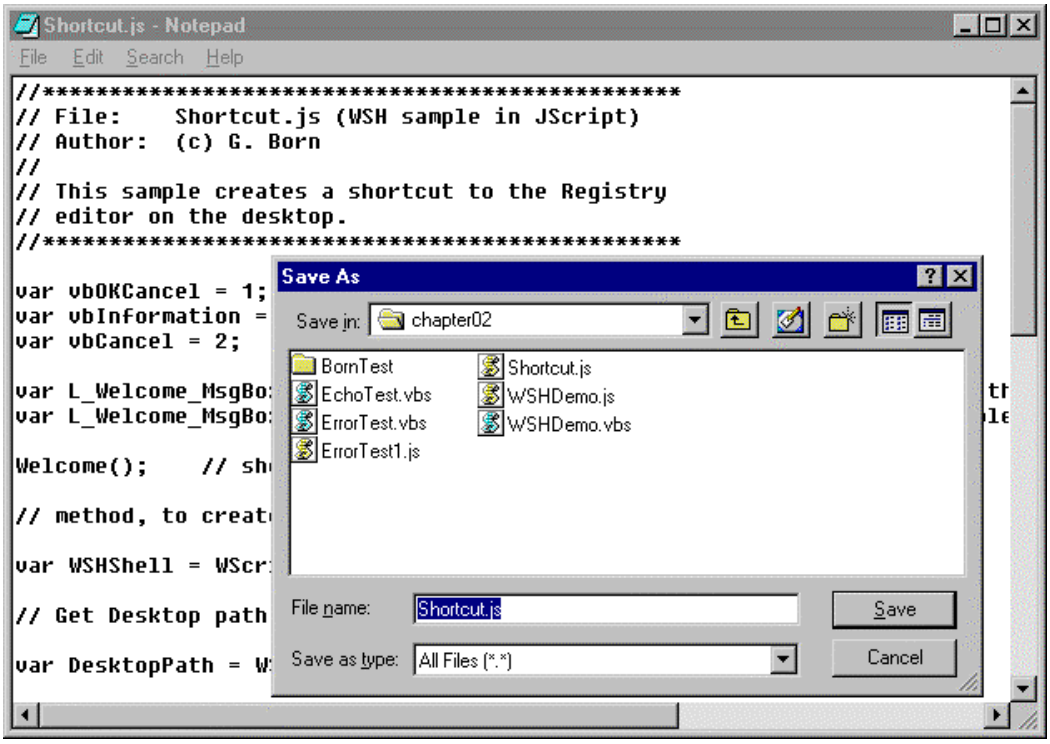


Figure 2.1.
Save a script file within the Windows Editor

Afterward store the script program in a script file. For VBScript use a text file with the file name extension *.vbs*, and for JScript use the file name extension *.js* (**Figure 2.1**).

> **TIP:** Set the *Save as type* within the *Save As* dialog on *All files (*.*)*. In this case the *Save As* dialog shows you all script files already stored in the destination folder.

Building a script from scratch isn't the optimal way. If you use for instance the header – I have proposed in chapter 1 – you must add these comments each time you create a new script. In chapter 1 I mentioned the template files *VBScript.vbs* and *JScript.js* you can use to build script files. What do you think about expanding the Windows shortcut menu with two new commands? These two commands enable you to create VBScript or JScript files based on the templates in any folder (**Figure 2.2**).

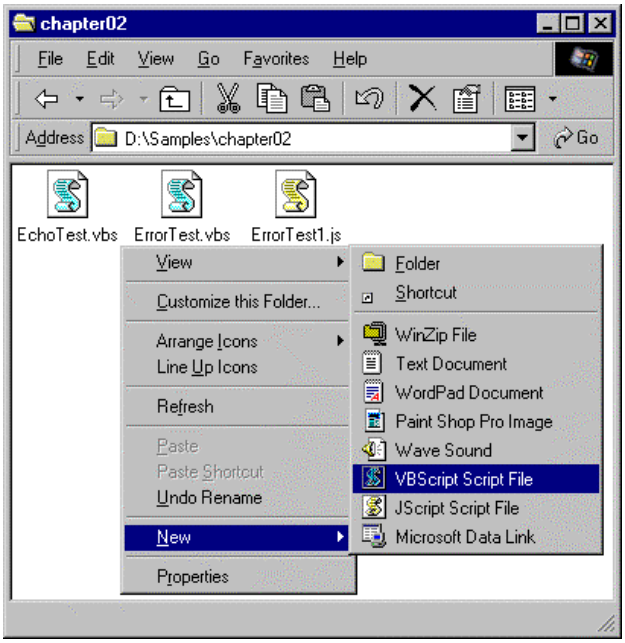


Figure 2.2:
Creating a new script file using the shortcut menu

Basically you have already all resources to do this – just configure Windows in a way that it knows script templates:

1. Open the folder *\samples\chapter01* from the sample files. This folder contains the two templates files (*VBScript.vbs* and *JScript.js*) mentioned in chapter 1. Copy both files into a local folder on your hard disk.
2. Open the Control Panel, double-click on the Tweak UI icon and select the *New* property page.
3. Drag one template file from your folders window onto the *New* property page (**Figure 2.3**). If you release the left mouse button, Tweak UI creates a new entry for the template on the property page.
4. Repeat this step for the second template file.

Closing the *New* property page using the *OK* button causes Tweak UI to copy the files into the Windows folder *\ShellNew* and register the new templates.

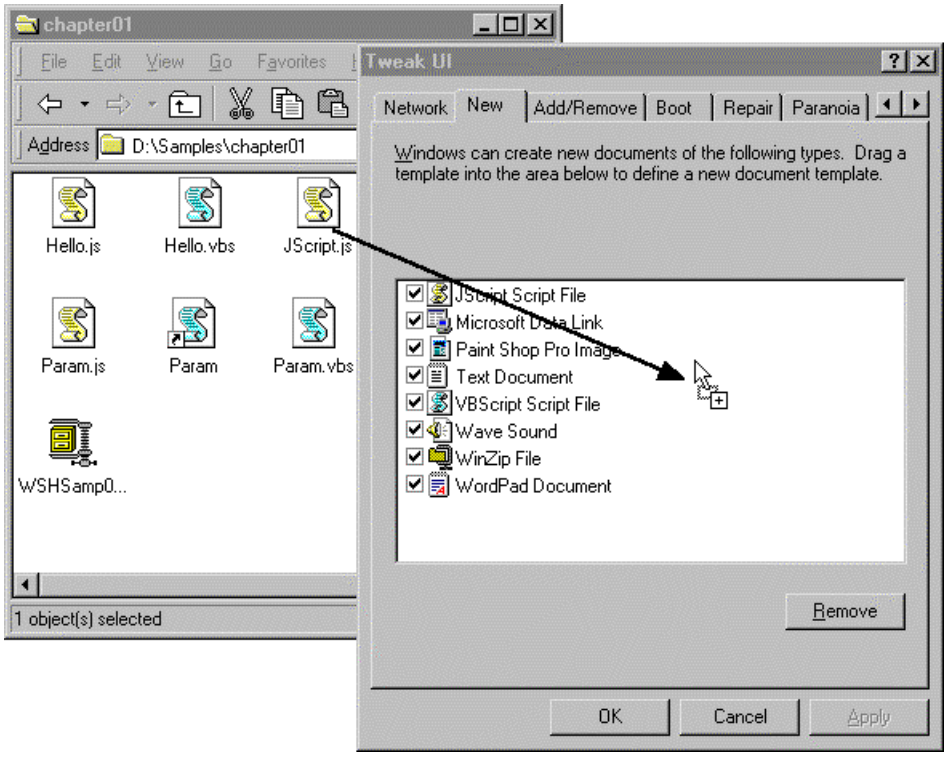


Figure 2.3.
Registering a script template using Tweak UI

After processing these steps you will find two new commands in the *New* shortcut menu (see **Figure 2.2**). You can use these commands in any folder to create a new script file.

> **Note:** Tweak UI is a Windows tool to customize the operating system. You must install Tweak UI explicitly. Windows 98 users may find this module on the Windows 98 CD-ROM in the folder *\tools\reskit\powertoy*. Windows 95 and Windows NT 4.0 users may download a version of Tweak UI from several Internet sites (search *www.microsoft.com* for the most recent version of Tweak UI). To install the tool, browse the directory to the folder with the Tweak UI install files. Right click onto the file *Tweakui.inf*, and choose *Install* in the shortcut menu. After the installation the Tweak UI icon is shown in the control panel folder.

Edit existing script files?

