4Working with WSH objects

In the preceding chapter I have discussed a few basics of script programming. We have also used a few objects, methods and properties. In this chapter I would like to extend your knowledge how to use the Windows Scripting Host, to automate certain tasks. Take a look how to read the properties of the *WScript* object and display them in a dialog box. This allows you to retrieve the most important information from WSH and of the current script. Or access the environment variables of your operating system using a script. Another sample demonstrates how to access the arguments passed to a WSH script (a topic which we know from chapter 1). Accessing other objects requires creating these objects. Below I will discuss, how the methods *CreateObject* and *GetObject* are used with *WScript* objects. And I like to show how to launch an external application from a script using the *Run* method.

> **NOTE:** At this place I recommend download a copy of the WSH Programmers Reference from Microsoft's website *http://msdn.microsoft.com/scripting*. This reference comes handy for the upcoming chapters.

Using the *WScript* object

The *WScript* object is the *application* object of the Windows Script Host. This object is exposed automatically to the running script. So you need not to create a reference to the *WScript* object. The object exposes several methods and properties. In previous chapters we already have used the methods *Echo* and *Quit* of this object. Below I will show how you can access the object's properties.

Displaying WSH and script properties

In the previous chapter I mentioned that the Windows Script Host and the script currently running are available as the *WScript* object. If a script is execute in the Windows Command Prompt (the MS-DOS window) using *Cscript.exe*, the Host echoes the current version number within the command line. This version number is really interesting, since Microsoft just started to develop different versions of the WSH. Also properties like the path to the host program, the host's name and so one are surely informative in different cases. The table shown below contains a short overview about the properties exposed by the WScript object.

Property	**Description**
Application	Returns the IDispatch interface of the *WScript object.*
Arguments	Returns a collection object containing the script parameters.
FullName	Contains the full path to the host executable (*Cscript.exe* or *WScript.exe*).

Name	The friendly name of *WScript* (this is the default property).
Path	The name of the directory where the host (*Wscript.exe* or *Cscript.exe*) resides.
ScriptFullName	This is the full path to the script that is currently run within the Windows Scripting Host.
ScriptName	This is the file name of the script that is currently run within the Windows Scripting Host.
Version	A string containing the version of the Windows Scripting Host (not the version of the language engine).

Table 4.1.
Properties of the WScript *object*

> **NOTE:** The WSH Programmers Reference (*http://msdn.microsoft.com/scripting*) contains further details about all *WScript* properties.

The next sample script collects the host and the script properties. These results are shown in a dialog box (**Figure 4.1**).

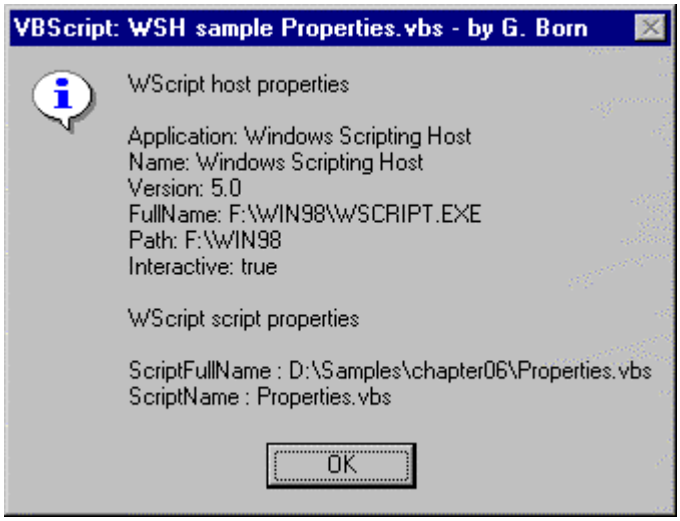

Figure 4.1.
WScript host and script properties

Retrieving the properties (only read access is allowed) of the *WScript* object is really simple. The *WScript* (application) object is exposed automatically from WSH to the script during script execution. Therefore the statement:

```
Name = WScript.Application
```

reads the *Application* property of the *WScript* object and assigns this value to the variable *Name*. If the script is executed under WSH, the variable shall contain the text »Windows Scripting Host«. Therefore you may use this property to check, whether the script is executed under WSH or not (a script could run for instance within the Internet Explorer, or as an external script of an Active Server Page (ASP) file).

