

5 User Input in scripts

In the chapter 3 have introduced a few techniques to create simple user dialogs. Now I like to discuss how to implement simple dialogs allowing a user input within WSH scripts.

User input in VBScript

VBScript supports the *InputBox* function to retrieve user input. The function uses the following syntax:

```
result = InputBox (prompt[, [title] , [default] , [xpos] , [ypos]])
```

The function call opens a dialog box window with a user-defined message, a title text and a simple input box (**Figure 5.1**). The user may enter something into the textbox and he/she may click onto the buttons.

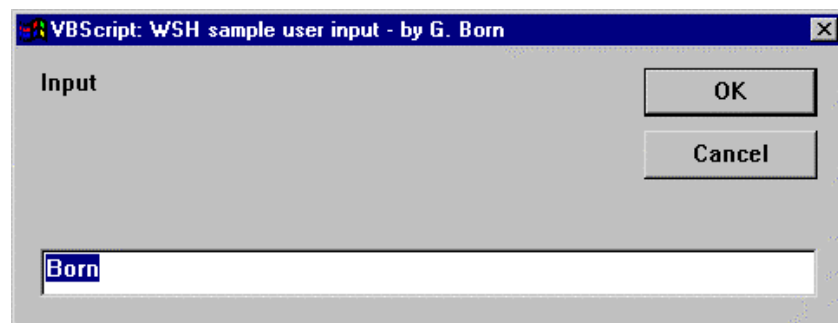


Figure 5.1.
User dialog using the InputBox function

The *InputBox* function uses the following :

- ◆ *prompt*: This parameter is mandatory. It defines the message, which is shown in the dialog box window. In **Figure 5.1** this is the text »Input«.
- ◆ *title*: Defines the optional title text for the dialog box window.
- ◆ *default*: Specifies the default value, which shall be shown within the text box after invoking the input dialog. This parameter is optional.
- ◆ *xpos* and *ypos*: Define the position of the upper left corner of the dialog box window. This parameter is optional.

If one of the optional parameters is missing, VBScript uses default values within the dialog box window. To omit an optional parameter between other parameter, you must set a comma (like `InputBox ("Hello", "Test", ,100,200)`). The function doesn't supports a *buttons* parameter (like *MsgBox*).

As soon as the user closes the input dialog, the *InputBox* function returns a value. This value depends on the button that was used to close the dialog. The *OK* button returns the content of the input box (the user input). If the user aborts the dialog choosing the *Cancel* button (see **Figure 5.1**), an empty string "" is returned. This can be used to check whether the user input is valid or not. The following code sequence checks the return value *result* and uses an *if* statement to show the result.

```
If result = "" Then      ' Test for Cancel
    WScript.Echo "Canceled"
Else
    WScript.Echo "You entered: " & result
End If
```

The VBScript program shown below uses this technique. The script displays first an input box dialog window containing a text box with the default value (**Figure 5.1**). A user can change this input value. After closing the dialog box, the script displays either the result or a cancelled message within a second dialog box window (**Figure 5.2**).



Figure 5.2.

Display the user input retrieved from InputBox

The text shown in the dialog box is declared within the script's header in global variables. This keeps the function call as simply as possible. I have used also the VBScript constant *vbCrLf* to split the output string into two lines. Details about using the *InputBox* function are shown in **Listing 5.1**.

```
' *****
' File:      Input.vbs (WSH sample in VBScript)
' Author:    Günter Born
'
' The script implements a user input.
' *****
Option Explicit

Dim Message, result
Dim Title, Text1, Text2

' define dialog box variables
Message = "Input"
Title = "WSH sample user input - by G. Born"
Text1 = "User input cancelled"
Text2 = "You entered:" + vbCrLf

' Ready to use the InputBox function
```

```
' InputBox (prompt, title, default, xpos, ypos)
' prompt:    The text shown in the dialog window
' title:     the title of the dialog window
' default:   Default value shown in the input box
' xpos/xpos: position upper left of dialog box window
' If a parameter is omitted, VBScript used defaults

result = InputBox(Message,Title,"Born", 100, 100)

' Evaluate the user input
If result = "" Then      ' Canceled by the user
    WScript.Echo Text1
Else
    WScript.Echo Text2 + result
End If

WScript.Quit()          ' Ready
' End
```

Listing 5.1.

