WSH – Windows Script Host

Windows Script Host (**WSH**), a feature of the Microsoft® Windows® 2000 family of operating systems, is a powerful multi-language scripting environment ideal for automating system administration tasks. Scripts running in the **WSH** environment can leverage the power of **WSH** objects and other **COM-based** technologies that support Automation, such as **Windows Management Instrumentation** (**WMI**) and **Active Directory Service Interfaces** (**ADSI**), to manage the Windows subsystems that are central to many system administration tasks.

What is WSH?

The first time people encounter Windows Script Host (**WSH**), they often express some confusion. What exactly is **WSH**? Is it a language, like **VBScript** or **JScript**? No; although **WSH** enables you to run programs written in these languages, it is not a language itself. Is it an object model, like **WMI** or **ADSI**? No; **WSH** does provide a simple object model, but providing an object model is not its primary purpose.

So then what is WSH?
As the name implies, **WSH** is a script host. A script host is a program that provides an environment in which users can execute scripts in a variety of languages, languages that use a variety of object models to perform tasks.

You are probably already familiar with other script hosts. Microsoft® Internet Explorer, for example, enables users to execute scripts that use the **Dynamic HTML** object model. **Shell** programs (such as **C Shell**, **Bourne Shell** and **Korn Shell**) enable you to write scripts that use an object model capable of manipulating the file system. Even the command prompt can be thought of as a scripting environment because it can run scripts written in the "batch file" language.

WSH is an unusual script host in that it was designed to be general-purpose. Unlike most of the scripting tools mentioned above, **WSH** imposes restrictions on neither the language used to write scripts nor the object models used by scripts.

WSH capabilities can be divided into four major areas. These areas, which will be discussed in detail throughout the remainder of this chapter, include:

- The ability to intrinsically carry out system administration and programming tasks
- The ability to use **COM** objects to carry out system administration tasks
- The ability to add standard programming features to **WSH**-compatible scripting languages
- The ability to run command-line tools

WHS Links

- WSH home at MSDN - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/d78573b7-fc96-410b-8fd0-3e84bd7d470f.asp

- WSH reference - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/60f96092-6021-42f3-9a27-6848f4c38f64.asp
- Microsoft Windows Script Downloads - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/f2c93afe-483a-482b-9f03-3ecc8aa40e78.asp
- DevGuru WSH Quick Reference - http://devguru.com/technologies/wsh/

WSH Major Areas

The ability to intrinsically carry out system administration and programming tasks

In many ways, this might be the least important of the **WSH** capabilities. **WSH** is primarily a scripting runtime; it provides the environment in which scripts can run, in much the same way that the command processor provides the environment in which batch files can run.

However, even though **WSH** primarily serves as the conduit through which other scripting languages and technologies operate, it is still possible to use "pure" WSH scripts to carry out system administration tasks. You can use WSH to do such things as map and unmap network drives, and add and remove printer connections.

The ability to use COM objects to carry out system administration tasks

As noted, **WSH** can be used to map network drives and to add printer connections. Beyond that, however, few system administration tasks can be carried out using **WSH** alone. You cannot use **WSH** to take inventory of computer hardware or to determine the software that is installed on a computer.

But even though **WSH** has no intrinsic methods for carrying out these tasks, you can still perform system administration chores by using a **WSH** script. This is possible because **WSH** allows you to use **COM** (Component Object Model) objects within your scripts.

COM objects are a standard way for applications (.exe files) or programming libraries (.dll files) to present their capabilities as a series of objects. In turn, **WSH** can bind (connect) to these objects and harness these capabilities.
In fact, **WSH** can access any **COM** object that supports Automation. Automation refers to a standard way of accessing a **COM** object. For the most part, scripting languages can access only **COM** objects using Automation

The ability to add standard programming features to WSH

Applications vary widely both in what they are intended to do and in how they go about doing it; Calculator, for example, bears little resemblance to Ipconfig.exe, which bears even less resemblance to Microsoft® Word. Despite these wide variations, however, applications share some basic attributes. Many applications provide for input and output: They allow users to enter data, and they provide a method for displaying data to users. Many applications can read and write to the

registry. Many applications accept command-line arguments that affect how the application runs or what the application does. This is true even of graphical applications.

The ability to run command-line tools

Obviously, **WSH** is not required to run command-line tools; command-line tools can be run as stand-alone programs or can be called from batch files. However, **WSH** is extremely useful if you want to add "intelligence" to these tools or batch files.