NOTE: These properties are really helpful, if you need the name of a script or the path to the script file. Additional information about *WScript* properties may be found in the WSH Programmers Reference (*http://msdn.microsoft.com/scripting*).

Retrieving properties in VBScript

Listing 4.1 shows how you can retrieve the *WScript* properties using VBScript. You must specify the object name *WScript*, followed by a dot, followed by the property name. The result is assigned to a variable. In **Listing 4.1** I have used the variable *Message*, to collect all properties in one string. The constant *vbCrLf* splits the output into several lines within the dialog box. I have used also the *ScriptName* property in this sample to show the script name within the title bar (variable *Title*) of the dialog box.

```
'********************************************
' File:    Properties.vbs (WSH sample in VBScript)
' Author:  (c) G. Born
'
' Shows the properties of the WScript object
' within a dialog box.
'********************************************
Option Explicit

Dim Message
Dim Title

' Show the properties of the WScript object
' we start with Host properties

 Message = "WScript host properties" & vbCRLF & vbCRLF
 Message = Message & "Application: " & WScript.Application & vbCRLF
 Message = Message & "Name: " & WScript.Name & vbCRLF
 Message = Message & "Version: " & WScript.Version & vbCRLF
 Message = Message & "FullName: " & WScript.FullName & vbCRLF
 Message = Message & "Path: " & WScript.Path & vbCRLF

 ' Get the Interactive-Status
 If (WScript.Interactive) Then
   Message = Message & "Interactive: true" & vbCRLF
 Else
   Message = Message & "Interactive: false" & vbCRLF
 End if

' Get script properties
 Message = Message & vbCRLF
 Message = Message & "WScript script properties" & vbCRLF & vbCRLF
 Message = Message & "ScriptFullName : " & WScript.ScriptFullName & vbCRLF
 Message = Message & "ScriptName : " & WScript.ScriptName & vbCRLF

' init title
```

```
 Title = "WSH sample " & WScript.ScriptName & " - by G. Born"

 MsgBox Message, vbInformation + vbOKOnly, Title


'End
```

Listing 4.1.
Properties.vbs

NOTE: The file *Properties.vbs* may be found within the folder *\Samples\chapter04.*

Retrieving properties in JScript

For all readers preferring JScript I have also implemented the sample in this language. The properties may be read also using the *WScript* object, followed by a dot and the property name. The information is collected into the variable *Message*. New lines within the output text are marked with the "\n" escape sequence.

```
//*********************************************
// File:     Properties.js (WSH sample in JScript)
// Author:   (c) G. Born
//
// Show the properties of the WScript object
// within a dialog box.
//*********************************************
//

var Message, Title, tmp;
var vbInformation = 64;    // a few constants
var vbOKOnly = 0;

// collect the properties of the WScript object
// read the Host properties

 Message = "WScript host properties \n\n";
 Message = Message + "Application: " + WScript.Application + "\n";
 Message = Message + "Name: " + WScript.Name + "\n";
 Message = Message + "Version: " + WScript.Version + "\n";
 Message = Message + "FullName: " + WScript.FullName + "\n";
 Message = Message + "Path: " + WScript.Path + "\n";

// get Interactive-status
 if (WScript.Interactive)
  Message = Message + "Interactive: true" + "\n"
 else
   Message = Message + "Interactive: false" + "\n";

// get the script properties
```

```
 Message = Message + "\n";
 Message = Message + "WScript script properties \n\n";
 Message = Message + "ScriptFullName : " + WScript.ScriptFullName + "\n";
 Message = Message + "ScriptName : " + WScript.ScriptName + "\n";

// init title
 Title = "WSH sample  " + WScript.ScriptName + " - by G. Born";

 var objAdr = WScript.CreateObject("WScript.Shell");

 tmp = objAdr.Popup (Message, vbInformation + vbOKOnly, Title);

'

// End
```

Listing 4.2.
Properties.js

> **NOTE:** The file *Properties.js* is located in the folder *\Samples\chapter04*. Here I like to give also a 2nd remark. Did you noticed that the + operator is required in JScript to concatenate strings? In VBScript you can use also the + operator to concatenate sub strings. But this may cause trouble, because + stands for addition. Adding two strings is done as a concatenation operation. To create more transparent VBScript programs you should use however the & operator. Within this book you will find both operators, because I have ported some JScript scripts to VBScript.