Input.vbs

NOTE: The VBScript sample *Input.vbs* is located in the folder `\Samples\chapter05`.

User input in JScript

Creating a user input dialog box in JScript is a huge problem. The language doesn't support the VBScript *Inputbox* function already introduced in the previous section. Also a command like:

```
var txt = window.prompt("WSH sample ", "Name: ");
```

which may be used within a HTML script, won't work in WSH scripts. And the WSH doesn't provide a method for user input. To overcome this situation, I have developed a few solutions.

JScript-user input using a second script

VBScript supports the *InputBox* function, it would be an idea to call this function from a JScript script. Unfortunately you can't append a VBScript procedure to a JScript script (in WSH 1.0). Within one script file, you must use one language (because WSH 1.0 uses the file name extension to select the language engine).

The only solution is: create two scripts, the JScript script and a (helper) VBScript script. The VBScript script handles the user input. The *Run* method introduced in the previous chapter allows you to launch the VBScript script from a JScript script. But we have to solve a problem: How to transfer the input value from the child script to the parent script? Using the code returned from the *Quit* method restricts you to numerical values (0, 1, 2 and so on). This can't be used to return a string.

NOTE: We could use a file to exchange a string, or we could use Registry entries to exchange data. The best solution would be to use an environment variable to return the

value. As you have seen in the previous chapter, environment variables are local to a process. So it is not possible to share a variable between two processes. And environment variables are not permanent, if they are created within a script. If the script terminates, the environment variable is getting lost – so exchanging a value between two scripts using an environment variable might be failed. Surprisingly there is an exception: If a script calls a second script using the *Run* method, this child script may create a new environment variable. And this variable may be read from the parent script. So you can create an environment variable in the child process, which remains valid if this script, terminates. This environment variable may be read within the parent script. I was not able to access an environment variable created from the parent script within the child script. But this isn't a major problem, because you can submit parameters from the parent to the child using the *Run* method.

I have used this trick to implement a simple user input dialog within JScript. The following code sequence allows to read the environment variable within the parent script.

```
WSHShell = WScript.CreateObject("WScript.Shell");
var wshEnv = WSHShell.Environment("Volatile"); // get the environment variable
// Get user input from environment variable "Result"
var test = wshEnv("Result");
```

In the second line the *Environment* collection is assigned to the *wshEnv* variable (we use the *Volatile* section of the environment). Then the variable's value is read back, using the object *wshEnv*. To execute the child script with the user input dialog, the following commands may be used:

```
// get the path to the script file VBInput.vbs
var path = WScript.ScriptFullName;
path = path.substr(0,path.lastIndexOf("\\")+1);
path = path + "\\VBInput.vbs";
// execute VBScript program
WSHShell.Run ("WScript.exe "+ path,1, true);
```

The last line calls the child script. The first parameter may be used also to submit parameters from the parent script to the child. Because the third parameter of the *Run* method is set to *true*, the parent script waits till the child process terminates. Then the code snippet shown in the previous section is used to fetch the user input from the environment variable. The following Listing contains the implementation of the JScript parent script.

```
//*****
// File:      Input0.js  (WSH sample in JScript)
// Author:    (c) G. Born
//
// Demonstrates a trick to get user input in JScript.
// We call a VBScript program to get the user input.
// The input is stored into an environment variable.
// The variable created in the VBScript child remains
// valid during the parent process life time, if the
// child is created with the Run method!!!
//*****

var WSHShell;

// create Shell object
WSHShell = WScript.CreateObject("WScript.Shell");
```

```
// object to access environment (collection object)
var wshEnv = WSHShell.Environment("Volatile");

// get the path to the script file VBInput.vbs
var path = WScript.ScriptFullName;
path = path.substr(0,path.lastIndexOf("\\")+1);
path = path + "\\VBInput.vbs";

// execute VBScript program
WSHShell.Run ("WScript.exe "+ path,1, true);

// Get user input from environment variable "Result"
var test = wshEnv("Result");
WScript.Echo ("You entered: ", test);

WScript.Quit();           // close WScript object

// End
```

