3 Simple Dialogs in WSH

Most WSH scripts show results in dialog boxes or ask for user input. This chapter introduces a few techniques offered in JScript and VBScript to create dialogs within WSH scripts.

Using the Echo method

I have mentioned already the *Echo* method to show user dialogs within the Windows Scripting Host. You may use this method to show messages boxes from VBScript as well as from JScript in a simple manner. Now let's have a second look into the details. The *Echo* method uses the following syntax:

```
Object.Echo Parameters
```

Object is here the Windows Scripting Host with the object name *WScript*. Therefore the next statement uses the *Echo* method of the *WScript* object:

```
WScript.Echo ....
```

The *Echo* method may be called with one or more parameters, separated by commas. These parameters are used to pass the information shown in the message box. I will show the handling of the *Echo* method in the next two samples.

Remarks about objects, methods and properties

WSH script programmers use objects, methods and properties. An object can be an application (like the Word *application* object), a document, a paragraph within a document, a form, or something else. The Windows Script Host exposes also the *WScript* (application) object to the script. Some object may contain itself other objects. For instance a text document may consist of paragraphs, so a paragraph is a sub object of the document object. Because the document may contain a collection of paragraphs, the document is a collection object that enumerates the paragraphs. An application may contain a document as a sub object, the document may contain paragraphs, a paragraph may consist of sentences or word. The hierarchy of objects is called the object model. Starting from the root, the object model describes the hierarchy of several objects. You need to know the object model to access a single object.

Objects may have properties. For instance an application window may have a width and a height, a caption in the title bar, a background color and so on. Properties may be read or written (if the property is writeable). Methods are functions that may be applied onto the object. The *WScript* object offers for instance the *Echo* method to create an output.

Another method of the *WScript* object is the *Quit* method. This method is used to terminate a script. Methods requests parameters. In VBScript the parameters are passed as:

```
WScript.Echo "Hello world"
```

whilst in JScript the parameters must be passed in parentheses:

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

Using *Echo* in VBScript

Would you like to display a simple message box in VBScript? The *Echo* method offered by the *WScript* object is exactly the right tool for this. The Listing shown below demonstrates how to use this method.

```
'*********************************************
' File:    Echo.vbs (WSH sample in VBScript)
' Author:  Günter Born
'
' Uses the Echo method in WSH
'*********************************************
Option Explicit
' declare several variables
DIM price, vat, net

vat = 16.0
net = 100.0
' calculate price

price = net * (1.0 + vat/100.0)

' Display results - because the WScript (application) object is
' automatically available, we need no further steps.

WScript.Echo "Price: ", price, "US $ Tax: ", vat, _
             "% ", price - net, " US $"

' End
```

Listing 3.1.
Using the Echo *method in VBScript*

Figure 3.1.
Dialog box created from the Echo *method*

First there are a few variable declarations. Then a calculation is made, and the result shall be shown in a dialog box. The *Echo* method may be used to display the final result. Using the *Echo* method is more than simple. You must combine the *WScript* object (which is exposed automatically from the WSH) with the name of the method:

```
WScript.Echo "Price: ", price, "US $ Tax: ", vat, _
             "% ", price - net, " US $"
```

> **NOTE:** A dot is used to separate the object name *WScript* from the name of the *Echo* method. This notation is used also to separate the object's name from a property name.

In VBScript you must append the required parameters without parenthesis to the *Echo* keyword. Commas must separate several parameters. The result created from the listing above is shown in **Figure 3.1**.

> **NOTE:** The file *Echo.vbs* is located in the folder *\Samples\chapter03* in the sample files.

Echo and the Windows Command Prompt

You can you launch your script also within the Windows Command Prompt (MS-DOS window). If you use *Cscript.exe* as host, the output of the *Echo* method is send to the MS-DOS command line (**Figure 3.2**). Because *Cscript.exe* is a console-oriented application, *Cscript.exe* doesn't show a dialog box.