The WSH Object Model

People typically speak of **WSH** as though it were a single item. In truth, however, **WSH** is made up of multiple components. For example, the **WSH** environment includes a built-in object, **WScript**, and three **COM** objects: **WshShell**, **WshNetwork**, and **WshController**. Together, **WScript**, **WshShell**, **WshNetwork**, and **WshController** are referred to as the **WSH** objects. Your scripts access the capabilities of these objects through the **WSH** Object Model.

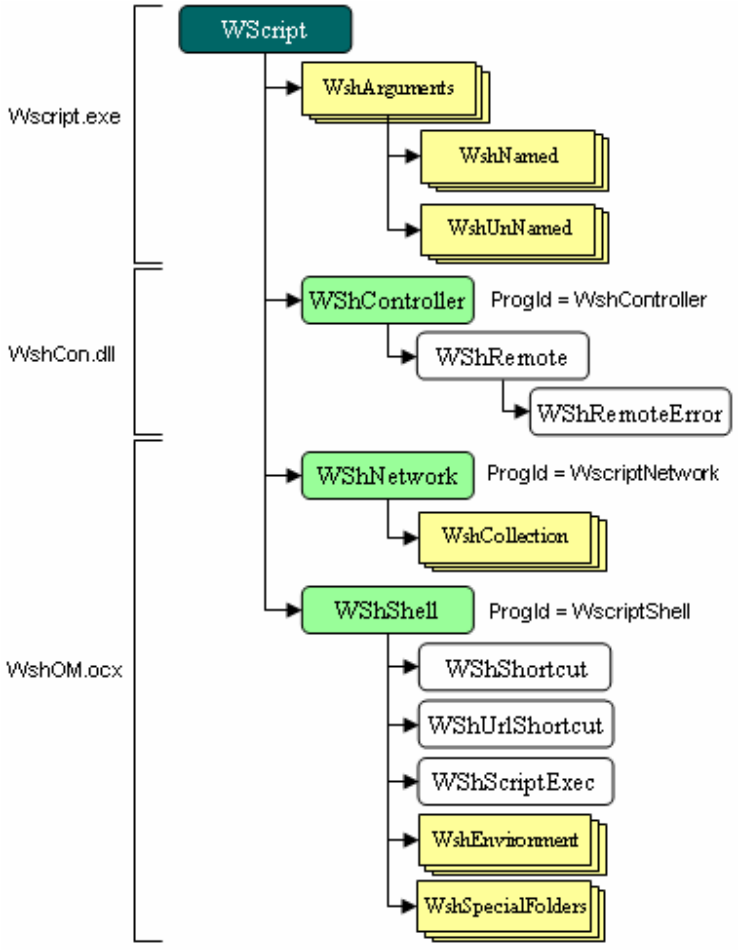

Figure 1 - WSH object Model

Object	Description
WScript	Provides access to most of the objects, methods, and properties in the **WSH** object model.
WshArguments	Gives you access to the entire collection of command-line parameters, in the order in which they were originally entered.
WshController	Exposes the method **CreateScript**() that creates a remote script process.
WshEnvironment	Gives you access to the collection of Microsoft Windows system environment variables.
WshNamed	Provides access to the named command-line script arguments within the **WshArguments**
WshNetwork	Gives you access to the shared resources on the network to which your computer is connected.
WshRemote	Provides access to the remote script process.
WshRemoteError	Exposes the error information available when a remote script (a **WshRemote** object) terminates as a result of a script error.
WshScriptExec	Provides status and error information about a script run with Exec, along with access to the **stdIn**, **stdOut**, and **stdErr** channels.
WshShell	Gives you access to the native Windows shell functionality.
WshShortcut	Allows you to create a shortcut programmatically
WshSpecialFolders	Allows you to access the Windows Special Folders.
WshUnnamed	Provides access to the unnamed command-line script arguments within the **WshArguments** object.
WshUrlShortcut	Allows you to create a shortcut to an Internet resource, programmatically.

WSH is not intended to be a self-contained, all-encompassing object model. It focuses on three major areas:

- Providing resources necessary to support script execution. For instance, the WScript object reports on the interpreter and version of **WSH** in use, while the **WshShell** object allows shortcuts and Internet shortcuts to be created.
- Enhancing the ease with which a system can connect to and disconnect from network resources. This functionality is supported by the **WshNetwork** object.
- Supporting functionality that is not readily available in other object models. For example, the **WshShell** object allows access to environment variables and to the location of Windows system folders.