Listing 5.2.
Input0.js

Now we still need the child script that contains the implementation of the input box dialog and which stores the user input into the environment. This may be done with VBScript using the following statements.

```
' *****
' File:      VBInput.vbs (WSH sample in VBScript)
' Author:    (c) G. Born
'
' Shows an input dialog and stores the user input
' into the environment variable Result.
' *****
Option Explicit

Dim Message, result, WshEnv, WSHShell
Dim Title, Text1, Text2

' Objects to access the environment variables
Set WshShell = Wscript.CreateObject("WScript.Shell")
Set WshEnv = WshShell.Environment ("Volatile")

' set dialog texts
Message = "Input"
Title = "WSH by Günter Born"

' Using the InputBox-Function
result = InputBox(Message,Title,"")
```

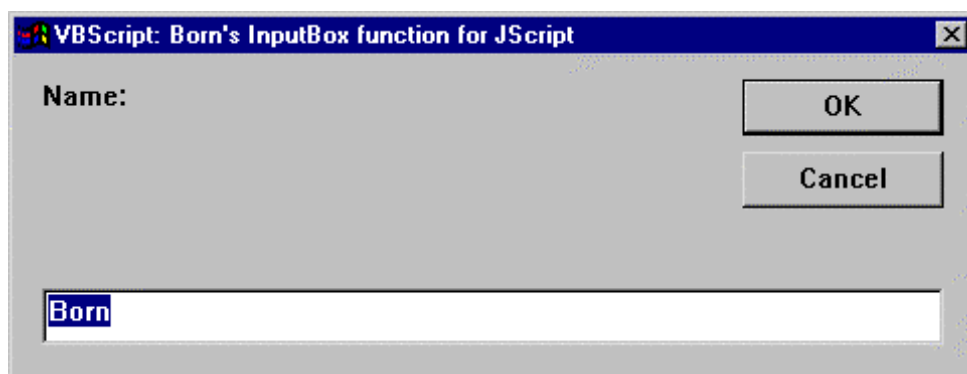
```
' Store user input into environment
wshEnv ("Result") = result
WScript.Quit()      ' ready
' End
```

Listing 5.3.*VBInput.vbs*

Executing the parent script causes that an input box dialog is shown from the child script. If the user dialog is closed, the result obtained from the child process is displayed within a second dialog box window. You may find the parent script *Input0.js* and the child script *VBInput.vbs* within the folder *\Samples\chapter05*.

Internet Explorer for JScript user input

The trick I have shown in the preceding section is based on the attempt that the behavior of the environment remains in further WSH versions. Therefore I have implemented a second solution, which enables you to invoke an input dialog from JScript. I use also VBScript to create the Input box dialog (as you can see in the dialog's title bar shown in **Figure 5.3**).

**Figure 5.3.***Dialog for user input in JScript*

The question is: How can we access a VBScript script without calling it using the *Run* method shown in the previous section. The idea is: Insert the VBScript script into a document, which can be loaded by an application, which supports VBScript. Besides Microsoft Office application the Microsoft Internet Explorer is also such a program – and all machines supporting WSH must have this browser installed. If the script is already loaded within an application, I can use the application's object model to access also the functions of the script. How this works will be discussed in detail below.

NOTE: The technique I describe below is a bit complicated. And I will show later a much more simple solution using an ActiveX control. Although I decided to let the sample in this chapter, because the technique may be the base for using the Internet Explorer for form input (I will discuss this in further chapters).

Implementing an *InputBox* function in HTML

A HTML document may contain scripts, which can be embedded using the <SCRIPT>-Tag. Therefore we can store a script written in VBScript within a HTML document. The HTML source code for such a HTML document is shown below:

```
<HTML>
<HEAD>
<Script LANGUAGE="VBScript">
<!--
Function InputBox1 (prompt,title, value)
    InputBox1 = InputBox (prompt, title, value)
End Function
//-->
</Script>
</HEAD>
<BODY></BODY>
</HTML>
```

Listing 5.4.