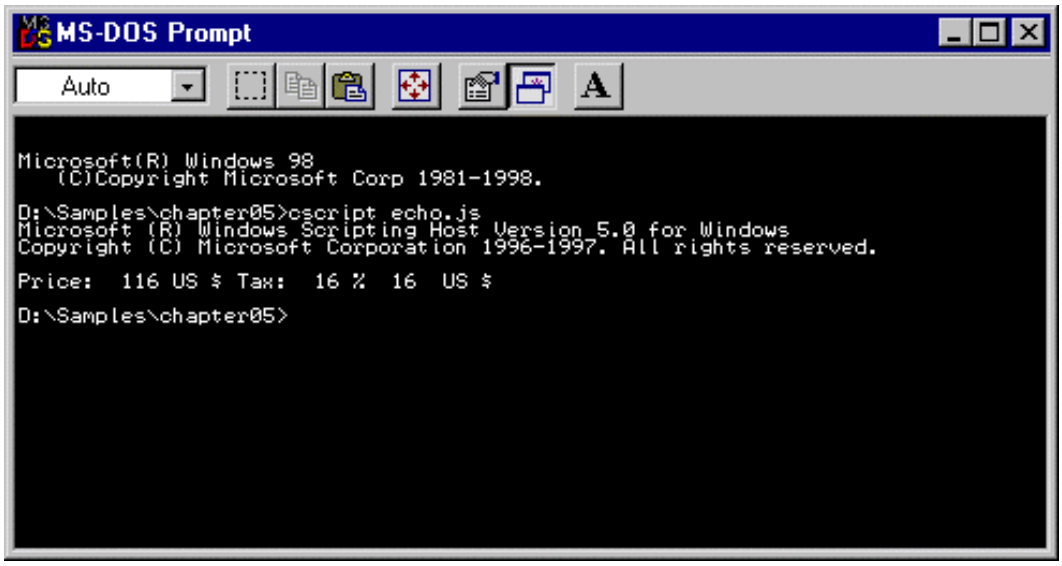

Figure 3.2.
Echo *output in MS-DOS*

This behavior can be used for a special purpose: If you need to write results obtained by a script into a file (for instance a kind of logging), you can use the *Echo* method within your VBScript or JScript scripts to display the results. If you execute your script in the Windows Command Prompt (in the MS-DOS window) using *Cscript.exe*, the output is send to the Command prompt.

> **NOTE:** Console applications write to the *stdout* device, so all of the *stdout* redirection operators work (>, >>, and |).

So you can redirect the *Echo* output into a text file using the following command:

```
Cscript //nologo echo.js >logfile.txt
```

This command causes two things. The *//nologo* switch suppress the two logo lines shown on the Windows Command Prompt window after the scripting host starts (see chapter 1). The redirection *>logfile.txt* causes MS-DOS to write all program output created with *Echo* into the file *logfile.txt*

instead displaying it in a window. Appending the output to an existing file requires the >> characters for redirection.

Using the *Echo* method in JScript

The *Echo* method presents the only possibility in JScript to show a message box without creating further objects (because the WSH exposes automatically the *WScript* object). The dialog shown by the *Echo* method corresponds to **Figure 3.1**. The listing shown below demonstrates how the *Echo* method may be used in JScript. In opposite to VBScript you must enclose the parameters in parenthesis, and the command must be terminated with a semicolon.

```
//*********************************************
// File:    Echo.js (WSH sample in JScript)
// Author:  Günter Born
//
// Uses the Echo method in WSH.
//*********************************************
// declare variable
var price;
var vat = 16.0;
var net = 100.0;
// calculate price

price = net * (1.0 + vat/100.0);

// Show results. Because we use the WScript object, we need not to
// create the Wscript object.

WScript.Echo ("Price: ", price, "US $ Tax: ", vat,
              "% ", price - net, " US $");

WScript.Quit();

// End
```

Listing 3.2.
Echo output in JScript

Here we see again the principle how you may use a method within a script. The *Echo* statement shown above uses the *WScript* object. The dot separates object name and method. The *Echo* method uses one parameter, which contains the text to be displayed. If you like to pass several parameters, separate them with commas. Numerical values within a parameter are converted automatically from the *Echo* method into strings. The title of the dialog box and the button to close the dialog are created by the method. The statement:

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

uses the *Quit* method provided from the *WScript* object. This method terminates the script and returns the optional *Process termination code* (which is omitted here, so the method returns 0).

You find the sample file *Echo.js* within the folder *\Samples\chapter03* in the samples.

How to get a line feed within a dialog box?

I was asked many times: How can a programmer force the *Echo* method (or the other methods mentioned below) to wrap a string into several lines? If you pass a text to the *Echo* method, the run-time system does the formatting and line wrapping automatically. This causes that a long text is wrapped into multiple lines, but short texts are shown in one line.