Retrieving the script engine's properties

In the previous section I have discussed how to retrieve the host and the script properties. Beside these information however the properties of the language engines are requested in several cases. For instance, you may request the version number from the interpreter. This is helpful, because Microsoft has released several versions of the language engines. These engines depend on the libraries installed for the Internet Explorer. Microsoft Internet Explorer 4.0 provides version 3.x languages engines. Visual Studio updates these language engines to version 4.0, and Microsoft Internet Explorer 5.0 installs the version 5.0 of the language engines. And you can download version 5.0 of the language engines from *msdn.microsoft.com/scripting*. VBScript as well as JScript provide a few functions to check the version of a language engine:

- *ScriptEngine():* This function delivers a string defining the script language supported by the engine.
- *ScriptEngineMajorVersion():* The function returns the major version of the language engine as a string (3 for instance).
- *ScriptEngineMinorVersion():* Returns the minor version number of the language engine as a string.
- *ScriptEngineBuildVersion():* This function returns the Build number of a script engine.

Figure 4.2.
Script Engine properties

The next listing demonstrates how to use these functions in VBScript to query the properties of the language engine and displays it in a dialog box (**Figure 4.2**).

```
'************************************************
' File:    Engine.vbs (WSH sample in VBScript)
' Author:  Günter Born
'
' Display the version of the language engine.
'************************************************
Option Explicit

DIM txt

' get the version of the language engine

txt = "Script-Engine: " & CStr(ScriptEngine()) & vbCRLF
txt = txt & "Version: " & CStr(ScriptEngineMajorVersion())
txt = txt & "." + CStr(ScriptEngineMinorVersion()) + vbCRLF
txt = txt & "Build: " & CStr(ScriptEngineBuildVersion())

WScript.Echo txt

\              )

' End
```

Listing 4.3.
Engine.vbs

> **NOTE:** The file *Engine.vbs* is located in the folder *\Samples\chapter04*.

If you prefer JScript, you need to use the following listing to retrieve the properties of the language engine.

```
//************************************************
// File:    Engine.js (WSH sample in JScript)
// Author:  Günter Born
//
// Shows the version of the script engine.
//************************************************
//
```

```
var txt = "Script-Engine: " + ScriptEngine() + "\n" +
          "Version: " + ScriptEngineMajorVersion() +
          "." + ScriptEngineMinorVersion() + "\n" +
          "Build: " + ScriptEngineBuildVersion();

WScript.Echo (txt);

// End
```

Listing 4.4.
Engine.vbs

NOTE: The file *Engine.js* is located in the folder *\Samples\chapter04*.

Accessing script arguments

In chapter 1 I have discussed the techniques to submit parameters, also called arguments (for instance a filename or a switch), to a script. Within the script you need a technique to access these arguments. I have used the script in chapter 1 to display submitted parameters without explaining the details of the implementation. Now its time to have a look into the script. WSH provides the *WshArguments* object which may be used to handle the script parameters (arguments). The table shown below describes the properties associated with the *WshArguments* object.

Property	Description
Item	Default property, which defines the nth parameter in the command-line, used to call the script.
Count	Returns the number of command-line parameters (arguments).
length	This returns also the number of parameters and is used for JScript compatibility.

Table 4.2.
Properties of the WSHArguments *object*

How can we access these properties? Unfortunately the *WshArguments* object is not exposed directly. Instead we must use the *Arguments* property of the *WScript* object to access the script arguments. But it isn't trivial, because a simple assignment like:

```
Parameter = WScript.Arguments
```

causes not only a run-time error, it comes with a second difficulty. A script may receive more than one parameter, but the assignment statement shown above assigns only a single value to the variable.