HTML source code of an document containing a VBScript script

The script just contains the *InputBox1* function, which calls itself the *InputBox* function provided in the VBScript syntax. The *InputBox1* call just passes its parameters to the *InputBox* function. Also the returned value from *InputBox* is passed to the calling module. So we may see the *InputBox1* function as a shell for the *InputBox* call.

NOTE: If you load the HTML file within a browser, just an empty document will be shown. This is intended, because we need only the *InputBox1*, to show the input dialog.

Call the function within the HTML document

Let's assume you have deposited the function discussed in the previous section into a HTML document. How can we access this function from a WSH script either written in JScript or in VBScript? At this point I can say already, that there are no differences between JScript and VBScript. Solely the differences in the syntax must be considered.

To use the Internet Explorer and its objects from a WSH script, you must obtain a reference to the Internet Explorer *Application* object. You may use the following statement in JScript to create the object:

```
var oIE = WScript.CreateObject("InternetExplorer.Application");
```

The object variable *oIE* may be used afterward to access the properties and methods supported by the *Application* object of the Internet Explorer. In the next step we must advice the Internet Explorer to load the requested document containing the HTML code with the script. If this file is located in *C:\input.htm* you may use the *navigate* property as shown below:

```
oIE.navigate ("c:\input.htm");
```

Set the *visible* property to 0 disables the browser window (we just need the dialog box). But this property is set by default to 0, so you can omit this statement.

Did you launched the Internet Explorer, loaded the HTML document containing the script and link the object to the WSH? Then you have to call the *InputBox1* function located in the Internet Explorer's document object from your WSH script. If we solve this question, we could use the Internet Explorer as an Extender for JScript.

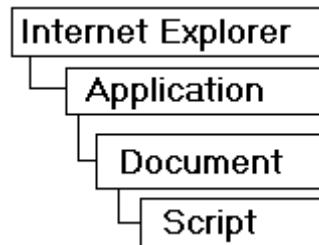


Figure 5.4.

Internet Explorer 4.01 object model

The Internet Explorer's object model shown in **Figure 5.4** allow us the navigation within the object hierarchy. As far as you specify the Type-Library »Internet Explorer« with the class »Application« within the *CreateObject* method, this method returns the Internet Explorer *Application* object. This object may contain the document loaded in the browser. And a *Document* object may contain beside the HTML tags also a script. The script will be exposed from the *Document* object that is a container for the *Script* object. To access a script function embedded in the HTML document, you must use the following statement:

```
var oIEscr = oIE.Document.Script;
```

The object variable *oIEscr* is created from the object hierarchy, because *oIE* points to the Internet Explorer *Application* object. Using this new object variable allows a direct access to all script functions embedded within the HTML document. The statement shown below accesses the *InputBox1* function in the document loaded in the Internet Explorer.

```
var result = oIEscr.InputBox1(prompt,title,x);
```

From the view of a script programmer this looks like using a method. And that's the point: *InputBox1* is nothing else then a method of the *Script* object.

The *InputBox* implementation for JScript

I would like to discuss now how the dialog box shown in **Figure 5.3** is implemented in a JScript program. And I like to introduce a few new techniques, which allow you to feed the statements for the script directly from the WSH script into the Internet Explorer's document window. Using this technique you need not to load an HTML document file into the browser. To keep the handling as simple as possible, I have implemented a function within the JScript script that may be called using the code:

```
var mobj = makeInputBox ();// launch IE and create the InputBox1 method
var result = InputBox (mobj, prompt,title,"Born");// get user input
```

The function *makeInputBox()* handles the task to load the HTML document with the helper script into the Internet Explorer. This function returns an object reference to the *script* object, which exposes the properties and methods. Using this object variable allows you to call the *InputBox* method from Jscript as shown above. The parameter *mobj* must contain the object variable re-

turned from *makeInputBox*. *InputBox* is implemented as a function in JScript using the following structure:

```
function InputBox (obj, prompt, title, x)
{
// Interface to the InputBox1 function in the Document.Script object
var oIEdoc = obj.Document.Script;
var result = oIEdoc.InputBox1(prompt,title,x);
return result;
}
```

Listing 5.5.