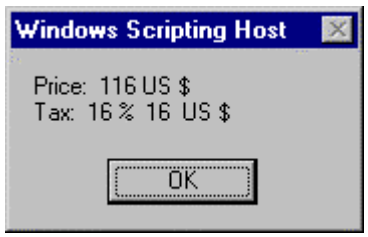

Figure 3.3.
Text using multiple lines in a Dialog box

As you can see in **Figure 3.3**, it is possible to use a line feed within the message box. Unfortunately the *Echo* method and the message box function discussed below don't provide an explicit option to force a line feed during output.

The solution is rather simple: you must insert the code for a new line into the text stream. The implementation differs between JScript and VBScript. JScript knows several escape sequences to embed special control codes into a string. A new line is forced in JScript with the escape sequence "\n". If you need to direct the output of the *Echo* method into several lines, you must insert "\n" into the parameters. The next statement uses this knowledge to divide the text into two lines.

```
WScript.Echo ("Price: ", price, "US $ \nTax: ", vat,
              "% ", price - net, " US $");
```

Alternatively you can set the escape string as a separate parameter:

```
WScript.Echo ("Price: ", price, "US $", "\n", "Tax: ", vat,
              "% ", price - net, " US $");
```

If you like to create the same result in VBScript, you must use a different approach. VBScript doesn't recognize the "\n" escape sequence. Fortunately VBScript supports the predefined named constant *vbCrLf*, which contains the requested code for a new line. The *Echo* method with a line feed may be written as:

```
WScript.Echo "Price: ", price, "US $" & vbCRLF & _
        "Tax: ", vat, "% ", price - net, " US $"
```

I have used the & character to concatenate the named constant *vbCrLf* with the other sub strings. But you may insert the *vbCrLf* also as a separate parameter like:

```
WScript.Echo "Hello", vbCRLF, "World".
```

> **NOTE:** You will find the sample files *Echo1.vbs* and *Echo1.js* in the folder *\Samples\chapter03*.

Using *MsgBox* in VBScript

The *Echo* method provides a simple way to create an output dialog within a script, but *Echo* don't let you influence the layout of your dialog box. Who already has worked with Visual Basic or VBA, knows presumably the *MsgBox* procedure (or function), which offers a comfortable way to show messages in dialog boxes. The *MsgBox* procedure uses the following syntax:

```
MsgBox prompt, buttons, title
```

The parameters requested within the call has the following meaning:

- *prompt:* Defines the text that is shown in the dialog box. This parameter is required. Inserting the VBScript constant *vbCrLf* into the output string produce a new line within the dialog text.
- *buttons:* Is optional and defines the buttons and the icon shown within the dialog box. If you omit this parameter, VBScript uses only the *OK* button within the dialog box.
- *title:* This optional parameter may contain the title text shown in the dialog's window. If the parameter is missing, Windows shows the application name within the title bar.

But which values must be used for the parameters, and what's happen, if optional parameters are omitted? We have already used the statement:

```
MsgBox "This is a test ", 0, "WSH sample - by Günter Born"
```

that creates a simple dialog box shown in **Figure 3.4**.

Figure 3.4.
A sample dialog box created with MsgBox

If you omit the third parameter with the title text, the dialog box shown in **Figure 3.5** is created. The title bar just contains the text »Visual Basic«.

Figure 3.5:
Dialog box with default title text created with MsgBox

> **NOTE:** You can omit the second and the third parameter in the MsgBox call (for instance *MsgBox "Hello world"*). If the second parameter is missing, a dialog box with an *OK* button and the text passed in the first parameter is created. If the third parameter is set, but the second should be omitted, you must set the comma as a placeholder (for instance *MsgBox*

"Hello world", ,"WSH Echo sample"). The folder *\Samples\chapter03* contains the script *MsgBox.vbs*, which demonstrates the use of *MsgBox*.

Define the buttons and the icon

The second parameter of the *MsgBox* call may be used to control the number of buttons, their captions and the icon shown in the dialog. The values for this parameter are predefined in Windows, and VBScript provides named constants for their values. **Table 3.1** contains an overview of the *MsgBox* constants already defined for the icons shown in a dialog box.

