Visual Basic Script (VBSCRIPT)—You Can Script Anything!

By John Papproth

AUTOMATING THE DESKTOP

Visual Basic Scripting is easy to learn and use! It can be used to automate tasks ranging from Windows Desktop Administration, to Microsoft Office Automation, to controlling and extracting information hosted by a 3270 emulation session for the mainframe. All you need is an ASCII text file that ends in a ".VBS" extension and a little creativity.

A few notes on the language itself:

- Visual Basic Script Edition is a subset of the Visual Basic language.
- Comments are declared by a single apostrophe, such as: ' This is a VBSCRIPT comment.
- Variables can be declared using the DIM, PRIVATE, or PUBLIC keywords.
- Variables have no explicit data type. All data types are "variant" by default.
- The language is not case-sensitive, so "Acounter," "ACounter" and "aCounter" are considered to be the same variable name regardless of the mix of upper and lower case.
- Scope is declared using "keyword … END keyword" pairings, such as CLASS…END CLASS, SUB…END SUB, IF … END IF.
- The unit of program development is the SCRIPT file: an ASCII text file containing executable script statements.
- Statements are continued by leaving a space at the end of the line followed by an underscore.
- Strings are concatenated using the ampersand symbol as the concatenation operator.

FIGURE 1:NOTEPAD OR ANY ASCII EDITOR CAN BE USED FOR VBSCRIPT FILES

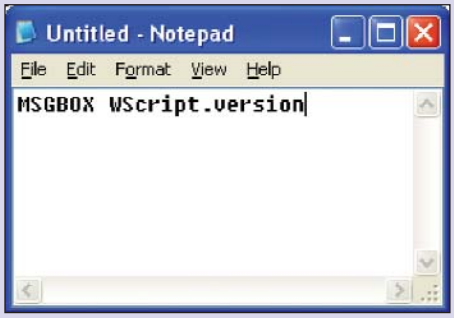
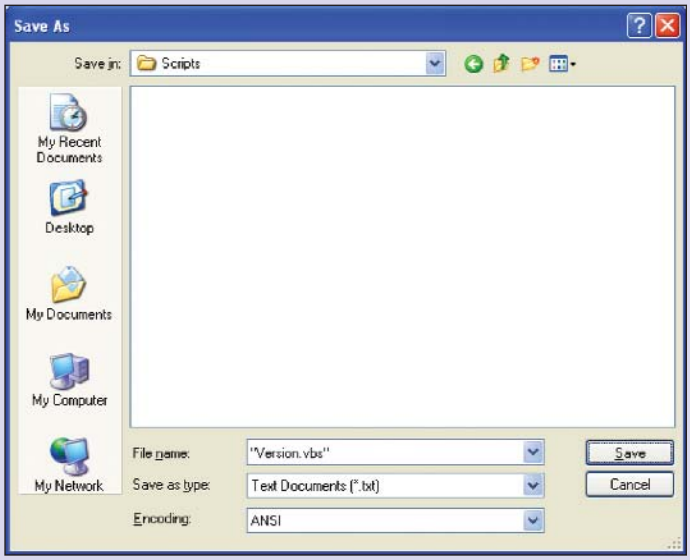


FIGURE 2: ENCLOSE THE FILE NAME IN QUOTES TO RETAIN THE .VBS EXTENSION



YOUR TOOLKIT FOR SCRIPT DEVELOPMENT

You'll need a text editor and a Windows client workstation running Windows 98 or above.

For the text editor use any ASCII only text editor like Windows Notepad.

The product that allows scripts to execute is Windows Script containing Visual Basic Script Edition. The current version at the time of this writing is version 5.6 and it is available for downloading at http://msdn.microsoft.com/scripting . But before you download anything try the following procedure to see if you have a current version installed on your desktop machine.

CHECKING FOR WINDOWS SCRIPT SUPPORT

Start by opening Notepad and entering the following line:

```
MSGBOX WScript.Version
```

Save the file as Version.VBS

Note: If you are using Windows Notepad enclose the filename in quotes to avoid having the .TXT extension appended to the filename.