At this point the object model comes handy. The *Arguments* property returns a collection object that is the *WshAguments* object we requested. Therefore you can use some code to access the indi-

vidual objects of this collection. There are several steps required to access the script parameters. These steps are shown below for VBScript. The statement:

```
Set objArgs = Wscript.Arguments
```

assigns the *Arguments* property of the *WScript* object to the object variable *objArgs*. We need the *Set* keyword, because the property is a collection object, so *objArgs* must be an object variable. You may use *objArgs* to access the objects and their properties of this collection. The first object of the collection may be accessed using the following statement:

```
param1 = objArgs(0)
```

In this case the variable *param1* receives the content of the default property of the object *objArgs(0)*. This is a new technique, because we have used the object name, a dot and the property name to retrieve a property. The argument strings are located in the *Item* property, therefore the statement may be written also as:

```
param1 = objArgs.Item(0)
```

Because *Item* is defined as the default property we can use also the short version of the command *param1 = objArgs(0)*. Both statements returns the script parameter stored in the *Item* property as a string into the variable *param1*.

Indeed there is still a further difficulty: How do we get the number of submitted arguments contained in the collection? If the script is executed without any parameter, the collection within the *Arguments* property is empty. The attempt to access the object variable *objArgs*(0) causes a runtime error in this case. According to **Table 4.2** the *WshArguments* object obtained for the *Arguments* property exposed also the *Count* property. This property may be used to check the number of items in the collection, which corresponds to the number of script arguments. Therefore you may use the following code to access the first script argument:

```
If objArgs.Count >=1 Then ' is there at least one argument
 Param1 = objArgs.Item(0)
End if
```

Accessing more than one or two parameters becomes a little bit too laborious with the code shown above. To access all script arguments, you should use a loop instead. The following VBScript code sequence shows how to get all arguments into a text variable.

```
For I = 0 to objArgs.Count - 1     ' all arguments
 text = text + objArgs(I) + vbCRLF ' get argument
Next
```

The number of items within the collection may be obtained with *objArgs.Count*. Then a simple *For* loop may be used to process all items. Accessing an item may be done with *objArgs(i)*, because *objArgs.Item(i)* is the default property.

Two solutions in VBScript

The code shown in **Listing 4.6** reflects my explanations given above. If parameters are submitted to the script, these parameters are shown in a dialog box that lists each argument found in a separate line (**Figure 4.3**).

Figure 4.3.
Displaying script arguments

This script retrieves the number of items within the *Arguments* collection. Then a simple *For* loop is used to access each entry within the collection.

```
'*********************************************
' File:    Param.vbs (WSH sample in VBScript)
' Author:  (c) G. Born
'
' Showing the parameters passed to the script
' within a dialog box.
'*********************************************

 text = "Arguments" + vbCRLF +vbCRLF

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

 Wscript.Echo text ' show arguments using Echo


' End
```

Listing 4.5.
Param.vbs

> **NOTE:** Submitting the parameters to the script may be done within a shortcut file or using the *Run* dialog box (or the Windows Command Prompt – see also in chapter 1). The file *Param.vbs* is located in the folder *\Samples\chapter04*. The shortcut file *Param_vbs.lnk* in the same folder defines a command line to call the script with predefined parameters.

The sequence shown above was used for demonstration purposes. In VBScript you have an alternative to evaluate submitted parameters within the loop. You can use a *For Each In* loop to enumerate all entries within a collection (or within an array). In this case counting the elements within the collection is up to VBScript. Let's assume we retrieve the *Arguments* object with the parameter collection using the following statement:

```
Set objArgs = Wscript.Arguments    ' create object
```

Then we can use the following sequence:

```
For Each i in objArgs      ' all arguments
  text = text & i & vbCRLF ' get argument
Next
```