To load an existing script file into the Windows editor right click the file and choose the *Edit* command in the shortcut menu. During installing the Windows Scripting Host the *Edit* command is added automatically into the Windows registry.

Define your own *Edit script* command

Maybe you like to use your own script editor instead of Notepad. Then you need to launch this editor and load the script file. During script development it comes handy, if you have something

like a customized *EditScript* command within the file's shortcut menu. This command shall invoke the script editor of your choice.

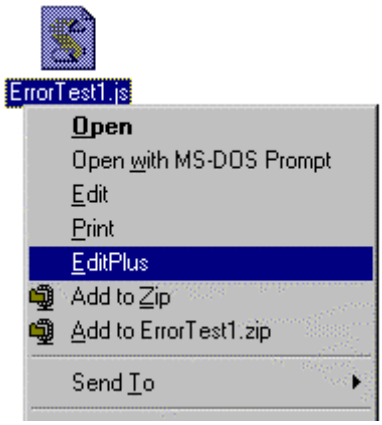


Figure 2.4:
Shortcut menu with a command to edit a script file

Figure 2.4 shows the new *EditPlus* shortcut command, that may be used to open a script file (.vbs or .js) within the program *EditPlus* introduced below. To register this shortcut menu extension use the steps described below:

1. Select *Folder Options* in the *View* menu within the folder window (Windows 9x).
2. Search the registered file type entry for your script file (*JScript Script File* for instance) on the *File Types* property page (**Figure 2.5**, left). Click the file's entry and use the *Edit* button on the property page.
3. Click *New* within the *Edit File Type* dialog (**Figure 2.5**, lower right)
4. Enter the new shortcut menu name into the *Action* text box within the *New Action* dialog (**Figure 2.5**, upper right).
5. Enter the command (path and exe file name to invoke the application) in the *Application used to perform action* text box. The characters %1 are placeholders for the current file (and will be expanded from Windows during command execution).

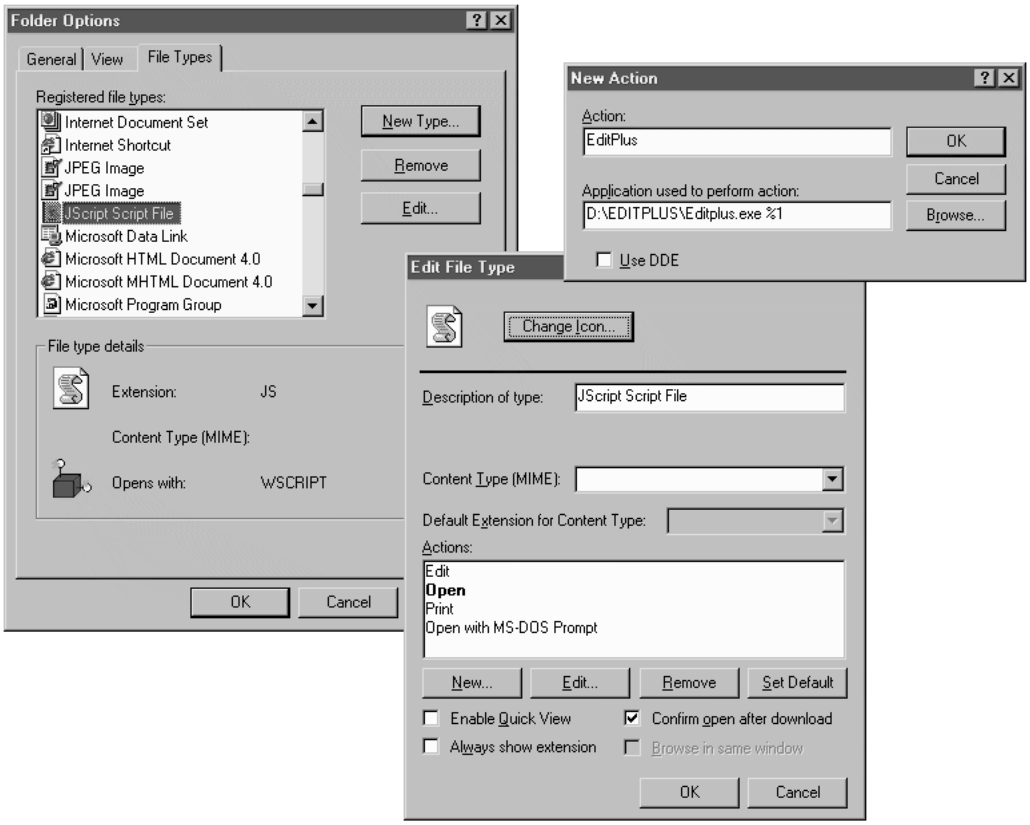


Figure 2.5.
Registering a new shortcut command

Closing all dialogs and property pages registers the new command for this file type. If you select such a file type afterward with a right-click, the new shortcut command is shown.

> **NOTE:** Keep in mind to register both file types (*.vbs* and *.js*) for scripts. Further information about registering file types and commands may be found in the title *Inside the Microsoft Windows 98 Registry, Microsoft Press, ISBN 1-57231-824-4*.
>
> **TIP:** A shortcut command *Print* may be used to print the source code of your script files. If you use one of the editors introduced below you may print the source code also with line numbers, which will be helpful during program debugging.

Script Editors

You may use the Windows editor Notepad for creating and editing script files. Unfortunately this tool provides only rudimentary features to edit a text file. The real problems occur during script debugging: The Windows Scripting Host parses the source code, and if a faulty statement is found, or if a runtime error occurs, an error dialog indicates the error code together with a line number (see **Figure 2.6**).

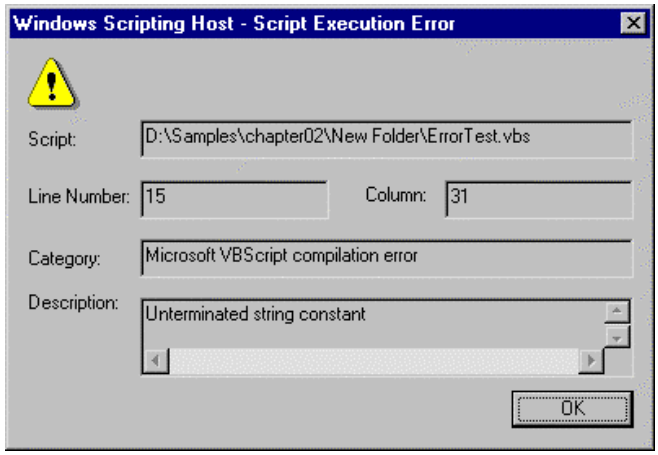


Figure 2.6.
Error dialog shown within script execution

Then you must edit the script file. Unfortunately locating the faulty statement may be awkward using the Windows editor, because you have to count the source code lines manually. This can't be done for lengthy scripts. Why not let the editor do this task for you? The following pages present some script editors, which support line numbering.

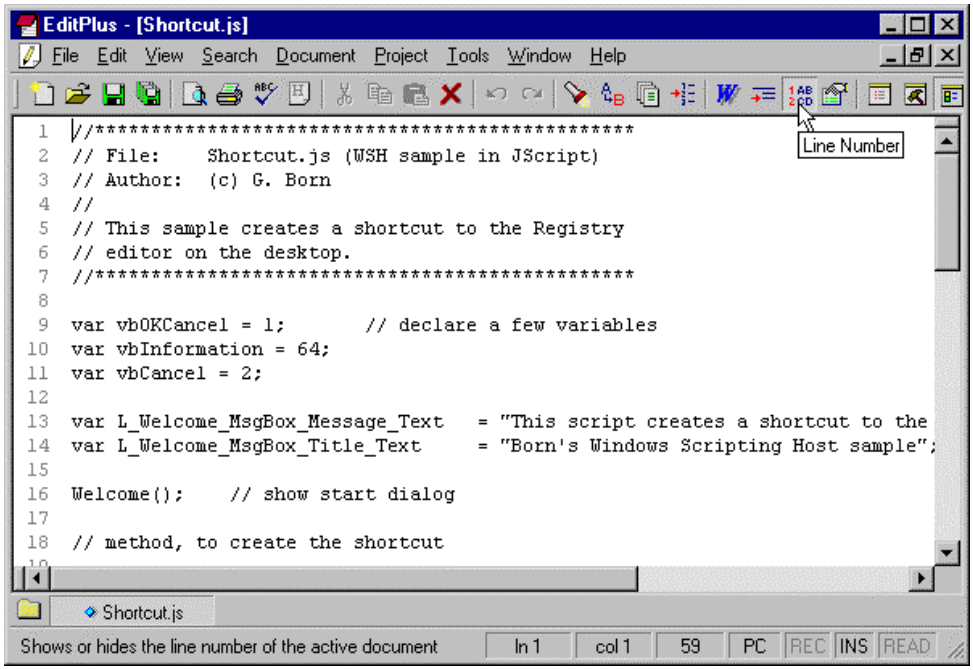


Figure 2.7.
EditPlus supports line numbering

EditPlus

EditPlus is an Internet-ready 32-bit text editor for Windows 95/98 and NT 4.0, which is distributed as top-quality shareware (ZDNet Software Library for instance). The editor provides a syntax-highlighting feature already available for HTML, C/C++, Pearl and Java, which can be extended to support other languages too. And the most important feature: this editor contains a toolbar with a button enabling/disabling line numbering within the loaded text file (**Figure 2.7**).