Locate the saved Version.VBS file. You should see an icon that looks like FIGURE 3.

You can run the script by opening the saved file. The .VBS extension should be associated with WSCRIPT.EXE by default. Just double-click the icon. If everything goes well, you should see a pop-up message box like FIGURE 4.

INTRODUCTION TO OBJECTS

Object Oriented Programming (OOP) allows you to encapsulate data (fields) and behaviors (properties and methods) using a class. The relationship between a class and an object is similar to the relationship between a blueprint and the home that is built from the blueprint. The class just defines how the object will be built. A full implementation of OOP allows existing classes to be extended by inheritance to create sub-classes.

To actually build an object we need to declare a reference of the class and then call the class's constructor. Each object that is created from a class has its own protected set of variables, properties, and methods.

In FIGURE 1 we referenced an implicit object named Wscript and a property of that object named Version. Note: We did not have

FIGURE 3: THE DEFAULT ICON FOR VBSCRIPT FILES

FIGURE 4: OUTPUT SHOWING THE CURRENT SCRIPT VERSION

FIGURE 5: A SAMPLE VBSCRIPT CLASS DEFINITION: DEMOCLASS.VBS

```
class DemoClass

     public sub class_initialize
          msgbox "Construction in progress",vbOkOnly,"class_initialize"
     end sub

     public sub class_terminate
          msgbox "Destruction in progress",vbOkOnly,"class_terminate"
     end sub

     public sub showTime()
          msgbox Now(),vbOkOnly,"showTime"
     end sub

     public function getTime()
          getTime = Now()
     end function

end class
```

FIGURE 6: DECLARING AN OBJECT REFERENCE

```
dim objRef
```

FIGURE 7: CALLING THE DEMOCLASS INITIALIZER (CONSTRUCTOR)

```
set objRef = new DemoClass
```

FIGURE 8: CALLING AN OBJECT'S PUBLIC METHODS

```
objRef.showTime

msgbox objRef.getTime(),vbOkOnly,"Calling getTime"
```

FIGURE 9: INPUTBOX DIALOG


```
dim response
response=inputbox(msg,title,defaultValue)
```

to construct an instance (instantiate) of the Wscript object because the script engine had already made this object available to our script.

VBSCRIPT has limited (no inheritance) Object Oriented Programming support, which allows you to:

- create your own classes using the CLASS statement
- provide constructors (initializers) for your class using the SUB CLASS_INITIALIZE event
- provide destructors (clean up) for your classes using the SUB CLASS_TERMINATE event
- hide or expose data or behaviors using the PRIVATE and PUBLIC access modifiers
- provide methods within the class using the FUNCTION and SUB keywords
- expose access to hidden variables using the PROPERTY LET and GET statements
- create (instantiate) objects that reference your own classes using the NEW keyword

In addition to creating objects from your own classes, you can also create references to COM objects using the SET statement and the CreateObject method of the Wscript object.

CREATING AND USING AN OBJECT IN VBSCRIPT

First we need to define a class. We'll create a class named DemoClass containing a constructor, a destructor, and two public methods: showTime and getTime. The showTime method will not return a value so we'll use a SUB keyword. The getTime method will return a value with the current time so we'll use the FUNCTION keyword. We could also have added variables using the PRIVATE access modifier, and PUBLIC properties to retrieve (GET) and update (LET) the variables. Methods, variables, and properties are referred to as "members" of the class.

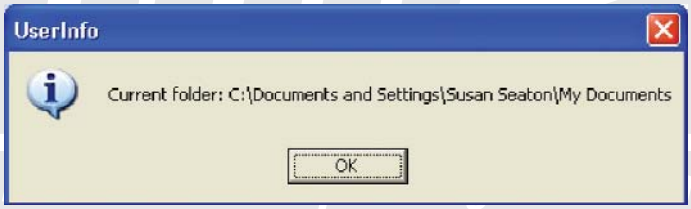
Next we need to declare a reference. This is just a placeholder that we can use to later address the object's members (methods and properties).