InputBox function

The parameter *obj* contains the object reference (here the reference to the Internet Explorer's *Application* object returned from *makeInputBox*). Now we need an object reference, to call the *InputBox1* method within the HTML document from JScript for opening the input dialog. This reference may be obtained (according to the Internet Explorer's object model) within JScript using the following statement:

```
var oIEdoc = obj.Document.Script;
```

Then the following simple statement may be used to call the *InputBox1* function within the HTML document:

```
var result = oIEdoc.InputBox1(prompt,title,x);
```

To keep handling as simple as possible, I have moved all the code into a JScript function *InputBox* of the WSH script. Therefore you may interpret all functions as black boxes, which are handling all necessary things in the background. The script programmer just has to call the two functions *makeInputBox()* and *InputBox()*. That's all!

Let's come back to the content of the *makeInputBox()* function. We need a solution to launch the Internet Explorer and to load a document including the script. Feeding the script's code into the Internet Explorer document may be done with the following code, encapsulated in the *makeInputBox* function.

```
function makeInputBox ()
{
// create Internet Browser object
var oIE = WScript.CreateObject("InternetExplorer.Application");
oIE.navigate ("about:blank"); // load empty document
oIE.visible = 0;             // keep MSIE invisible

while (oIE.Busy) {}          // Important: idle loop, wait till MSIE is ready

var doc1 = oIE.Document;     // fetch the current (empty) document object
doc1.open;                   // open document
                             // write script code into document
doc1.writeln("<HTML><HEAD>");
doc1.writeln("<Script LANGUAGE=\"VBScript\"><!-->");
doc1.writeln("Function InputBox1 (prompt,title, value)");
doc1.writeln(" InputBox1 = InputBox (prompt, title, value)");
doc1.writeln("End Function");
doc1.writeln("/>-->");
```

```
doc1.writeln ("</Script>");
doc1.writeln ("</HEAD><BODY></BODY></HTML>");
doc1.close;           // close document for write access

return oIE;
}
```

After executing the function, the Microsoft Internet Explorer is launched, a new document window is opened and this document contains the script code. It looks a little bit complicated, isn't it? Here are a few additional explanations. The first statement creates the Internet Explorer application object:

```
var oIE = WScript.CreateObject("InternetExplorer.Application");
```

If the browser is installed and registered (which is *true* under Windows 98), the *CreateObject* method returns the object reference into the object variable *oIE*. With a few statements you can set the properties of the new object (here the Internet Explorer window, which presents the application). Most of the properties provided by the browser doesn't interest in this example (since the Browser window is hidden). The statement:

```
oIE.visible = 0;
```

assures that the Browser window keeps hidden. This isn't required at all, because this property is set by default to 0. Important is the line to load the document:

```
oIE.navigate ("about:blank");
```

In normal cases the *navigate* property keeps the path to a HTML document file or URL address. This file will be loaded and displayed in the document window. But: The document must be loaded using an absolute URL/path, the path must be present, and it must contain the requested content. Therefore I decided to load an empty document and add the requested content. The statement above creates a blank document. Adding the content may be done with the following sequence:

```
var doc1 = oIE.Document;      // get Document object
doc1.open;                    // open document
                               // write script into document
doc1.writeln ("<HTML><HEAD>");
doc1.writeln ("<Script LANGUAGE=\"VBScript\"><!-->");
doc1.writeln ("Function InputBox1 (prompt,title, value)");
doc1.writeln (" InputBox1 = InputBox (prompt, title, value)");
doc1.writeln ("End Function");
doc1.writeln ("/-->");
doc1.writeln ("</Script>");
doc1.writeln ("</HEAD><BODY></BODY></HTML>");
doc1.close;                   // close document for write access
```

The first statement gets the reference to the *document* object. The next line opens the document for write access. Subsequent statements use the *writeln* method to write into the document. The *writeln* sequence creates a HTML structure that contains a *<SCRIPT>* tag to embed VBScript script. The script contains a function *InputBox1*, which calls itself the VBScript function *InputBox*. After feeding the HTML tags containing the script statements into the document, the WSH closes the (virtual) document for write access and return the reference to the object variable (the document).