WSH Objects and Services

- **Windows Script Host** (**WSH**) is a Windows administration tool.
- **WSH** creates an environment for hosting scripts. That is, when a script arrives at your computer, **WSH** plays the part of the host, it makes objects and services available for the script and provides a set of guidelines within which the script is executed. Among other things, **Windows Script Host** manages security and invokes the appropriate script engine.
- **WSH** is language-independent for **WSH**-compliant scripting engines. It brings simple, powerful, and flexible scripting to the Windows platform, allowing you to run scripts from both the Windows desktop and the command prompt.
- **Windows Script Host** is ideal for non-interactive scripting needs, such as logon scripting, administrative scripting, and machine automation.
- **WSH** and the **WSH objects** do not include all the things system administrators might want to do; far from it. In addition, even tasks that are covered by the **WSH** objects are often better handled by using technologies such as **WMI** and **ADSI**.
- The **WSH** environment includes the built-in **WScript** object and three **COM objects**: **WshShell**, **WshNetwork**, and **WshController**. Your scripts can use these objects to help automate system administration tasks.
- **Windows Script Host** provides several objects for direct manipulation of script execution, as well as helper functions for other actions. Using these objects and services, you can accomplish tasks such as the following:
- Print messages to the screen.
- Run basic functions such as **CreateObject** and **GetObject.**
- Map network drives.
- Connect to printers.
- Retrieve and modify environment variables.
- Modify registry keys.

Using COM Objects

If you have a question about practically anything sports, history, science, gardening it is likely that you can find the answer at the public library. However, this does not mean that you can walk into the library, pick out a book at random, and expect to find the answer. Instead, answers to specific questions are found in specific books, and you need to locate the correct book if you want to have your question answered.

The same thing applies to **COM** objects. There is likely to be a **COM** object that can be used to script most of your system administration needs. However, you cannot use just any **COM** object; **COM** objects have specific, oftentimes unique capabilities. By the same token, you cannot simply start using a **COM** object, just as you cannot start reading a book without first finding that book. (The one exception is the **WScript** object, which you can simply start using.) Instead, before a script can use a **COM** object, it must first bind to that object. The **WScript** object provides two methods for creating **COM** objects: **CreateObject**

and **GetObject**.

Creating a New Instance of a COM Object

To create a new instance of a **COM** object, a script can call the **WScript CreateObject** method and pass it the Programmatic Identifier (**ProgID**) of the **COM** object by using the following syntax:

```
CreateObject("ProgID")
```

To continue the analogy with the library, the **ProgID** is roughly equivalent to the call number assigned to a book. If you go to the library and give the librarian the call number, he or she can locate the book for you. Likewise, if you pass the scripting host a **ProgID**, the scripting host can look in the registry and locate the **COM** object you want to create. **ProgIDs** are unique to each **COM** object, just as call numbers are unique to each book.

How do you know the correct **ProgID** for a given **COM** object? Unfortunately, there is no simple answer to that question; your best course of action is to look at the documentation that accompanies the object. All **ProgIDs** are stored in the **HKEY_CLASSES_ROOT** portion of the registry, as shown in Figure 3.4. However, this listing is of only limited use because not all of these ProgIDs can be accessed using scripts.