> **NOTE:** You may download a 30-day evaluation copy of EditPlus from *http://www.editplus.com*.

PrimalSCRIPT script editor

PrimalSCRIPT is a powerful script editor for Microsoft Windows, developed by SAPIEN Technologies, Inc. If you are developing WSH scripts or scripts for different languages, PrimalSCRIPT is a powerful and versatile editor. The tool provides a consistent user interface and development environment for several scripting languages. One of the most interesting feature is the possibility to insert code snippets (*for ..., if ... else* and so on) into the source for all supported languages.

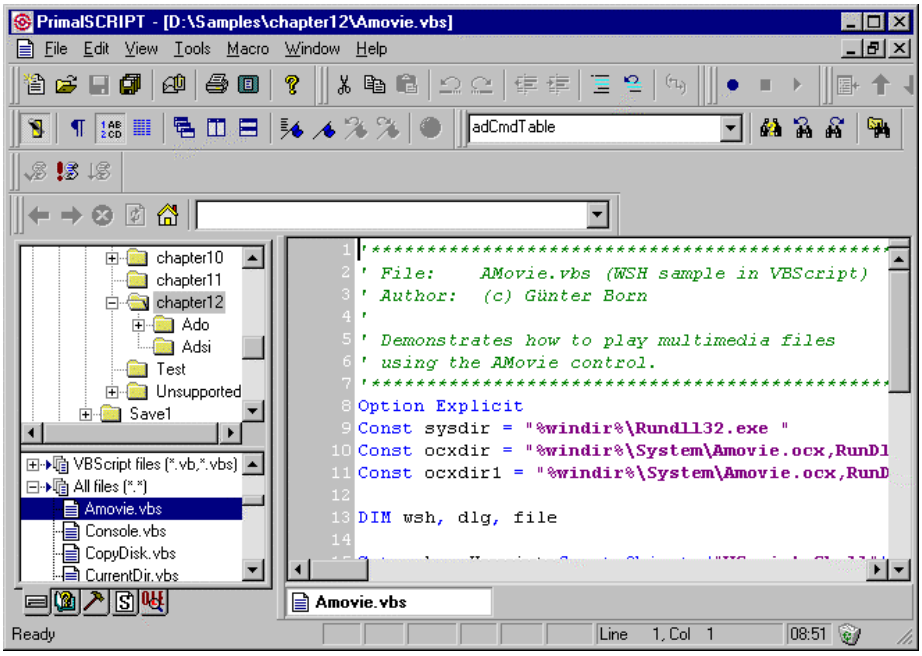


Figure 2.8.
PrimalSCRIPT script editor

Version 1.0 I have tested during writing the German version of the WSH book doesn't support line numbering. Instead the current line is shown in the status bar and you may use the *Jump line* command to locate the requested line.

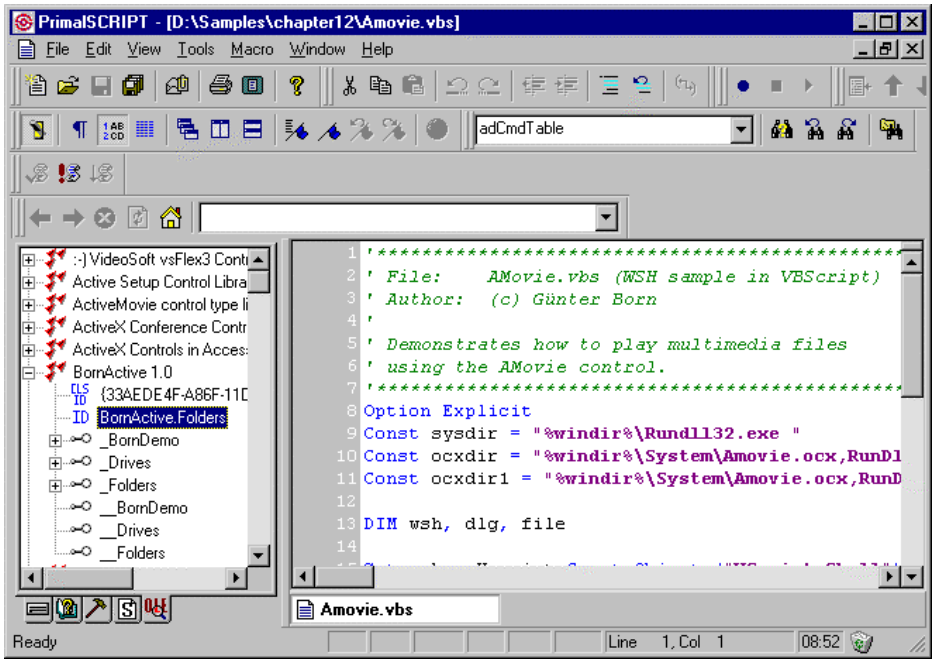


Figure 2.9:
PrimalSCRIPT version 2.0 showing the interface definition for object libraries

The vendor provided me with a preliminary version 2.0 of this tool to write the English version of this book. Version 2.0 comes with line numbering, a COM interface viewer and much more features like debugging support for WSH. You may download a 30 days evaluation copy from their web site *www.sapien.com.* This site contains also a »Script Exchange« section where programmers can get new sample scripts.

Other editors

There are some other editors, which may be used for script editing (but I haven't tested it). Take a look at CodeMagic. This tool is also a scripting IDE, it is free (in the first version) and fairly customizable. The program may be downloaded from *http://www.petes-place.com/codemagic.html*. TextPad and NoteTab are both shareware editors NoteTab also has a Light version that's freeware. Both editors doesn't support text color highlighting.. You can download TextPad from *www.textpad.com* and NoteTab from *www.notetab.com.*

The OLE/COM Object Viewer

As soon as you would like to use external automation objects within a script, you need information about their interface definition. You must know for instance the program identifier (shortly also called *ProgID*), to use the *GetObject* or the *CreateObject* method. Furthermore it is important to know, whether the automation object is registered already under Windows. The Microsoft OLE/COM Object Viewer provides these information. This is a tool written by Microsoft developers to analyzes the Windows registry and shows you the object classes, the application IDs, the

Type libraries and the interface definitions (which are not of interest for script programmers) available on the current system (**Figure 2.10**).

Using the program is similar to the Windows Explorer. At program start the viewer shows you in the left pane the available classes obtained from the Windows Registry. Double-clicking an icon opens the sub-branch. If no further sub branches exists, the program shows the registered program information of the selected icon in the right pane of an associated entry. These information are obtained from the Registry branch *HKEY_CLASSES_ROOT*. For script developers these information may be quite interesting.

You get for instance the ProgID (Programmatic identifier), under which the concerned component is registered in Windows. This ProgID is required to access the object. Furthermore the viewer indicates you the path of the file of the component. These are EXE-, DLL- or OCX-files, from which the functions of the component are loaded. Using this path you can find out very easily, whether the concerned component is still required and where the associated files are stored. This comes handy during uninstalling ActiveX components for instance (see below). Furthermore you need the path to an OCX-file, if you try to obtain information about the automation interface of an object using the object browser.