The *For Each i in objArgs* statement is used to process each item within the collection. The loop index *i* contains the current object. Therefore you may use the variable *i* to read the script parameter. The assignment:

```
Param1 = i
```

assigns the argument to the variable *Param1*. The next listing shows the implementation of the VBScript script using a *For Each* loop.

```
'*********************************************
' File:    Param1.vbs (WSH sample in VBScript)
' Author:  (c) G. Born
'
' Show the parameter passed to the script within
' a dialog box.
'*********************************************
Option Explicit

Dim text, i, objArgs

 text = "Arguments" + vbCRLF +vbCRLF

 Set objArgs = Wscript.Arguments    ' create object
 For Each i in objArgs     ' all arguments
  text = text + i + vbCRLF ' get argument
 Next

  Wscript.Echo text ' show arguments using Echo


' End
```

Listing 4.6.
Param.vbs

> **NOTE:** Which of the two solutions you use depends on your personal flavor. The file *Param1.vbs* is located in the folder *\Samples\chapter04*. The shortcut file *Param1_vbs.lnk* in the same folder defines a command line to call the script with predefined parameters.

A solution in JScript

In JScript you may access also parameters using the *WScript.Arguments* property. The syntax of this language causes a few slight differences within the code. Accessing the *Arguments* property doesn't requires the *Set* keyword. JScript creates automatically the sub-data type for the requested value. Therefore we may use the following statement to get the *Arguments* collection:

```
var objArgs = WScript.Arguments;       // create object
```

This statement creates a variable and assigns the object reference. After getting the collection into the object variable, we can access the items. This may be done within the following code sequence:

```
for (var i=0; i < objArgs.length; i++) // all arguments
  text = text + objArgs(i) + '\n';     // get argument
```

Here we assign the content of the *Item* property (this is the default property of an object) of the current item into the variable *text* (see also my explanations on the previous pages). You should note that you must use the *length* property to estimate the number of items within the collection. This property is provided for compatibility purposes, because the *Count* property causes a run-time error in JScript. **Listing 4.7** shows the whole script. If you execute this script with parameters, these parameters are shown in a dialog box similar to **Figure 4.3**.

```
//*********************************************
// File:    Param.js (WSH sample in JScript)
// Author:  (c) G. Born
//
// Show script parameters in a dialog box.
//*********************************************

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

 var objArgs = WScript.Arguments;       // create object

 for (var i=0; i < objArgs.length; i++) // all arguments
  text = text + objArgs(i) + '\n';      // get argument

  WScript.Echo (text);  // show arguments per Echo
  \

// End
```

Listing 4.7.
Param.js

> **NOTE:** The file *Param.js* is located in the folder *\Samples\chapter04*. The shortcut file *Param_js.lnk* in the same folder defines a command line to call the script with predefined parameters.

Accessing environment variables

Windows 95/98 as well as Windows NT 4.0 and/or Windows 2000 stores several information within environment variables. The following section discusses how you can access these environment variables from VBScript or JScript.

Execute other programs from scripts

This sections discusses how you can use the *Run* method to execute commands and other programs from a script.

Remarks about the *Run* method

To launch an other application from a WSH script you must use the *Run* method provided by the *Shell* object. The *Run* method creates a new process, and executes the command contained in the *strCommand* parameter. A second parameter *intWindowStyle* may specify the window style. The method used the following syntax:

```
WshShell.Run (strCommand [, [intWindowStype] [,bWaitOnReturn]])
```

The parameter *strCommand* is required, because it contains the path and the name of the application or the command to be executed. I should note also, that the *Run* method expands environment variable names contained in the parameter *strCommand* automatically. The sequence:

```
Set WshShell = WScript.Shell ("WScript.Shell")
Command = "%WinDir%\Calc.exe"
WshShell.Run (Command)
```

causes the script to expand the environment variable %WinDir% contained in *Command* and launches the Windows program *calc.exe*. The other two optional parameters control how the application window must be shown and whether the script should wait till the executed process terminates.

intWindowStype is an optional parameter specifying the window style of the new process. The argument is of type *Variant* and may contain an integer between *0* and *n*. If the parameter *intWindowStype* is omitted, the window gets the focus and will be shown in normal mode.