Finally, let's call the constructor to create the object and tie it to our reference. If this were a COM object we could also have used the CreateObject method instead of the NEW keyword. Note that Visual Basic Script uses the SET keyword to assign a value to an object

FIGURE 10: MSGBOX DIALOG


```
dim response
dim msg
dim title
msg = "Continue?"
title = "Confirm"
response=msgbox(msg,vbYesNo+vbInformation,title)
If response = vbYes then
Else
    ' No was clicked!
End if
```

FIGURE 11: USING THE POPUP METHOD OF THE SHELL OBJECT




```
dim btnReturn
dim waitSeconds
dim buttons
dim icon
dim wshShell
Set wshShell = CreateObject("Wscript.Shell")
buttons = vbOkOnly
icon = vbInformation
waitSeconds = 10
btnReturn = wshShell.Popup(msg, waitSeconds, title, buttons + icon)
```

FIGURE 12: VBSCRIPT CONSTANTS TO DEFINE MSGBOX BUTTON CONFIGURATIONS

Constant	Value	Description
vbOKOnly	0	Display OK button only.
vbOKCancel	1	Display OK and Cancel buttons.
vbAbortRetryIgnore	2	Display Abort, Retry, and Ignore buttons.
vbYesNoCancel	3	Display Yes, No, and Cancel buttons.
vbYesNo	4	Display Yes and No buttons.
vbRetryCancel	5	Display Retry and Cancel buttons.
vbCritical	16	Display Critical Message icon.
vbQuestion	32	Display Warning Query icon.
vbExclamation	48	Display Warning Message icon.
vbInformation	64	Display Information Message icon.
vbDefaultButton1	0	First button is the default.
vbDefaultButton2	256	Second button is the default.
vbDefaultButton3	512	Third button is the default.
vbDefaultButton4	768	Fourth button is the default.
vbApplicationModal	0	Application modal. The user must respond to the message box before continuing work in the current application.
vbSystemModal	4096	System modal. On Win16 systems, all applications are suspended until the user responds to the message box. On Win32 systems, this constant provides an application modal message box that always remains on top of any other programs you may have running.

FIGURE 13: MSGBOX CONSTANTS RETURNED BY THE MSGBOX DIALOG

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

FIGURE 14: REFERENCING THE USERINFO AND DESKTOPIO CLASS

```
Option Explicit
'
' Desktop.VBS
' written by John Papproth
' Demonstrates limited Desktop Input/Output and OOP
  facilities of VBSCRIPT
'
Dim objUserInfo
Dim objDesktopIO
Set objUserInfo = new UserInfo
Set objDesktopIO = new DesktopIO
objUserInfo.currentDirectory = _
     objDesktopIO.prompt("Enter a directory: ", _
          "Change Directory", _
          objUserInfo.currentDirectory)
Call objUserInfo.showDir()
'-----------------------------------------------------
```

reference. The class_initialize is invoked each time the class is instantiated into an object.

Finally we can use the reference to call our object's methods and properties using the dot notation. Each public member of the object is available by following the object reference with a period followed by the member name.

READY-MADE OBJECTS—WSCRIPT, WSHSHELL, WSHNETWORK, AND FILESYSTEMOBJECT

You have seen in our first example that the Wscript object is implicitly available to the script.

The following definitions were taken directory from the Microsoft Developer Network (MSDN) and describe other objects that are built into the scripting engine.

The **WScript** object is the root object of the Windows Script Host object model hierarchy. It never needs to be instantiated before invoking its properties and methods, and it is always available from any script file.

The **WScript** object provides access to information such as:

- command-line arguments
- the name of the script file
- the host file name
- host version information

The **WScript** object allows you to:

- create objects

FIGURE 15: DESKTOPIO CLASS ENCAPSULATES MSGBOX AND INPUTBOX

```
'
' Class DesktopIO
'
Class DesktopIO

 public sub alert(msg,title)
          msgbox msg,vbOkOnly+vbInformation,title
 end sub

 public function prompt(msg,title,defaultValue)
          prompt=inputbox(msg,title,defaultValue)
 end function

End Class
```

- connect to objects
- disconnect from objects
- sync events
- stop a script's execution programmatically
- output information to the default output device (either a Windows dialog box or the command console)

You create a **WshShell** object whenever you want to run a program locally, manipulate the contents of the registry, create a shortcut, or access a system folder. The **WshShell** object provides the **Environment** collection. This collection allows you to handle environmental variables (such as WINDIR, PATH, or PROMPT).