NOTE: I have used above the *<Script>* tag to enclose the code. This is not necessary, if you trust that the browser understand also the pure script code.

If you have a reference to the object, you may access the *InputBox1* function from the WSH script. The Internet Explorer executes the script within the document, and an input dialog box is shown (Figure 5.3).

IMPORTANT: If you need the *InputBox1* method no more, you must terminate the Internet Explorer (otherwise the application remains in the memory). You may apply the *Quit* method on the Internet Explorers *application* object (this object supports this method).

Now we have all tools to use an input box function within any WSH script. All we need are two function calls. The following listing shows the whole program structure.

```
//*****
// File:      Input1.js (WSH sample in JScript)
// Author:    Günter Born
// Check out the WSH Bazaar at:
// ourworld.compuserve.com/homepages/Guenter_Born
//
// Demonstrates how to realize a user input in JScript.
// The script uses the Internet Explorer !!!
//
//*****

var title = "Born's InputBox function for JScript"
var prompt = "Name:"
var WSHShell;
var vbOKCancel = 1;      // auxillary variable
var vbOKOnly = 0
var vbInformation = 64;
var vbCancel = 2;

// helper function -> creates an InputBox function
// in the Internet Explorer
function makeInputBox ()
{
// create Internet Browser application object
var oIE4 = WScript.CreateObject("InternetExplorer.Application");
oIE4.navigate ("about:blank"); // empty HTML document
oIE4.visible = 0;             // keep MSIE invisible

while (oIE4.Busy) {}          // Important: Wait till MSIE ready

var doc1 = oIE4.Document;     // get document object
doc1.open;                     // open document
                               // write script into the document object
doc1.writeln("<HTML><HEAD>");
doc1.writeln("<Script LANGUAGE=\"VBScript\"><!-->");
doc1.writeln("Function InputBox1 (prompt,title, value)");
doc1.writeln(" InputBox1 = InputBox (prompt, title, value)");
doc1.writeln("End Function");
doc1.writeln("//-->");
doc1.writeln("</Script>");
```

```
doc1.writeln("</HEAD><BODY></BODY></HTML>");
doc1.close; // close document for write access

return oIE4;
}

function InputBox (obj, prompt, title, x)
{
// Interface for the InputBox1 function in the document object
// of the MS IE 4/5
var oIE4doc = obj.Document.Script;
var result = oIE4doc.InputBox1(prompt,title,x);
return result;
}

//+++++
// main module
//+++++
{
// Create our Shell object (is needed to use Popup).
WSHShell = WScript.CreateObject("WScript.Shell");

// launch IE 4.0/5.0, create InputBox method
var mobj = makeInputBox ();

// Get user input using the InputBox method

var result = InputBox (mobj, prompt,title,"Born");
WSHShell.Popup("You entered: " + result,
               0,
               "Result",
               vbOKOnly + vbInformation );

               // 2nd test
result = InputBox (mobj, prompt,title,"James Brown");
WSHShell.Popup("You entered: " + result,
               0,
               "Result",
               vbOKOnly + vbInformation );

mobj.Quit(); // close Explorer object

WScript.Quit(); // Ready, close WScript object
}
// End
```

Listing 5.6.

Input1.js

NOTE: The JScript sample *Input1.js* is located in the folder `\Samples\chapter05`. To test this sample, just double-click on the script file. If the Internet Explorer 4.0/5.0 is present, the input dialog is shown.

An *InputBox* ActiveX control

I discussed the samples given below to allow you a few insights into script programming and using methods provided from the Internet Explorer (and scripts loaded in MS IE documents). If you need to implement a user input in JScript, a more simple solution is recommended. Although the WSH doesn't support an object, which provides an *InputBox* method, we can extend the WSH with such an object. All we need is an ActiveX control that provides the object and the method.