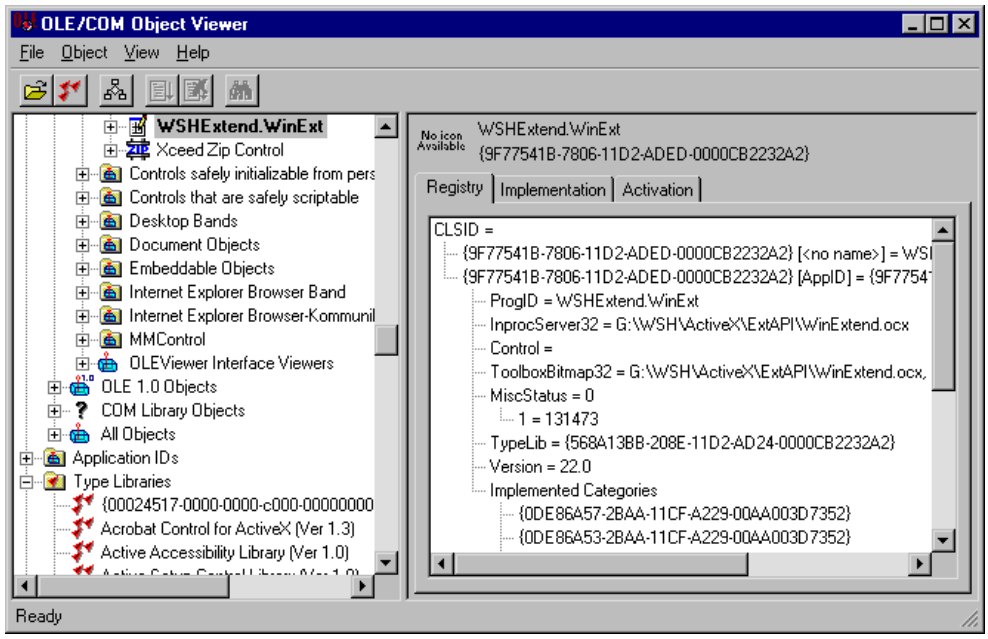


Figure 2.10.
The OLE/Com Object Viewer

> **NOTE:** The OLE/COM Object Viewer is part of several Microsoft development packages (for instance Visual Studio). You may also download the viewer from the site *http://www.microsoft.com/com/resource/oleview.asp*.

Using the Object Browser

One of the largest problems for a WSH programmer are caused from the missing information about the features of automation objects. Of course, the best support comes with the help files shipped with a component. But not all programmers ship an online help. Even as I begun with scripting, not too much Microsoft documentation about WSH was available. At this time I spend many hours to read several books about ActiveX programming, I consulted the VBA programmer's title I wrote for Microsoft Press and other sources. At this time I had a lot of questions like: Which objects are available? Which methods do I need? How to get some information about the properties? If you had an idea, which object to use, you need further information about its interface, its methods and properties. How to get these information, if no documentation is available?

The entire concept behind Windows automation for OLE, COM, ActiveX etc. is based on the attempt that all automation objects communicate mutually over certain interfaces. Without bearing you with too much details, we can assume that Windows must keep information somewhere about the installed automation objects. And there must be a way to obtain an interface description from the automation objects (to negotiate the interface with other components).

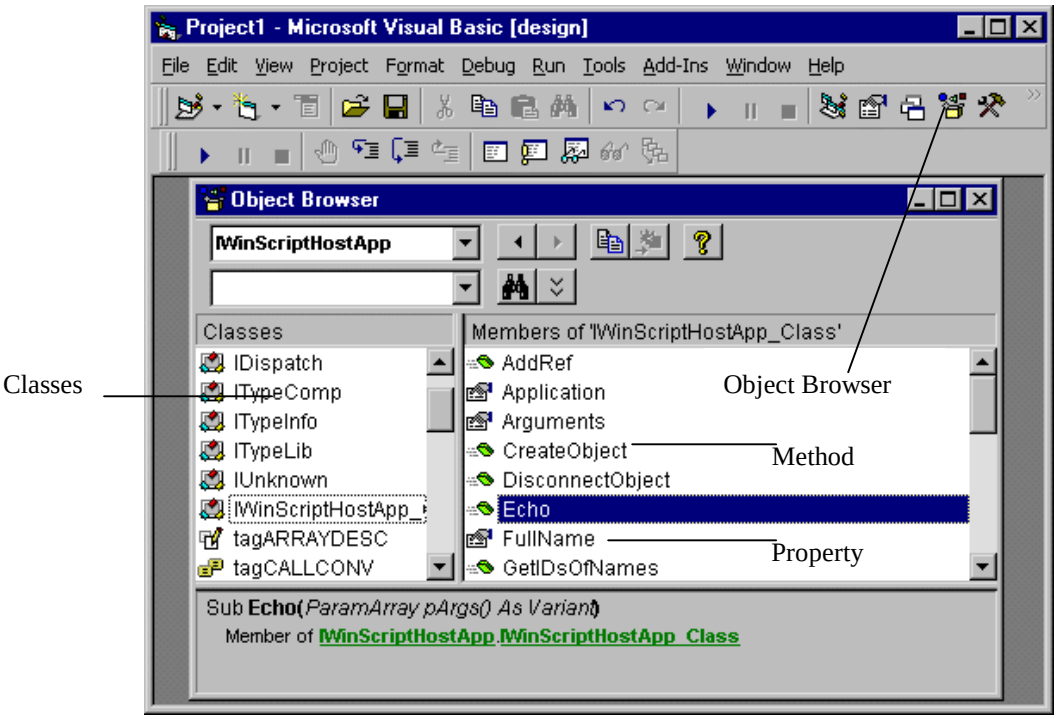


Figure 2.11.
VB 5 CCE with Object Browser window

On the previous page I mentioned that the program OLE/COM Object Viewer retrieves some information from the Registry. Inspecting such an entry delivers information about the files of a component, you get the ProgID (which is used in *CreateObject* to create a reference to an object). To find out more about an object, we need a tool, which shows all information in a human readable format. One tool that can do this for you is the Object Browser provided in the Visual Basic development environment. And the Visual Basic editor is also included in all Microsoft Office

applications (Word, Excel etc.) or in the Visual Basic 5.0 Controls Creation Edition (VB 5 CCE). If you have one of these applications you must enter the Visual Basic editor (press Alt+F11 in Word, Excel etc.). The Object Browser may be invoked using the *Object Browser* button in the toolbar of the development environment window (**Figure 2.11**).

> **NOTE:** The Visual Basic 5 Control Creation Edition (VB 5 CCE) may be downloaded from Microsoft's Web site (*http://www.microsoft.com/gallery/tools/visualbasic/vbcce.asp*). The VB 5 CCE provides a Visual Basic 5.0 development environment without the function for preparing standalone EXE files. As soon as you install the VB 5 CCE, you dispose of the concerned VB development environment. We will use this environment in this book in later chapters to prepare ActiveX controls to extend the Windows Scripting Host capabilities.

The Object Browser lists in the left pane all classes (objects) within a project. If you select a class, the right pane shows the methods (marked with a green block) and properties (indicated with a stylized hand) supported by this object.

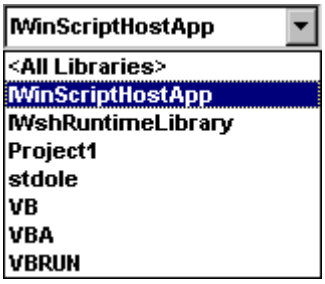


Figure 2.12.
Selecting libraries

The objects shown in the *Classes* window may be selected within the Object Browser's list box (**Figure 2.12**). If you select the entry »All Libraries«, the Object Browser shows all objects (classes) known in the development environment. Or your can select another entry to restrict the viewed content to a class library. The right pane shows the methods and properties of the selected object of the class.

Click an object to retrieve the properties and methods within the Object Browser. If you select an entry (method, property) in the right pane, the definition of the entry is shown in the footer of the Object Browser's window. For a property you get the information, whether these are read only items, about the data type and to which class the property belongs. For a method the Object Browser delivers in addition information about the interface along with the required parameters. In **Figure 2.11** the interface definition of the *Echo* method is shown for instance.

The Object Browser indicates only the classes (objects as well as their methods and their properties), which are currently registered within the development environment. If you need information about a special component, you need to define a reference to the required object library (before you open the Object Browser). Use the following steps:

1. Open the *References* dialog box. The *References* command may be found in the VB 5 CCE in the *Project* menu. In the VBA-development environment you must choose the item *References* in the *Tools* menu.

2. **Figure 2.13** shows the *References* dialog (upper left). The list *Available References* contains the names of the libraries, which are known in the development environment. A checked

checkbox in front of an entry indicates the associated Library is used within the project. Therefore these checked libraries will also be shown in the Object Browser. If the *Available References* list contains a library that you want to use, check the checkbox in front of the library name.

3. If the required library is missing in the *Available References* list, click the *Browse* button. A second dialog box is shown (**Figure 2.13**, right). Browse for the requested library file and select it within this dialog box.