You create a **WshNetwork** object when you want to connect to network shares and network printers, disconnect from network shares and network printers, map or remove network shares, or access information about a user on the network.

The FileSystemObject is used to provide access to a computer's file system.

LIMITED INPUT AND OUTPUT TO THE WINDOW'S DESKTOP—INPUTBOX, MSGBOX, AND WSHSHELL.POPUP

For returning input text from the user, VBSCRIPT provides an INPUTBOX function that accepts three string values as input parameters: a prompt, a title for the window caption, and a default value. If the OK button is clicked, the value entered by the user is returned as a string. If the CANCEL button is clicked a zero length string is returned.

The MSGBOX also accepts three input parameters: a message to be displayed, a button and icon value, and a title for the caption of the dialog window. It displays a dialog box that waits for the user to click a button before returning to the script. Unlike INPUTBOX the response that is returned is not text, but the numeric value corresponding to the button that was clicked. These numeric values are represented by mnemonic constants.

WSHSHELL.POPUP—MSGBOX WITHOUT THE WAIT!

The wshShell.PopUp method performs in a manner similar to the MSGBOX function. However, the POPUP method does not have to wait for a response by the user. A number of seconds to wait is supplied to the POPUP function as an additional input parameter. If "SecondsToWait" equals zero (the default), the pop-up message box

remains visible until closed by the user. In this case the effect is the same as MSGBOX. However, if "SecondsToWait" is greater than zero, the pop-up message box closes after "SecondsToWait" seconds have elapsed with no interaction from the user.

MSGBOX (AND WSHSHELL.POPUP) CONSTANTS

The Msgbox statement and the PopUp method can share the same VBSCRIPT MSGBOX constants that are used to define the values for the buttons and icon. The following tables were taken directly from the VBSCRIPT Reference page on MSDN.

The following constants are used with the MsgBox function to identify what buttons and icons appear on a message box and which button is the default. In addition, the modality of the MsgBox can be specified. Since these constants are built into VBSCRIPT, you don't have to define them before using them. Use them anywhere in your code to represent the values shown for each.

The following constants are used with the MsgBox function to identify which button a user has selected.

PIECES OF A WORKING SCRIPT

The following example, DESKTOP.VBS, was developed to demonstrate the basic desktop input and output facilities that are available to the script developer. The script itself also uses encapsulation to demonstrate the Object Oriented capabilities in VBSCRIPT. It is composed of the script mainline and two in-line class definitions. All definitions are contained in a single text file named: Desktop.VBS

The function of this script is a bit contrived (but serves as a good demonstration): to prompt for a directory name, change the current directory, and then list the files within that directory.

The script mainline declares two variables to hold object references of each class type.

Public methods and properties of the objects are called to change and then display the contents of a folder.

The DesktopIO class simply provides an encapsulation around the MSGBOX and INPUTBOX functions. While this was definitely not necessary it provides a simple example of the CLASS statement and the use of both a SUB and FUNCTION as methods