NOTE: Unfortunately Microsoft's WSH Programmers Reference doesn't contain a description of the possible window styles. In a first step I did used the parameters defined for the VBA *Shell* command. The *vb* constants shown in **Table 4.5** are obtained from the VBA help files. Later on I detected, that a Microsoft Website contains a document with a description of the *Run* method. This document says that the parameter sets the *wShowWindow* element within the STARTUPINFO structure of the new process. Therefore the values are the same as for the parameter *nCmdShow* of the *ShowWindow* function. Based on this information, I have created **Table 4.5** with the values for the parameter *intWindowStype*. The column *constant* contains two names. The names with the prefix *vb* was obtained from the VBA help, whilst the values with the prefix *wm* belongs to the *STARTUPINFO* structure. But I should mention, that none of these named constants are defined under VBScript.

Constant	Value	Description
vbHide SW_HIDE	0	Hides the window, another window is activated (activate = is shown and get the focus).
vbNormalFocus SW_SHOWNORMAL	1	Activates the windows and shows it. If the process is already active and the window is minimized/maximized, the previous size and position will be restored.
vbMinimizedFocus SW_SHOWMINIMIZED	2	The window is activated. It is getting minimized and the button within the taskbar receives the focus.
vbMaximizedFocus SW_SHOWMAXIMIZED	3	The windows will be activated. It is maximized and receives the focus.
vbNormalNoFocus SW_SHOWNOACTIVATE	4	Displays a window in its most recent size and position. The active window remains active.
SW_SHOW	5	Activates the window in its current size and position.
vbMinimizedNoFocus SW_MINIMIZE	6	Minimizes the specified window and activates the next top-level window in the Z order.
SW_SHOWMINNOACTIVE	7	Displays the window as an icon (minimized), the active window remains active.
SW_SHOWNA	8	Displays the window in its current state, the active window remains active.
SW_RESTORE	9	Activates and displays the window. If a window is minimized or maximized, Windows restores the origi-

		nal size and position. An application should specify this flag when restoring a minimized window (can't be used with *Run*).

Table 4.5.
Values for intWindowsType

> **NOTE:** The values for the windows style in **Table 4.5** suggest, that the *Run* method you can both call up an instance of an application as well as switch an already running application in the foreground. Unfortunately there is a (small) problem at the *Run* method: This method creates always a new instance of the process. It isn't possible to re-activate a window of a running application or minimize/maximize it. The consequence: You can't use all values shown in the table above. At tests I found also, that the window style works only with applications supporting those styles. The Windows Editor accept the styles, whilst the calculator causes trouble (because the window can't be maximized). In chapter 8I discuss an ActiveX control which implements an object supporting a method to switch a background windows into the foreground.

The optional *Run* parameter *bWaitOnReturn* owns the sub-type *Logical* (it may contain the values *true* and *false*). The parameter controls whether the script wait on the termination of the executed process. If *bWaitOnReturn* is missing or set to f*alse*, the *Run* method executes the command and returns, if the command is send to the message loop. The next script statements can be processed. If *bWaitOnReturn* is set to *true*, the *Run* method creates a new process, executes the command and waits till the process terminates. In this case the *Run* method returns the error code obtained from the terminated process. If *bWaitOnReturn* is missing or set to *false*, *Run* returns the error code 0 (zero).

> **NOTE:** The error code may be set within a script using the *Quit* method (see below).

Launch the Windows Editor from VBScript

Let's create a few small scripts. The Windows Editor window supports the window styles shown in **Table 4.5**. The script shown below demonstrates how to launch the Editor *Notepad.exe* from VBScript. The path to the Windows folder will be obtained from the environment variable %*WinDir%*. Therefore this script is independent from the Windows install location. In a second step the script launches the Editor again. Within this step the Editor window is minimized (to a button in the taskbar). And the editor should load the source code of the script currently executed.

```
'*********************************************
' File:    Run.vbs (WSH sample in VBScript)
' Author:  Günter Born
'
' Calls the Windows-Editor using the Run method.
'*********************************************
'
DIM WshShell

Set WshShell = WScript.CreateObject ("WScript.Shell")

WshShell.Run "%Windir%\Notepad.exe", 1
```

```
WScript.Echo "Load source code in a minimized edit window"

WshShell.Run "%Windir%\Notepad.exe " + WScript.ScriptFullName, 6

WScript.Quit()  ' terminate script

' End
```

Listing 4.14.
Run.vbs

NOTE: The file *Run.vbs* is located in the folder *\Samples\chapter04*.