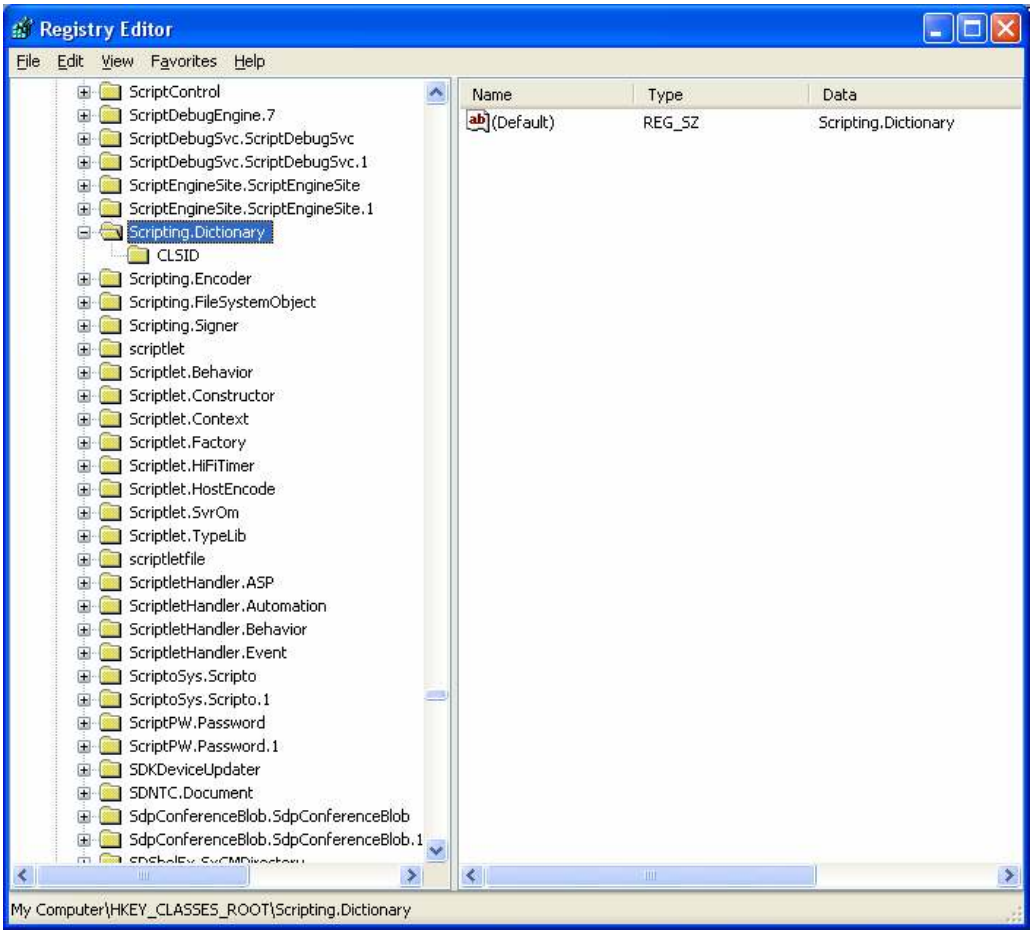

Figure 2 - ProgIDs in the Registry

To be able to use a newly created object, the script needs to store a reference to the object in a variable by using the following syntax:

Set objVariable = WScript.CreateObject (" ProgID")

After a reference to the object is stored in a variable, the script can call a method or access a property of the object by using dot notation.

Using a COM Object

The following script creates a new instance of the ADSI System Information object (using the ProgID ADSystemInfo), stores a reference to it in the objSysInfo variable, and then displays the Domain Name System (DNS) domain name for the logged-on user.

```
Set oSysInfo = CreateObject("ADSystemInfo")
MsgBox "Domain DNS name: " & oSysInfo.DomainDNSName
```

For example, the following lines of code bind to various **COM** objects. From line to line, the only item that changes is the ProgID:

```
Set oWord = CreateObject("Word.Application")
Set oIE = CreateObject("InternetExplorer.Application")
```

```
Set oDict = CreateObject("Scripting.Dictionary")
Set oNet = CreateObject("Wscript.Network")
```

Attaching to an Existing Instance of a COM Object

If the **COM** object you want to use is already running, you can use that existing object rather than create a new instance. The **WScript GetObject** method lets you reference and use a previously instantiated object instead of creating a new one.

When you write scripts that use **WMI** or **ADSI**, you will typically use the **GetObject** method; this is because both **WMI** and **ADSI** are always available. (For example, you can stop the **WMI** service, but if you run a script that uses **WMI**, the service will automatically restart.) Although there are exceptions, a typical **WMI** script might start like this:

```
Set oWMIService = GetObject("winmgmts:")
```

When you are writing scripts that use the **WSH** objects, you will usually use the **CreateObject** method because the **WSH** objects are not usually preinstantiated.