FIGURE 16: USERINFO CLASS EXPOSING PRIVATE FIELDS WITH PUBLIC PROPERTIES

```
'
' Class UserInfo
'
Class UserInfo

 private strDomainName
 private strComputerName
 private strUserName
 private strCurrentDirectory
 private wshShell
 private wshNetwork
 private fso
 private io

 public sub class_initialize
     set wshShell = CreateObject("Wscript.Shell")
     set wshNetwork = CreateObject("Wscript.Network")
     set fso = CreateObject("Scripting.FileSystemObject")
     set io = new DesktopIO
     strDomainName = WshNetwork.UserDomain
     strComputerName = WshNetwork.ComputerName
     strUserName = WshNetwork.UserName
     strCurrentDirectory = WshShell.CurrentDirectory
     call show("Hello " & strUserName & vbCrLf & _
          "Logged in at: " &  _
          "\\" & strDomainName & "\" & strComputerName & vbCrLf & _
          "Current folder: " &  _
          strCurrentDirectory & _
          "","UserInfo")
 end sub

 public sub class_terminate
     call show("Goodbye " & strUserName,"UserInfo")
       set wshShell = Nothing
     set wshNetwork = Nothing
     set fso = Nothing
     set io = Nothing
end sub

 public property GET currentDirectory()
        currentDirectory = strCurrentDirectory
 end property

 public property LET currentDirectory(value)
        WshShell.CurrentDirectory = value
     strCurrentDirectory = WshShell.CurrentDirectory
     call show("Current folder: " &  _
          strCurrentDirectory & _
          "","UserInfo")
 end property

 public sub showDir()
     dim folder
     dim file
     dim fileList
     set folder = fso.GetFolder(strCurrentDirectory)
     fileList = ""
        for each file in folder.Files
          fileList = fileList & file.name & vbCrLf
        next
     call io.alert(fileList,"Dir for " & strCurrentDirectory)
 end sub

 private function show(msg,title)
     dim btnReturn
     dim waitSeconds
     dim buttons
     dim icon
     buttons = vbOkOnly
     icon = vbInformation
     waitSeconds = 10
     btnReturn = wshShell.Popup(msg, waitSeconds, title, buttons + icon)
     show = btnReturn
 end function

End Class
```

of the class. In VBSCRIPT, a method created using the SUB keyword cannot return a value, while a method created as a FUNCTION returns a value by setting the name of the FUNCTION equal to the return value before exiting.

The UserInfo class is an example of encapsulating both data and behavior together. All data items are hidden from direct access outside of the class by using the PRIVATE access modifier. The SUBs CLASS_INITIALIZE and CLASS_TERMINATE are used to instantiate and then release the object references that are used by the class.

This class also provides examples of:

- a READ/WRITE property, currentDirectory.
- a public method, showDir, which is used to display the directory contents.
- a private method, show, which acts as a wrapper around the wshShell.PopUp method.

CREATING AND READING ASCII FILES—FILESYSTEMOBJECT AND THE TEXTSTREAM OBJECT

The FileSystemObject is used to provide access to a computer's file system. Two of its methods will allow us to create a TextStream object. The TextStream object is used to provide sequential access to a file. You can create a new file or overwrite an existing file by using the FileSystemObject.CreateTextFile method. You can open both a new and existing file by using the FileSystemObject.OpenTextFile method.

A CLASS TO READ AND WRITE TEXT FILES

The TextIO class that is defined below encapsulates the relationship between the FileSystemObject and the TextStream object and provides several methods for interacting with text files. The code in FIGURE 18 demonstrates the use of the TextIO class. We begin by creating a TextIO object reference, and then demonstrate how to create, append, and read from a text file.

ADVANCED INPUT/OUTPUT USING HTML AND ACTIVEX

You can combine the class definitions, script, and HTML to take full advantage of the presentation capabilities that are available in the browser.

In the example below we added a script tag to bring in our TextIO class definition as a separate file. We then added a method to read the file using the filename that was selected in the HTML page. Finally we tied the readFile method to the onClick event of the READ button on the HTML page. The result is an ActiveX browser that reads external text files into a table on the HTML page.

FIGURE 17: TEXTSTREAM OBJECT (CREATED BY FILESYSTEMOBJECT METHODS)

Methods:	Properties:
Close Method (FileSystemObject object)	AtEndOfLine Property
Read Method	AtEndOfStream Property
ReadAll Method	Column Property Line Property
ReadLine Method	
Skip Method	
SkipLine Method	
Write Method	
WriteBlankLines Method	
WriteLine Method	