How to call the Windows calculator from JScript?

This sample uses JScript to launch the Windows program *calc.exe*. The path to the Windows folder is obtained from the environment variable *%WinDir%*. The script is independent from the location of the Windows folder. The script need some named constants, which I have declared as variables within the program's header. But note that the Windows calculator doesn't support all window styles. Therefore the style to minimize the window is ignored. If you port a VBScript sample to JScript, you must not only take care to the different syntax rule. Keep also in mind that the parameters for the *Run* method must be set into parenthesis. Also you have to use the \\ code within paths to separate folder names, because JScript interprets the \ as an escape sequence (see the JScript language reference provided from Microsoft). Further details may be found in the listing shown below.

```
//*********************************************
// File:    Run.js (WSH sample in JScript)
// Author:  Günter Born
//
// Launches the Windows-calculator using
// Run. Attention: The Run method doesn't
// supports all window styles!
//*********************************************

var SW_SHOWNORMAL = 1;
var SW_MINIMIZE = 6;

var WshShell = WScript.CreateObject ("WScript.Shell");

// First try
WshShell.Run ("%Windir%\\Calc.exe", SW_SHOWNORMAL);

WScript.Echo ("Launching calc minimized");

// Should not work !!!
WshShell.Run ("%Windir%\\Calc.exe", SW_MINIMIZE);

 WScript.Quit() ; // terminate script
```

```
// End
```

Listing 4.15.
Run.js

> **NOTE:** The program *Run.js* is located in the folder *\Samples\chapter04*.

A few remarks about the Quit method

Within the previous sections I have already used the *Quit* method without further explanations at the scripts end. This method terminates the object (this means: This method is used to terminate the script). Within a script you may call this method using the following statement:

```
WScript.Quit();
```

in JScript or:

```
WScript.Quit
```

in VBScript. First comes the object name, on which the method must be applied. The statements shown above uses the *WScript* object, which is exposed automatically by WSH. The object name is followed by a dot, followed by the name of the method *Quit*. *Quit* accepts an optional parameter, which must be set in a parenthesis following the method's name. The sample used above doesn't pass parameters to the *Quit m*ethod, the parenthesis in the JScript statement remains empty.

> **NOTE:** The parameter used in the *Quit* method defines the *Process-Exit-Code.* If no value is passed, the Windows Scripting Host returns the value 0 to the operating system. Alternatively you may insert an integer value (0 or 1 for instance) as a parameter. This code indicates to the operating system whether the process terminates with an error. The value 0 indicates that the process terminates in a regular manner. Omitting the parameter means the code is also 0. All positive values, which are above 0, indicates an error that causes the process termination. Windows doesn't check this code, but you can check the termination code for instance, using the host *Cscript.exe* to execute the script in the Command Prompt window (MS-DOS window). The error code may be checked using the ERRORLEVEL function within a batch program.
>
> It is not mandatory to insert the command. If the language engine reach the last line within a script, the process terminates automatically. But you may use the *Quit* method to terminate a script on a defined point.

Wait for process termination and check exit codes

The next sample discuss how to launch an external application and wait till the process terminates. And we use the explanation given in the previous paragraphs to check the return code. If the process returns an error code during termination, this *Process Exit Code* shall be examined by the parent script (the code shall be shown in a dialog box). To simulate a second process, we execute the following short VBScript program. This program uses the *Echo* method to show a dialog box. After the user clicks the *OK* button the script terminates. So you can check whether the parent script waits to the termination of the child process. The script *Test.vbs* returns the error code 2 using the *Quit* method. The *Quit* method supports error codes between 0 and 255. **Listing 4.16** contains the statements for *Test.vbs*.

```
'*********************************************
' File:    Test.vbs (WSH sample in VBScript)
```