NOTE: In chapter 10 I will discuss how to write your own ActiveX control which exposes the *WSHInputBox* method.

Because I use the *WSHInputBox* method in upcoming samples, I like to show you here how to access this method from JScript. To use the *WSHInputBox* method, you need to create a reference to the object. This may be done with the statement:

```
var objExt = WScript.CreateObject("WSHExtend.WinExt");
```

The *CreateObject* method of the *WScript* object is advised to search the entry *WSHExtend.WinExt* within the Registry. *WSHExtend* is the name of a Type-Library (ActiveX control), whilst *WinExt* defines a class name (the object). If the control is registered, *CreateObject* creates an object instance and returns a reference to this object into the object variable *objExt*. This variable may be used later on to access the methods of this object.

NOTE: You need to install and register the *WSHExtend* ActiveX control before you can use the *WSHInputBox* method. The OCX-file may be found in the folder `\Samples\chapter10\ActiveX\ExtAPI`. Information about installing/uninstalling OCX-files may be found in chapter 2.

If you have an object variable with a reference to the *WinExt* object, you may access the *WSHInputBox* method using the following syntax:

```
result = objExt.WSHInputBox(prompt, title, default)
```

The syntax is equivalent to a VBScript *InputBox* call, and it opens an input box dialog. The parameters are defined as follows:

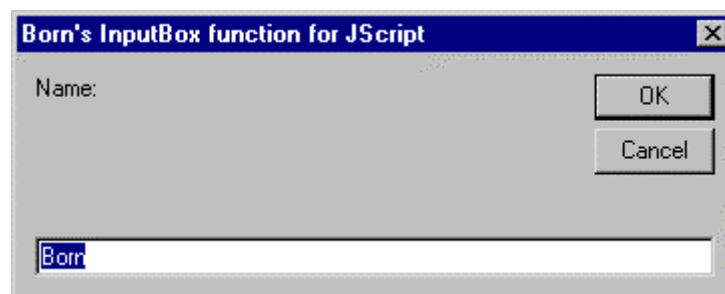


Figure 5.5:
Input dialog using the *WSHExtend*-objects

- ◆ *prompt*: This parameter is mandatory and defines the text shown in the dialog box.
- ◆ *title*: Defines the title text of the dialog box window.
- ◆ *default*: Contains the initial value shown in the input box.

As soon as the method is called, an input dialog is shown (**Figure 5.5**). **Listing 5.7** contains the whole JScript program to implement a user input using the *WSHExtend* ActiveX control.

```
//*****
// File:      Input2.js (WSH sample in JScript)
// Author:    Günter Born
//
// Demonstrates how to implement a user input in JScript.
// The script uses the ActiveX control WSHExtend.ocx !!!
//*****
var vbOKOnly = 0;
var vbInformation = 64;
var vbCancel = 2;
var title = "Born's InputBox function for JScript";
var prompt = "Name:";

// Create the Shell object (needed to use Popup).
var WSHShell = WScript.CreateObject("WScript.Shell");

// Get the WSHExtend.WinExt object. This extends the WSH with
// the InputBox1 method. WSHExtend is an ActiveX control.
var objAdr = WScript.CreateObject("WSHExtend.WinExt");

// open Input dialog
var result = objAdr.WSHInputBox (prompt,title,"Born");

if (result != "") // Test, whether Cancel clicked
{ // No, get input
    var intDoIt = WSHShell.Popup("You entered: " + result,
                                0,
                                "Result",
                                vbOKOnly + vbInformation );
}
else
{ // Cancel button was clicked
    var intDoIt = WSHShell.Popup("Sorry, no input",
                                0,
                                "Result",
                                vbOKOnly + vbInformation );
}

// here I have omitted WScript.Quit() to demonstrate
// that it also works, because the WSH automatically executes
// Quit after reaching the last statement.
// End
```

Listing 5.7.
Input2.js

NOTE: The JScript file *Input2.js* is located in the folder *\Samples\chapter05*.