FIGURE 18: IOTEST MAINLINE REFERENCING THE TEXTIO CLASS

```
Option Explicit
'
' IOTest.vbs
' written by John Papproth
' Demonstrates the text Input/Output facilities of VBScript
'
dim i
dim io
set io = new TextIO
'
' Create
'
msgbox "Creating a new file!",vbOkOnly,"IO Test"
call io.openOutput("IOTest.txt")
for i = 1 to 5
     call io.putLine("Line " & i & "..." & Now)
next
call io.close
'
' Append
'
msgbox "Appending to an existing file!",vbOkOnly,"IO Test"
call io.openAppend("IOTest.txt")
for i = 1 to 5
     call io.putLine("Appended Line " & i & "..." & Now)
next
call io.close
'
' Read
'
msgbox "Reading the file!",vbOkOnly,"IO Test"
call io.openInput("IOTest.txt")
do
     msgbox io.getLine()
loop until io.EOF
call io.close
```

AUTOMATING DESKTOP ADMINISTRATION

MSDN states that Windows Management Instrumentation (WMI) provides access to information about objects in a managed environment. Through WMI and the WMI application programming interface (API), applications can query for and make changes to static information in the Common Information Model (CIM) repository and

dynamic information maintained by the various types of providers.

Combining script and WMI gives you easy access to the objects within the Windows Operating System.

For more information on WMI check out MSDN at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/anch_wmi.asp

See FIGURE 21 for a script that reboots the workstation after a 10-second delay.

AUTOMATING A MICROSOFT OFFICE APPLICATION

Microsoft Office applications can be automated using Visual Basic for Applications (VBA) which runs within the Office product as a macro extension. However, you can also control Microsoft Office by creating a reference to the Application object.

FIGURE 22 is a short example to demonstrate how easy it is to start an office application and then interact it with using VBScript.

Each office application has a macro recording facility that creates VBA as you are manually performing the steps (such as changing fonts, saving files, etc). See the Tools/Macro menu under your office application to try this.

After recording a macro in VBA you can then view the recorded VBA statements and change them into a stand alone VBSCRIPT by qualifying the object references with your script application object name (objExcel in FIGURE 22).

AUTOMATING ATTACHMATE MAINFRAME HLLAPI FROM VBSCRIPT

Attachmate Extra has a facility for creating macros by recording keystrokes and producing VBA-like output. In a manner similar to automating Microsoft Office, we can also create stand-alone scripts for our host session.

FIGURE 23 is an example of a script that connects to an existing Host session, sends a simple command, and then captures and displays the output.

WHERE TO GO FOR MORE INFORMATION

Microsoft provides many code examples on its scripting development center: http://msdn.microsoft.com/scripting

FIGURE 19: TEXTIO.VBS ENCAPSULATES TEXTSTREAM METHODS IN AN EXTERNAL CLASS

```
'
' Class TextIO
'
Class TextIO

     Private ForReading     ' constants for IO
     private ForWriting     ' constants for IO
     private ForAppending   ' constants for IO

     private fso            ' File System Object
     private ts             ' Text Stream
     private s              ' Stream line

     private sub Class_Initialize
          ForReading = 1
          ForWriting = 2
          ForAppending = 8
          Set fso = CreateObject("Scripting.FileSystemObject")
     End sub

     private sub Class_Terminate
     end sub

     public function openInput(s)
          Set ts = fso.OpenTextFile(s, ForReading, True)
     end function

     public function openOutput(s)
          Set ts = fso.OpenTextFile(s, ForWriting, True)
     end function

     public function openAppend(s)
          Set ts = fso.OpenTextFile(s, ForAppending, True)
     end function

     public function getLine()
          Dim s
          s = ts.readLine()
          getLine = s
     end function

     public function putLine(s)
          ts.WriteLIne(s)
     end function

     public function close()
          ts.Close
     end function

     public Property GET EOF()
          EOF = ts.AtendOfStream
     End property

end Class
```

Although this article has focused on VBSCRIPT, the Windows Script product also supports Jscript (Microsoft's version of Javascript). If you are so inclined, you may also want to experiment with other scripting languages such as ActiveState's Perl (http://www.activestate.com/perl).

DID I MENTION?…VB.NET, ADO, WINHTTP, AND MSSOAP

The real power of VBScript (or any script interface) is the ability to create and attach to COM objects. Did I mention that you could create your own COM objects using Visual Basic 6.0 or COM Wrappers for VB.NET Components?