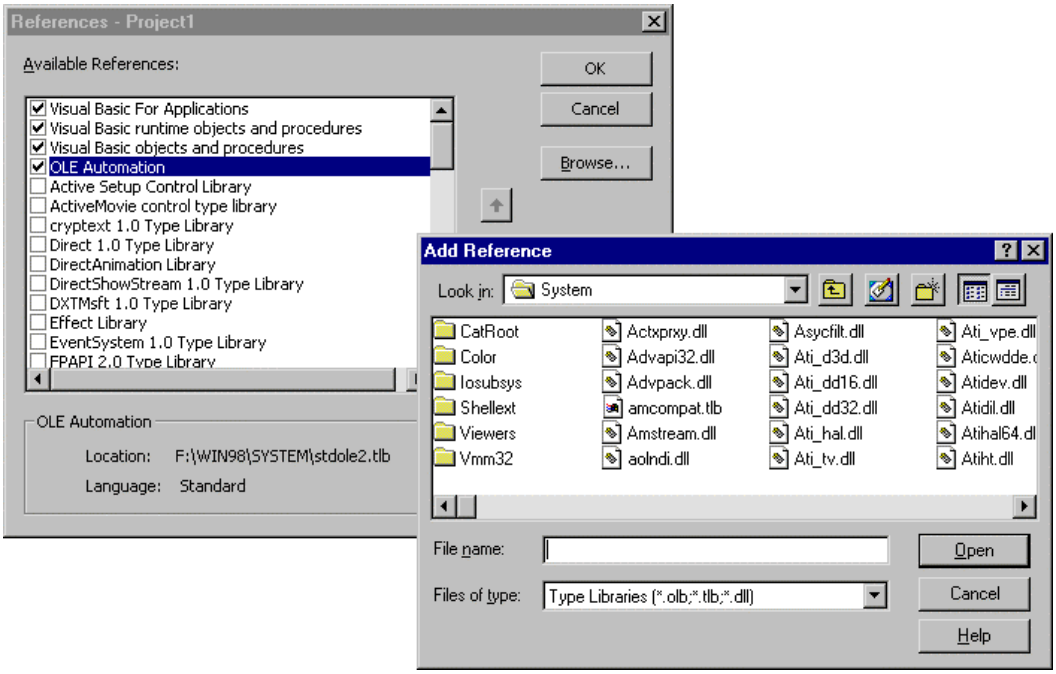


Figure 2.13.
Defining a new reference to an object library

After closing the *Add Reference* dialog box using the *Open* button, and after checking the associated checkbox in the *References* dialog box, the Object Browser shows the classes, methods and properties contained in the object library.

A few other tips

Above I had mentioned already in the OLE/COM Object Viewer section that this tool indicates the path to the file(s) of a registered automation component. This file contains however the description of all exported interfaces. As soon as you know the location of the file, you can set the path to the registered automation component. There are several file types, which may be referenced to get the interface definitions:

- EXE, DLL: These files contains the executable program code of the out of-process-server.
- OCX: These files are used to save the code for ActiveX controls.
- TLB, OLB, DLL: These files contain Type Libraries: *Type Libraries* (TLB), *Object Libraries* (OLB) and *Dynamic Link Libraries* (DLL).

These file types can be selected over the File type list box within the dialog box. The only difficulty exists therein, that you must know the path to the concerned files. Furthermore not each file delivers the required information in great detail. But I feel, this function is extremely helpfully to retrieve information about an object, its methods and properties.

Installing/uninstalling ActiveX controls

Within a script you may use objects provided by applications like Microsoft Word or Microsoft Excel. Other objects are exposed by the WSH itself of by the Windows operating system. Many objects are provided from ActiveX controls. Some of these ActiveX controls may be downloaded from the Internet. In some cases these controls comes with a setup program that allows you to install/uninstall the component.

> **NOTE:** Some authors of ActiveX control include these control into a website. If the browser detects a missing ActiveX control, a message box asking you whether to install this component. If you click *Yes*, the component will be installed on your machine. The installed ActiveX control can be used also in WSH scripts. Uninstalling this control is discussed below.
>
> Also, if you develop an ActiveX control using a development environment (like Visual Basic or the VB 5 CCE), the component is registered automatically on the development machine and the control is stored in a OCX file.

Before you can use a downloaded ActiveX control within a script, you need to register this control. Do you dispose of an ActiveX control (an OCX file), which isn't registered yet? If the developer of the OCX component doesn't provide an install program which overtakes the necessary steps to register the object, you need to use the program *RegSvr32.exe*. Microsoft ships this program with several applications and operating systems. To register the content of an OCX file as an ActiveX control, use the following command:

RegSvr32.exe C:\Samples\chapter02\BornTest\BornActive.ocx

The program locates the file using the path, analyses files content and adds all requested information into the Registry. After this step you can use the ActiveX control within a WSH script.

Figure 2.14.
Options for RegSvr32

NOTE: *RegSvr32.exe* is part of all Microsoft development environments. Fortunately a copy of this program is also shipped with Windows 98. You will find the program in the Windows *System* folder. If you invoke this program with the following command:

RegSvr32.exe /?

an error dialog with all possible options for this program is shown (**Figure 2.14**).

TIP: Within the sample files there is a file *\Samples\chapter02\BornTest\OCXReg.reg*. Double-clicking this file once extends the shortcut menu of your OCX files with two new commands *Register OCX* and *Unregister OCX*. One command registers the ActiveX control contained in the OCX file, and the other command unregisters the ActiveX control. The folder contains also the web page *Active1.htm* that registers the OCX control *BornTest.ocx* automatically, if you view the page in the Internet Explorer.

Licensing ActiveX controls

Whilst I was developing the samples of this book I discovered a major problem – the licensing of ActiveX controls. It is not a problem to register an ActiveX control using *RegSrv32.exe*. The CD-ROM shipped with Microsoft Windows 98 contains several OCX-files, which might be rather helpful for a script developer (for instance the file *MSInfo.ocx*). I have had downloaded also several OCX files from websites. After installing these OCX files, I failed to use the objects provided from these ActiveX controls. Each attempt to access an object after executing the *CreateObject* method results in the run-time error »Win32-Error 0x80040112«. Unfortunately this error code is not documented. So it took me a while till I found out that this error indicates a missing license info for the control.

Microsoft started a while ago to add a kind of license check to its ActiveX controls. Such ActiveX controls only works on the target machine, if a valid license key is detected. For some ActiveX modules it was sufficient to install the VB 5 CCE, for others I need to install Visual Studio 6.0. For other ActiveX modules I need to install Microsoft Office 97. For instance, the ActiveX control *Mswinsck.ocx* is part of the Microsoft Office 97 Developers Edition. Because I own this edition, I tried to register the OCX file manually using *RegSvr32.exe*. After getting the licensing error mentioned above, I tried to re-install the whole Microsoft Office package, without any positive result. The OCX control could not be used. Only after a clean install of the whole system I was able to run the tests. This behavior is caused from the Microsoft Office Setup program that checks the system during the installation process only for missing components. Components already installed or present will be left without updating on your machine. So re-installing a feature won't fix the bad situation of a missing license key. Also many vendors of ActiveX controls use this license keys. ActiveX controls from these vendors are only useable on the development machine.

And I need to mention another problem. Updating to a new version of some Microsoft products (like updating to VB 5.0 or VB 6.0) causes licensing conflicts again, because the new controls requires a different license key as the controls from the previous version. Because Setup won't change Registry entries for existing components, the license keys won't be updated properly. To solve this problem, Microsoft provides some tools, which fix the wrong license keys. Microsoft's support pages (*www.microsoft.com*) contains two Knowledge Base-Articles Q181854 and Q194751, discussing such problems during upgrading new controls for VB 5.0/VB 6.0. The tool *Vbe.exe* offered for download fixes a few (but not all) problems with VB 5.0 controls. After installing Visual Basic 6.0 my problems with these controls was gone. Missing license information for updated VB 6.0 controls may be fixed using the tool *VB6Cli.exe* (downloadable from the same web site).

ActiveX controls works like an onion: internally they refer to other ActiveX controls. It is highly probable that the dependencies caused by this architecture results in missing components on the

target machine. Then you are not able to use the control. As long as such packages are distributed in whole, it will be no problem. But ActiveX controls are offered many times for download from websites. And then you take a high risk that not all parts of the »game« are already on your machine. Just another »joke« which caused me a few more gray hairs. During developing the samples for the German version of this book, I had installed Visual Studio for a week on my development machine. So I could use many ActiveX controls coming with this package. Then I send over parts of the samples to my technical editor for testing. And he reported that nearly all my samples using external OCX files won't work (for licensing reasons), till the VB 5 CCE is installed. And a few examples depends on an installed Visual Studio 6.0. So I kept my test machines as clean as possible (no Visual Studio installed), to recognize missing components. But this situation is really bad, especially for the non-experienced script programmers.