Constant	Icon	Remark
0	–	No icon (default)
16 vbCritical		Stop icon
32 vbQuestion		Question mark
48 vbExclamation		Exclamation mark
64 vbInformation		Information

Table 3.1.
MsgBox constants for icons within the MsgBox *dialog box*

> **NOTE:** The first column in the table contains the constants 0, 16 and so on, and the named constants like *vbOkOnly*. You may use a numeric value like 32 for the question mark within the *MsgBox* call. But the named constant *vbQuestion* is much more readable. JScript doesn't support the *MsgBox* procedure.

The parameter *buttons* may contain also additional constants to define the buttons shown in the dialog box. **Table 3.2** contains the predefined constants used for this purpose. If you like to show an icon and a button combination within the dialog box, you must simply add the constants given in **Table 3.1** and **Table 3.2**.

Constant	Remark
0 vbOkOnly	Show *OK* button (default)
1 vbOKCancel	Show buttons *OK* and *Cancel*
2 vbAbortRetryIgnore	Show buttons *Abort*, *Retry* and *Ignore*
3 vbYesNoCancel	Show buttons *Yes*, *No* and *Cancel*
4 vbYesNo	Show buttons *Yes* and *No*
5 vbRetryCancel	Show buttons *Retry* and *Cancel*.

Table 3.2.
MsgBox constants for buttons

> **NOTE:** I should mention here that the caption for the buttons depend on the localized Windows version. The table given here contains the caption for the US Windows version. The folder *\Samples\chapter03* contains the sample file *MsgBox1.vbs*, which shows a sequence of dialog boxes with all available button combinations.

A statement to show a dialog box containing the *OK* button and the question mark must be written as:

```
MsgBox "Question", _
       vbOKOnly + vbQuestion, _
       "MsgBox demo "
```

This statement demonstrates how useful named VBA constants are. Using *vbOkOnly + vbQuestion* clearly indicates what the script developer intend to do, whilst the constant *32* is a bit cryptic.

One button in the dialog box receives the focus to close the dialog box. For default, always the first button (counted from the left) gets the focus. But you can set the focus to a predefined button within the dialog box. **Table 3.3** contains the constants, which may be used to select the button getting the focus. You must add the constant to the values for the icon and the button.

Constant	Remark
0 vbDefaultButton1	Set the focus to the first button (this is the default setting)
256 vbDefaultButton2	Set the focus to the second button
512 vbDefaultButton3	Set the focus to the third button

Table 3.3.
MsgBox constants to select the default button within the dialog box

Which button was clicked in a dialog box?

Sometimes the programmer need to know which button is clicked to close a dialog box. In **Figure 3.6** the dialog box contains two buttons, so the user may choose *OK* or *Cancel* to close the dialog box.

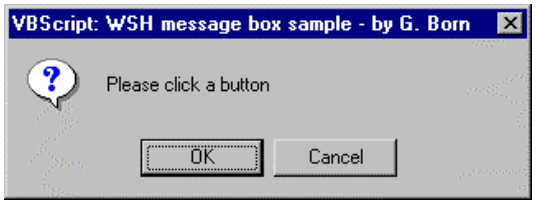

Figure 3.6:
Dialog with two buttons

In VBScript you can use *MsgBox* as a function to retrieve a code that indicates the button clicked to close the dialog box. The value returned from the function call indicates which button has been clicked. The sample shown below checks which button is used to close the second dialog box. The result with the button's name is shown in a third dialog box.

```
'*********************************************
```

```
' File:    MsgBox2.vbs (WSH sample in VBScript)
' Author:  G. Born
'
' Use MsgBox to show a message.
'*********************************************
Option Explicit

Dim WSHShell
Dim Text1, Text2, Text3, Text4
Dim Title, result

' define texts

Title = "WSH message box sample - by G. Born"
Text1 = "Just a note: Windows Scripting Host"
Text2 = "OK button clicked"
Text3 = "Cancel button clicked"
Text4 = "Please click a button"

' Call the MsgBox-function
' MsgBox prompt, buttons, title
' prompt:    the text shown in the dialog
' title:     the dialog box title
' buttons:   the buttons

MsgBox Text1, + vbInformation + vbOKOnly, Title

' Try to check which button was clicked
' result = MsgBox (prompt, buttons, title)

 result = MsgBox (Text4, vbQuestion + vbOKCancel, Title)

' try to inspect the return value
If result = vbOK Then
 WScript.Echo Text2    ' show result with Echo method
Else
 WScript.Echo Text3
End If


' End
```