But before you create your own objects, do a little research on MSDN. There are existing COM objects for database (ADODB), web document (WINHTTP), and web service (MSSOAP) processing!

RESOURCES

http://msdn.microsoft.com/scripting

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/anch_wmi.asp
http://www.activestate.com/perl

NaSPA member John Papproth is both a MCSD.NET (Microsoft Certified Solution Developer) and SCJP (Sun Certified Java Programmer). He is a consultant for Bass & Associates Inc. a Business Solutions Provider in Omaha NE. John has taught courses in both the Microsoft .NET language suite and the Sun Microsystems Java Language.

FIGURE 20: READFILE.HTML USING AN HTML PAGE FOR PRESENTATION

```
<html>
     <head><title>Read File</title></head>
     <script language="vbscript" src="TextIO.vbs">
     </script>
     <script language="vbscript">
          option explicit
          sub readFile()
               dim io
               dim s
               dim f
               f = InputFile.value
               set io = new TextIO
               io.openInput(f)
               s = ""
               do
                    s = s & io.getLine() & vbCrlf
               loop until io.EOF
               io.close
               contentName.innerText = f
               content.innerText = s
          end sub
     </script>
     <body>
          <font face='Arial' size=4>
               Click BROWSE to select a file:
                      <input type=file id=InputFile />
               <br clear=all />
               Click READ/SHOW to view the selected file:
                      <input type=button
                             id=btnRead
                             value='Read/Show'
                             onClick='readFile()' />
               <br clear=all />
               <table border=1 align=center cellspacing=0 cellpadding=10>
                    <tr>
                    <td>
                         <font face='Arial' size=4>
                              <div id=contentName></div>
                         </font>
                    </td>
                    </tr>
                    <tr>
                    <td>
                         <font face='Courier New'>
                              <div id=content></div>
                         </font>
                    </td>
                    </tr>
               </table>
          </font>
     </body>
</html>
```

FIGURE 21: SYSTEMREBOOT.VBS USING THE WMI API

```
Option Explicit
'------------------
'
' SystemReboot.vbs
' Written by John Papproth
'   Reboots after 10 seconds
'
'------------------
Dim obj
Set obj = new SysCmd
Call obj.Reboot

'
' SysCmd Class
'      Methods: Logoff,Shutdown,Reboot,Poweroff,Confirm,Notify,Shell
'
Class SysCmd
     '-------------------------------------------------------------
     '
     ' private:
     '       wshNetwork
     '     wshShell
     '
     '     Private Sub Class_Initialize
     '     Private Sub Class_Terminate
     '     Private Sub Win32Shutdown(opt)
     '     Private Function popup(Prompt,Title,SecondsToWait,Buttons,Icons)
     '
     ' public:
     '     Public Sub Logoff()
     '     Public Sub Shutdown()
     '     Public Sub Reboot()
     '     Public Sub Poweroff()
     '     Public Function Confirm(Prompt,Title,SecondsToWait)
     '     Public Sub Notify(Prompt,Title,SecondsToWait)
     '     Public Sub Shell(cmdString)
     '
     '-------------------------------------------------------------
     private wshNetwork
     private wshShell

     Private Sub Class_Initialize
          set wshNetwork = CreateObject("Wscript.Network")
          Set wshShell = CreateObject("Wscript.Shell")
     End Sub

     Private Sub Class_Terminate
          ' Nothing to do here!
     End Sub

     Private Sub Win32Shutdown(opt)
          if Confirm("Continue?","Shutting down...",10) then
               dim objWMIService
               dim strComputer
               dim colItems
               dim objOperatingSystem
               strComputer = "."
               Set objWMIService = GetObject("winmgmts:" & _
                    "{impersonationLevel=impersonate,(Shutdown)}!\\" & _
                    strComputer & _
                    "\root\cimv2")
               set colItems = objWMIService.ExecQuery( _
                    "Select * from Win32_OperatingSystem" _
                    )
               for each objOperatingSystem in colItems
                    objOperatingSystem.Win32Shutdown(opt)
               next
          end if
     End Sub

     Public Sub Logoff()
          const LOGOFF = 0
          Win32Shutdown(LOGOFF)
```