Uninstalling OCX files

Above it was shown, how to install an ActiveX control. After installing ActiveX controls using *RegSvr32.exe*, or after using a development environment like Visual Basic or VB 5 CCE to create your own controls, your system (and your Registry) becomes cluttered by installed ActiveX components. Thus raises the question, how to remove unused ActiveX controls? It's not sufficient to delete the OCX file. Instead you must uninstall the whole component, which means remove the Registry entries. Before you start to do this job with the Registry Editor, use the program *RegSvr32.exe* and enter the following command:

RegSvr32.exe /u C:\Samples\chapter02\BornTest\BornActive.ocx

The program locates the OCX file given in the path. The switch */u* forces the program to remove all entries for this ActiveX component from the Registry. Afterward you may delete safely the OCX file without the risk that unused entries remains in your system.

> **TIP:** If you import the REG file mentioned on the previous page, you can right-click on the OCX file and use the shortcut command *Unregister OCX*, to unregister the OCX file.

Uninstalling an OCX ActiveX control which was installed from the browser

Did you install an ActiveX component using a browser and a web site? The sample file *ActiveX1.htm* located in the folder *\Samples\chapter02\BornTest* does this with *BornTest.ocx*. In this case the browser stores the OCX file under Windows 98 in the folder *Temporary Internet files.* And the module is getting registered automatically. To uninstall an ActiveX module that is installed in this way you must process the following steps:

1. Launch the Internet Explorer and select *Internet Options* in the *View* menu (in Microsoft Internet Explorer 5.0 you need to use the *Tools* menu).
2. Select the *General* property page on the *Internet Options* property sheet and click on the *Settings* button (**Figure 2.15**, left).
3. The browser opens a second dialog box *Settings.* Click the *View Objects* button (**Figure 2.15**, right).
4. The *Downloaded Program Files* folder windows, which contains all program files downloaded and registered during an Internet session, is shown (**Figure 2.16**). Right-click the file, which you intent to remove and select *Remove* in the shortcut menu.

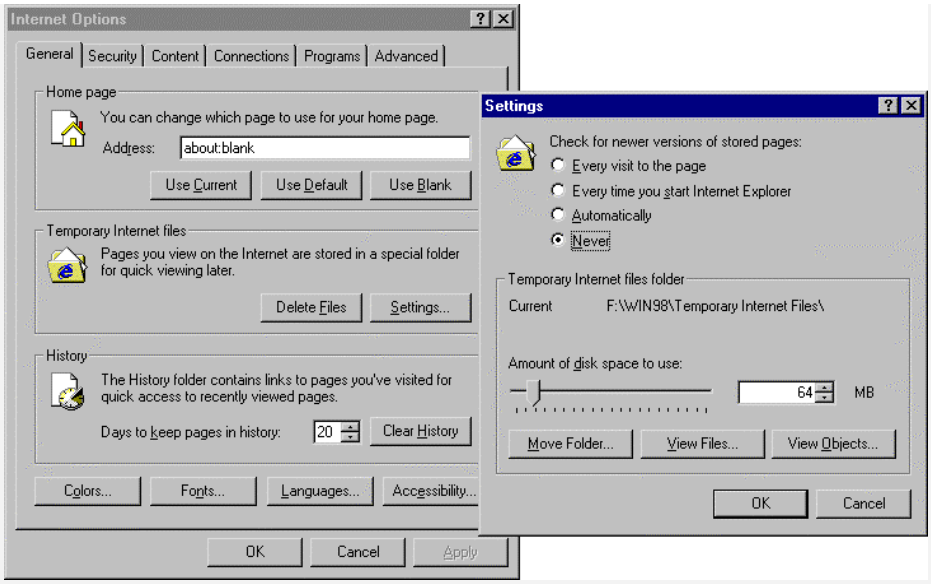
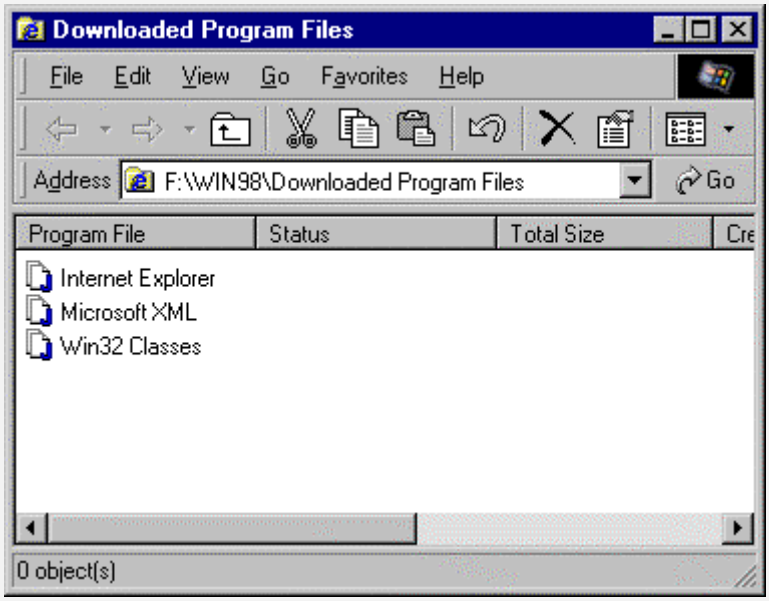


Figure 2.15.
Internet Options

The browser shows a third dialog box, which you must confirm by clicking the *Yes* button. Afterwards Windows uninstalls this object (delete the file and remove all Registry entries).

Figure 2.16:
Folder Downloaded Program Files



The objects in the *Downloaded Program Files* folder may be ActiveX objects stored as OCX files and other items like Java class modules stored in CAB archives. You may use the COM/OLE Object Viewer after uninstalling to check whether the ActiveX entries are removed from the Registry.

TIP: Some ActiveX objects come with its own install program. In this case you may use the entry provided in the *Install/Uninstall* property page (use the *Add/Remove Programs* icon in the control panel folder to invoke the property page).

Script debugging

A syntax error or a run-time error causes the WSH to terminate the script execution with the error dialog shown in (**Figure 2.17**). The error dialog contains the path and the name of the script as well as a note on the error category along with an error description. In most cases also the line number and the column, in which the error appeared, is shown.

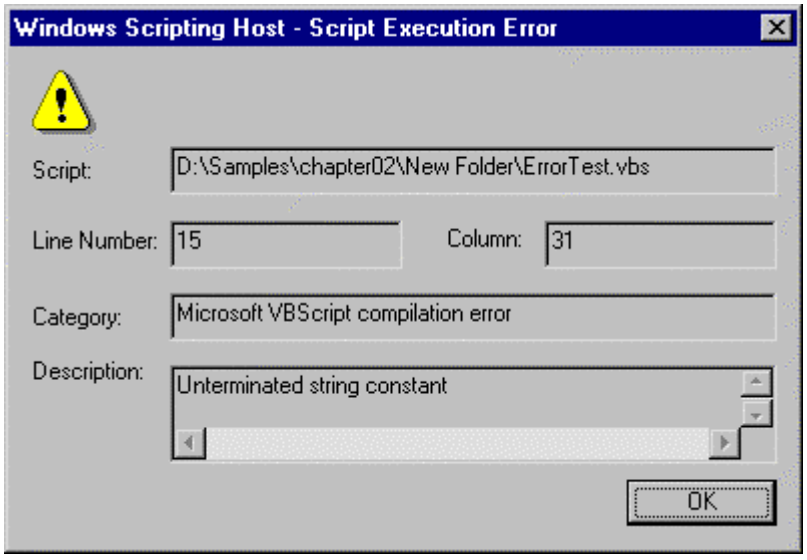


Figure 2.17.
Error dialog, which terminates a script's execution

Using one of the editors mentioned above may help to identify the line that causes the error. After amending the source code, you can run the next test. If the script doesn't contains syntax errors anymore, you may begin with debugging.

NOTE: For your own tests you will find the file *ErrorTest.vbs* in the folder *\Samples\chapter02* of the sample files. The VBScript program still contains a syntax error that causes the error dialog shown in **Figure 2.17**. A string constant in line 15 isn't closed with ".You must amend the source code to execute the script.