Listing 3.3.
Sample MsgBox2.vbs

The variables *Text1*, *Text2* etc. defines the messages shown in the dialog boxes. During execution the script shows several dialogs. The first dialog box has only one *OK* button, the next dialog box contains two buttons. This dialog is created with the *buttons* value set as:

```
result = MsgBox (Text4, vbQuestion + vbOKCancel, Title)
```

MsgBox is used here as a function, because the name is on the right side of an assignment statement. Therefore the *MsgBox* parameters must be set into parenthesis! The *MsgBox* function returns a value indicating the button used to close the dialog box. **Table 3.4** lists possible values returned from the *MsgBox* function. Which values returned depends on the buttons shown in the dialog box.

Value	Constant	Remark
1	vbOK	*OK* button clicked
2	vbCancel	*Cancel* button clicked
3	vbAbort	*Abort* button clicked
4	vbRetry	*Retry* button clicked
5	vbIgnore	*Ignore* button clicked
6	vbYes	*Yes* button clicked
7	vbNo	*No* button clicked

Table 3.4.
MsgBox return codes

You may use a number between 1 and 7 or a symbolic constant to check which button has been clicked. This enables you to implement a structure within your script that allows the user to choose between two options. In **Listing 3.3** I have used the following statements to check the returned code:

```
' Check which button was clicked
If result = vbOK Then
 WScript.Echo Text2    ' use Echo for output
Else
 WScript.Echo Text3
End If
```

If the user clicks the *OK* or *Cancel* button, the program tests the return value. The result is shown in a second dialog box. I have used the *Echo* method to show the result within a dialog:

```
WScript.Echo Text2
```

In the statement above *WScript.Echo* shows the content of the variable *Text2* within the dialog box.

NOTE: You find the VBScript sample *MsgBox2.vbs* in the folder *\Samples\chapter05*.

Using the Popup method

The *Echo* method introduced above shows only simple dialog boxes. To gain more control over the dialog, I have used the *MsgBox* procedure or function. With the *MsgBox* function we can check the returned value to detect which button was clicked to close a dialog box. Unfortunately the *MsgBox* function isn't supported in JScript. The following section introduces the *Popup* method which can be used in VBScript (to substitute the *MsgBox* function) and in JScript. In JScript *Popup* provides the only possibility to control the number of buttons and icons within a dialog box and to check which button was used to close the dialog.

Using Popup in JScript?

I have mentioned it in the previous section: JScript doesn't support a *MsgBox* function. So JScript depends in whole on the object model exposed by the underlying host. The Microsoft Internet Explorer provides a rich set of output features like *confirm()* or the *write* method (*document.write*). All these features are not supported in the object model exposed from the WSH. So functions supported in the Internet Explorer may not work for JScript in a WSH script. To create customized dialogs similar to the *MsgBox* function we must use the *Popup* method provided by the *Shell* object. The *Shell* object and the *Popup* method are not exposed automatically from the WSH object model. Therefore you must create first your object instance before you may access the *Popup* method. Till now I haven't discussed how to create a reference to an object.

To instantiate an object, there are several ways. For WSH scripts you may use the *CreateObject* method of the *WScript* object. *CreateObject* uses the ProgID of the requested object as a parameter. The *Shell object exposes the Popup method*, which is itself a sub object of the *WScript* (application) object. The following statement shows how a reference to the object may be created in JScript:

```
var WSHShell = WScript.CreateObject("WScript.Shell");
```

A simple variable declaration creates the object variable *WSHShell*. This variable keeps a reference to the object, which is returned from *WScript.CreateObject*. *WScript* is an object, which supports the *CreateObject* method. The *CreateObject* method requires the ProgID of a new object as a parameter. For the *Shell* object this ProgID is defined as *WScript.Shell*. The statement shown above creates a new object instance and stores a reference to this object into the object variable. You can use this object variable to access the methods and properties provided by this object. The next statements show the steps required to use the *Popup* method:

```
var WSHShell = WScript.CreateObject("WScript.Shell");
var intDoIt;
intDoIt = WSHShell.Popup(Message,
                         0,
                         Title,
                         vbOKCancel + vbInformation );
```