Continued on next page

FIGURE 21: CONTINUED

```
    End Sub

    Public Sub Shutdown()
         const SHUTDOWN = 1
         Win32Shutdown(SHUTDOWN)
    End Sub

    Public Sub Reboot()
         const REBOOT = 2
         Win32Shutdown(REBOOT)
    End Sub

    Public Sub Poweroff()
         const POWEROFF = 8
         Win32Shutdown(POWEROFF)
    End Sub

    Private Function popup(Prompt,Title,SecondsToWait,Buttons,Icons)
         dim btnReturn
         btnReturn = wshShell.Popup( _
              Prompt, _
              SecondsToWait, _
              Title, _
              Buttons + Icons)
         popup = btnReturn
    End Function

    Public Function Confirm(Prompt,Title,SecondsToWait)
         if popup(Prompt,Title, SecondsToWait, vbYesNo, vbQuestion) = vbNo then
              Confirm=false
         else
              Confirm=true
         end if
    End Function

    Public Sub Notify(Prompt,Title,SecondsToWait)
         Call popup(Prompt, Title, SecondsToWait, vbOkOnly, vbInformation)
    End Sub

    Public Sub Shell(cmdString)
         wshShell.run(cmdString)
    End Sub

End Class
```

FIGURE 22: EXCELTEST.VBS SCRIPTING MICROSOFT OFFICE

```
Option Explicit
'
' ExcelTest.vbs
' written by John Papproth
'
dim objExcel
dim row
dim col
Set objExcel = CreateObject("Excel.Application")
objExcel.Workbooks.Add
row = 1
col = 1
objExcel.ActiveSheet.Cells(row,col) = Now
objExcel.Selection.NumberFormat = "mmmm d, yyyy"
objExcel.Selection.Font.Name = "Arial"
objExcel.Selection.Font.FontStyle = "Regular"
objExcel.Selection.Font.Size = 14
objExcel.ActiveWorkbook.SaveAs "ExcelTest.xls"
objExcel.Visible = True
```

FIGURE 23: SCRIPTING THE MAINFRAME WITH ATTACHMATE EXTRA!

```
Option Explicit
'
' HostTester.vbs
' written by: John Papproth
'
' Purpose: Demonstrates a connection to an Attachmate Extra Session
'
'
Call Main

Sub Main()
     Dim System     ' Attachmate System
     Dim Session     ' Current Host Session
     Dim Screen     ' Current Screen
     Dim milliseconds ' milliseconds to wait
     Dim Rows     ' Rows in Current Screen
     Dim Cols     ' Cols in one Row

     Dim Row          ' Row Counter
     Dim Buffer     ' Buffer to hold Screen Image
     Dim Line     ' One Line
     Dim Lines     ' All Lines belimited by vbCrLf

     milliseconds = 500     ' 1000=1 second

     Set System = CreateObject("Extra.System")
     if System is Nothing Then
          Msgbox "Could not create the System Object", _
                    vbOkOnly+vbCritical,"Get Screen"
          Exit Sub
     End If

     Set Session = System.ActiveSession
     if Session is Nothing Then
          Msgbox "Could not create the Session Object", _
                    vbOkOnly+vbCritical,"Get Screen"
          Exit Sub
     End If

     Set Screen = Session.Screen

     Call Screen.SendKeys("TIME<Enter>")     ' Can be any HOST command
     Call Screen.WaitHostQuiet(milliseconds)

     Rows = Screen.Rows()
     Cols = Screen.Cols()
     Buffer = Screen.GetString(1,1,Rows*Cols)
     Lines = ""
     For Row = 1 to Rows
          Line = Mid(Buffer, ((Row-1)*Cols) + 1, Cols)
          Lines = Lines & Line & vbCrlf
     Next
     Msgbox Lines,vbOkOnly+vbInformation,"Screen"
End Sub
```