Trace your programs

After loading a WSH script faultlessly, you may start functional tests. Only in an ideal case the script delivers the expected result. Sometimes it is helpful to place statements to show a message box within the source code. Each time such a statement is executed, the message box will be displayed. Within the message box you can show interims results. This technology is shown in the following Listing.

```
'*********************************************
' File:   WSHDemo.vbs (WSH sample in VBScript)
' Author: (c) Günter Born
'
' Show the interims results within message boxes.
'*********************************************
' The following lines activates the trace messages.
' You must set the comment into the requested line.

' DebugFlag = false   ' trace output off
 DebugFlag = true     ' trace output on

 j = 0
 debug "Start", 0, 0

 For i = 1 to 10   ' loop 10 times
  debug "Step: ", i, j
  j = j + i        ' Add all numbers
 Next

 debug "End", i, j

 WScript.Echo "Result: ", j

 WScript.Quit

Sub debug (text, count, val)
 If DebugFlag Then   ' Debug mode active?
  WScript.Echo text, i , "Interims result: ", j
 End if
End sub
'* End
```

Listing 2.1:
Script with trace output

The program's function is quite trivial: The program uses a loop to calculate the sum of the numbers from 1 to 10. The results are shown subsequently using the *Echo* method.

For test purposes you would like to know however, when the program enters into the loop and how many steps are executed within the loop. To trace these steps the *Echo* method can be called up in the loop or at concerned places. To keep this sample as simple as possible, I have chosen here a particular attempt. All trace messages were evaluated in an own procedure with the name *debug*. Thus the program needs only the following statement:

```
debug "Start", 0, 0
```

to show the entry into the loop. The first parameter defines the text that is shown in the Message box. The other parameters may contain numbers, which are also shown in the message box. Inspecting the listing above, you will find this statement in several places. Within the loop this statement shows the index value and the calculated interims value (**Figure 2.18**).

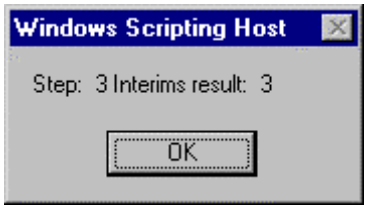


Figure 2.18.
Message box with trace values

Now I like to mention a particularity: As long as you test that program, the trace values should be shown according to **Figure 2.18**. After the script is tested and runs faultlessly, these trace messages are no longer required. The trace statements may be removed from the source code. But, if you change something later in the program, you need the trace statements again. This time consuming task is superfluous, because you can control the trace output using an option within the *debug* procedure. This procedure contains a condition like:

```
If DebugFlag Then    ' Debug mode activated?
```

Only if the value of the global variable *DebugFlag* is set to *true*, the *Echo* method is called and the message box is shown. The variable *DebugFlag* is set in the script's header to the value *true*.

```
' DebugFlag = false   ' trace output off
 DebugFlag = true     ' trace output on
```

One of the two statements contains a comment. So you can set the value of this variable to *true* or to *false* changing the comment sign from one line to the other. Depending on the value set for *DebugFlag* the trace output is shown or suppressed.

> **NOTE:** You will find the script *WSHDemo.vbs* (and *WSHDemo.js*) within the folder *\Samples\chapter02* of the sample files.

Using the Microsoft Script Debugger

The technique shown in the previous section may be applied only for simple scripts. This procedure however isn't appropriate, if a script causes run-time errors or unexpected results. Here we need a debugger to trace the program and its values. Microsoft offers the Microsoft Script Debugger for free download within their website. This program was developed primarily to test scripts within HTML documents or in Active Server of page (ASP files). However with some tricks the debugger can also be used to test WSH scripts.

> **NOTE:** You may download the Microsoft Script Debugger from Microsoft's web site *http://msdn.microsoft.com/scripting*. The debugger will be installed in the folder *\Programs\Microsoft Script Debugger*.

Testing scripts under the control of the Microsoft Script Debugger raise us to the question: How can we execute the script under the control of the debugger? The script engines support under WSH 1.0 some commands to invoke the Microsoft Script Debugger automatically.

- In VBScript you must insert the command *stop* within the source code.
- JScript requires the *debugger* statement to invoke the debugger.

Both commands stop the script execution and launch the Microsoft Script Debugger. Then the debugger may overtake the control over script execution.

Note

Note: In the WSH newsgroup I found some postings that in rare cases both statements won't work. The reason is not clear. If this case happens, there is a third method to use the debugger. Just place a statement like *WScript.Echo "start debugger"* into the script to invoke a dialog box. After invoking the dialog box (**Figure 2.19**), the script execution pauses until the user closes the dialog.

Figure 2.19.
Dialog asking to start the debugger

You can use this pause to launch the debugger manually and take over the control: Select the command *Running Documents* in the debugger's *View* menu and select the script's name in the *Running Documents* window (**Figure 2.20**) to take over the control. Then click onto the button *Break at Next statement*. Click the *OK* button to close the message box.

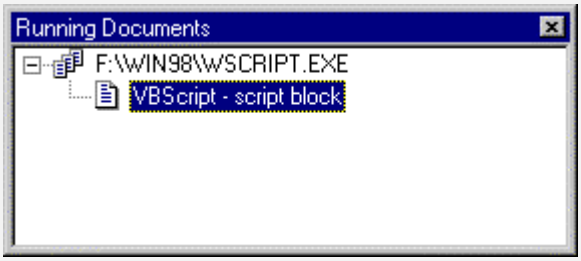


Figure 2.20.
Running Documents *window*

> **Note:** The two sample files *ErrorTest1.vbs* and *ErrorTest1.js* in the folder *\Samples\chapter02* contain a statement to show a message box. The two sample files *ErrorTest2.js* and *ErrorTest2.vbs* within the folder *\Samples\chapter02* contain the *stop* and *debugger* commands.

After the Microsoft Script Debugger has taken over the control to the script execution, the source code is shown in a window (**Figure 2.21**).

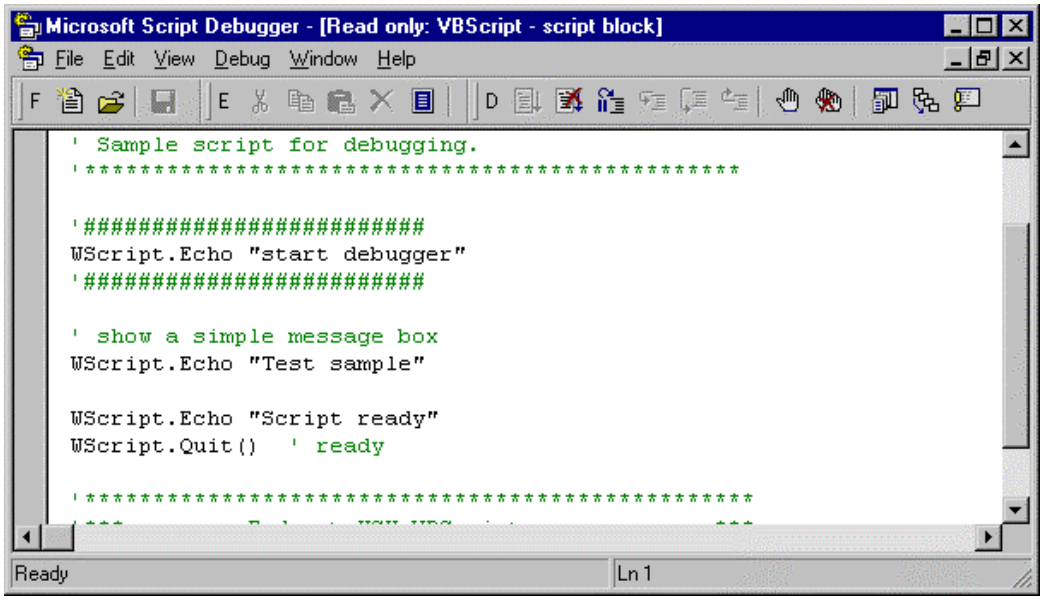


Figure 2.21.
Microsoft Script Debugger window with a sample script

Comments will be shown in green color. But you should note that you can't edit the source code within the debug window.

> **NOTE:** Only if you load the source code into the debugger, you may edit the statements within a separate window. The debug windows is indicated by the text »Read only« in window's title bar.