After creating the object variable *WSHShell*, you can use *WSHShell.Popup* to display the dialog. The *WSHShell.Popup* call uses the following syntax:

```
res = WSHShell.Popup(prompt, wait, title, buttons);
```

The parameters of the *Popup* method has the following meaning:

- *prompt*: Contains the text that should be displayed within the dialog. This is similar to *prompt* used with *MsgBox*. You may concatenate constants, sub strings and variables.
- *Wait*: Defines a time-out value for the dialog box. If the user doesn't click a button within the time-out interval, the script closes the dialog automatically. The value 0 disables the time-out.
- *Title*: Defines the dialog's title text.
- *Buttons*: Defines the icon and the buttons shown in the dialog box. The values used for this parameter are the same as for the VBScript *MsgBox* function (see **Table 3.1** till **Table 3.4**). But note that JScript doesn't know predefined symbolic constants (like *vbOKOnly*). So you need to combine the numbers given in the tables to define the icon and the buttons.

After closing the dialog box, *Popup* returns the code of the selected button.

NOTE: As mentioned above, *Popup* may be used in JScript and in VBScript. Although JScript doesn't support named constants like *vbOkOnly*, you can define such constants within your script.

```
var vbOKCancel = 1;
var vbOK = 1;
var vbInformation = 64;
var vbCancel = 2;
```

Then you may use these constants within your JScript script. If you port a VBScript script to JScript, keep in mind that the *Popup* method uses similar parameters and produces the same result as the *MsgBox* call mentioned above. But the number of parameters and their order differs between both calls.

The next listing shows the source code of a JScript sample using the *Popup* method.

```
//*********************************************
// File:    Popup.js (WSH sample in JScript)
// Author:  Günter Born
//
// Using the Popup method to show a dialog.
//*********************************************
// declare variable

var vbOKCancel = 1;
var vbOK = 1;
var vbInformation = 64;
var vbCancel = 2;
var result;

var Message = "Click a button";
var Title = "WSH Popup sample";

// create the WSHShell object variable
var WSHShell = WScript.CreateObject("WScript.Shell");

result = WSHShell.Popup(
         Message,
         0,
         Title,
         vbOKCancel + vbInformation ); // show dialog

if (result == vbOK) // check and show results
{
 WScript.Echo ("OK button clicked " +
           "(Code: " + result + ")");
}
else
 {
  if (result == vbCancel)
   {
   WScript.Echo ("Cancel button clicked " +
```

```
            "(Code: " + result + ")");
   }
 }

WScript.Echo("We are ready");


// End
```

Listing 3.4.
Sample Popup.js

The statement:

```
result = WSHShell.Popup(
         Message,
         0,
         Title,
         vbOKCancel + vbInformation ); // show dialog
```

uses the *Popup* method to create the user dialog. I have used the »pseudo constants« *vbOKCancel* and *vbInformation* to set the *buttons* value. After closing this dialog, the *Popup* method returns the code for the clicked button. We may use the statement:

```
if (result == vbOK)
```

to test the value returned from *Popup*. Depending on the value stored in *result* a message box is shown using the *Echo*.

```
WScript.Echo ("OK button clicked " + "(Code: " + result + ")");
```

Details may be obtained from the listing above.

NOTE: The JScript sample *Popup.js* may be found in the folder *\Samples\chapter03*.

Can we use Popup also in VBScript?

I believe most VB programmers will use the *MsgBox* call within their VBScript code. But you may use also the *Popup* method exposed by the *Shell* object to handle your user dialogs. The following listing demonstrates, how this method may be used in VBScript. I kept the sample consciously simply, to show solely the employment of the method.

```
'*********************************************
' File:    Popup.vbs (WSH sample in VBScript)
' Author:  G. Born
'
' Creates a message box using the Popup method.
'*********************************************
Option Explicit

Dim result
Dim WSHShell

' we need the Shell object !!!
```

```
Set WSHShell = WScript.CreateObject("WScript.Shell")

' use the Popup method

result =  WSHShell.Popup("Click an button", _
                          0, _
                         "WSH Popup sample", _
                         vbOKCancel + vbInformation)

WScript.Echo "Return value ", result


' End
```

Listing 3.5.
Sample Popup.vbs

NOTE: The VBScript sample file *Popup.vbs* is located in the folder *\Samples\chapter03*.