Microsoft Script Debugger commands

The Microsoft Script Debugger contains several toolbars and a menu with commands to test the script (**Figure 2.22**). Beside the *Break at Next Statement* command the debugger supports further commands. These commands may be invoked using either the menu *Debug* or the buttons within the *Debug* toolbar. Some commands may also be invoked by keystrokes:

- *Run:* Executes the script statements till a breakpoint is found or till the script terminates.
- *Stop Debugging:* Aborts the script execution and terminates debugging.
- *Break at Next Statement:* Click onto this button to ensure, that the script is not executed in a piece. This is necessary, if you use method 1 mentioned above to invoke the debugger and take control over script execution manually. In this case the debugger stops script execution on the next statement, if the user closes the dialog or message box.
- *Step Info:* Select this command (within the *Debug* menu) or using the button in the toolbar or pressing the key, to execute the next script statement.
- *Step Over:* This command may be invoked using the *Debug* menu, the button or the key combination Shift+F8. This command causes the debugger to execute a whole procedure or func-

tion till program control returns to the calling module. If the current statement doesn't belongs to a procedure/function, the debugger uses the *Step Info* mode.

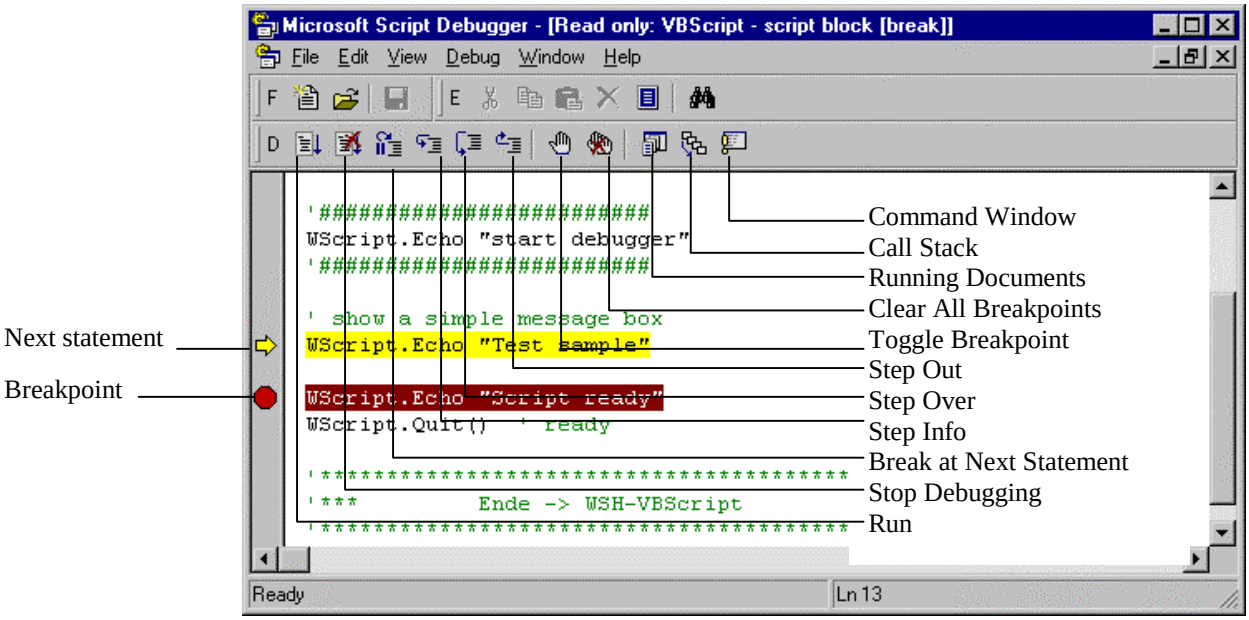


Figure 2.22.
Debug window with a breakpoint

- *Step Out:* Executes all statements within a procedure/function till the command is found which transfers the control to the caller.
- *Toggle Breakpoints:* Select an executable statement in the debug window and to set or clear a breakpoint within this code line either by clicking this button, using the *Debug* menu or by pressing the key. Lines with breakpoints are marked with a brown dot in the left border of the debug window. If a breakpoint is reached during script execution, the debugger stops.
- *Clear All Breakpoints:* This command clears all breakpoints set within the script.

The button *Command Windows* opens a window which allows you to enter and execute commands directly. The button *Call Stack* invokes a window showing the names of all active procedures on the stack. The stack is empty, if your script uses only a linear program structure (without calling procedures or functions).

Execute a script step-by-step?

During debugging a single step mode is rather helpful. Clicking onto the *Step Info* button or pressing the function key F8 executes the next statement in a script.

The next executable statement is market within the code windows with a yellow arrow on the left border (**Figure 2.22**).

Using the *Step Over* and *Step Out* mode

If your script contains already tested procedures and functions, a single step mode doesn't makes sense to execute the statements within these procedures/functions. In this case you may use the button *Step Over* or press the key combination Shift+F8. If the next executable statement contains a procedure or function call, the whole procedure/function will be executed without any disruption. The execution stops, after the control was returned from the procedure/function back to the caller. If you use this command to execute other statements which doesn't call a procedure/function, it works like the *Step Info* mode.

If the script execution terminates within a procedure or function, you may use the button *Step Out* or the key combination Strg+Shift+F8 to execute all statements within the procedure/function. The debugger takes over the control, if the return statement is reached.

Using breakpoints

To interrupt script execution on a certain statement, you may use breakpoints. Select the requested statement within the debug window with a single mouse click. Then you set the breakpoint using either the *Toggle Breakpoint* button or the key F9. This sets or clears a breakpoint within this statement. If you run program again, the debugger stops the execution, if the line with the Breakpoint is reached. Breakpoints are marked in the code window with a brown dot in the left border (**Figure 2.22**).

Showing the Call Stack

Using the command *Call Stack* in the *Debug* menu or pressing the button in the toolbar open the *Call Stack* window (**Figure 2.23**) This window contains a list of all active procedures/functions. In the current sample the procedure *debug* is executed.

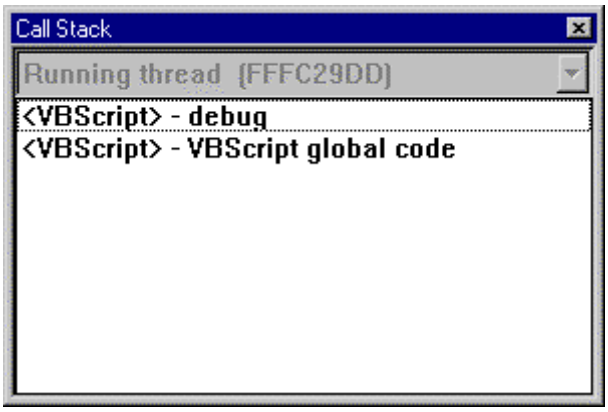


Figure 2.23.
Call Stack window

Show interims values using the *Command Window*

If a program is interrupted, you can inspect the current value of a variable (or a property). Open the command window. In VBScript you must type a question mark followed by a variable name to show the associated value (**Figure 2.24**).

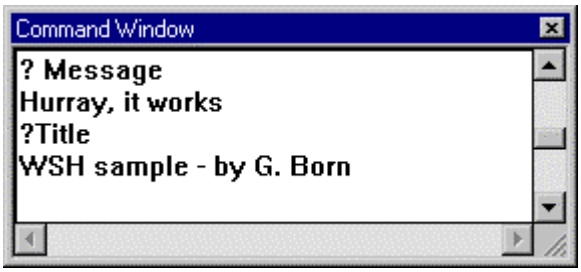


Figure 2.24.
Showing variable values within the command window

In JScript programs you must use a statement like *WScript.Echo text* to show a variable value. The debugger executes the statement and shows a message box with the value of the variable *text*.

> **NOTE:** If you would like to set the value of a variable, enter the statement to associate a value to a variable into the command window:
>
> ```
> Message = "Hello World"
> ```
>
> You can use the command window also to try any program statement (we just have used this with the *WScript.Echo* method).

The techniques mentioned above allow you to debug a WSH script written either in VBScript or in JScript in a comfortable way. But the debugger doesn't reach in his functionality the comfort offered in other Microsoft development environments. Consider also, that the Microsoft Script Debugger may not be used, if you have already installed Visual Interdev. This environment owns its own debugger, which is called up at the appearance of an error in a HTML and/or ASP script automatically.

> **TIP:** If Visual Studio or Microsoft Office 2000 is installed on your machine, the Microsoft Script Debugger won't work. If a *debugger* or *stop* statement is getting executed, of if a runtime error occurs, a message box asking for debugging is shown. Clicking the *Yes* button invokes the debugger of the Visual Studio Script Editor (which is also part of Microsoft Office 2000). You can use this debugger in a similar way as the Microsoft Script Debugger. The Script Editor supports a few additional debug commands. Details may be found in the program help of the program.