Comparing CreateObject and GetObject Functions with WSH

The **VBScript** language also provides **CreateObject** and **GetObject** functions. The **VBScript CreateObject** function and the **WScript CreateObject** method both instantiate **COM** objects when they are called with a single parameter, the **ProgID** of the **COM** object to instantiate. For example, these two lines of code the first using the **VBScript** version of **CreateObject** and the second using the **WSH** version are functionally identical; both instantiate an instance of the **FileSystemObject**:

```
Set oFSO = CreateObject("Scripting.FileSystemObject")
Set oFSO = Wscript.CreateObject("Scripting.FileSystemObject")
```

Both versions of **CreateObject** can also accept a second parameter; however, each interprets this second parameter in a completely different way. Consider these two lines of code. The first line uses **VBScript**, and the second uses **WSH**:

```
Set oExcel = CreateObject("Excel.Application", "Parameter2")
Set oExcel = Wscript.CreateObject("Excel.Application", "Parameter2")
```

The **VBScript CreateObject** function interprets the second parameter as a remote computer name and tries to create the **COM** object on that remote computer; in this example, it tries to instantiate an instance of **Microsoft Excel** on a remote computer named *Parameter2*. The **WScript CreateObject** method interprets a second parameter as a subroutine prefix to be used in handling events from the object. The two **GetObject** functions are similarly related.

The **VBScript CreateObject** function interprets the second parameter as a remote computer name and tries to create the **COM** object on that remote computer; in this example, it tries to instantiate an instance of **Microsoft Excel** on a remote computer named *Parameter2*. The **WScript CreateObject** method interprets a second parameter as a subroutine prefix to be used in handling events from the object. The two **GetObject** functions are similarly related.

On the other hand, if you want to use the remote object creation or event-handling capabilities, you need to choose the method or function that provides that additional functionality.

QuickTest Professional does not support the WScript directly, so **Wscript.CreateObject** and **WScript.GetObject** can't be implemented.

Where is WSH?

Windows Script Host is built into Microsoft Win98, 2K, Millennium Editions and WinXP. If you are running Win95, you can download **Windows Script Host 5.6** from the **Microsoft Windows Script Technologies** Web site

http://msdn.microsoft.com/scripting

You can also go to the web site listed above to upgrade your current engines. The version of **WSH** in Windows 98, 2000, and Millennium Editions is either version 1.0 or 2.0. You must upgrade to version 5.6 to get the new features.

Hosting Environments and Script Engines

Scripts are often embedded in Web pages, either in an **HTML** page (on the client side) or in an **ASP** page (on the server side), in the Automation tool, **VBScript** is embedded in the **QuickTest** engine.

WScript and CScript

wscript.exe is a process relating to Microsoft Windows operating system which allows additional functions to scripting. This program is a non-essential system process, but should not be terminated unless suspected to be causing problems. **WSH** is really made up of three main components:

- The core host that ties all the pieces together;
- The scripting engines that provides the scripting languages, such as **VBScript**, **JScript** and **PerlScript**
- The **Scripting Hosts** that actually execute the scripts, two are shipped as standard **WSCRIPT.EXE** (a Windows GUI version) and **CSCRIPT.EXE** (a text mode command-line version)
- **Windows Script Host** enables you to run scripts from Windows.
- **WScript.exe** provides a Windows-based dialog box for setting script properties.
- Using **WScript.exe**, you can run scripts under Windows in the following ways.
- Whether you use **WScript** or **CScript**, you still run the scripts in the same manner.
- The difference is only in the output — **WScript** generates windowed output, while **CScript** sends its output to the command window in which it was started.

Scheduling the Running of Scripts

Tasks that you script often need to be done repeatedly according to a prescribed schedule. You can use the Windows 2000 Task Scheduler or At.exe to schedule the running of these scripts. The scripts still run under one of the script hosts, but they run at the designated times without your interaction.

The ability to run scripts without the need for any human intervention is one of the major advantages of automation scripting.

For example, suppose you want to have a script that runs once a week and performs a Regression/Smoke test. There is no need for you to remember to manually run this script each week, and no need for you to arrange for someone else to run this script should you be out of the office. Instead, the script can be scheduled to run once a week, and it can do so without any need for human intervention. As an added bonus, the script can be scheduled to run during off-hours, thus minimizing any disruption to users.

You can also use **WMI** to create, delete, and manage scheduled tasks.

For example, this script creates a scheduled task that runs a script (vbs) that activates **QuickTest** and runs a test, every Monday, Wednesday, and Friday at 12:30 P.M.

```
Set oService = GetObject("winmgmts:")
Set oJob = oService.Get("Win32_ScheduledJob")
oNewJob.Create _
    ("Wscript c:\scripts\monitor.vbs", "********123000.000000-420", _
        True , 1 OR 4 OR 16, , , JobID)
```

Running WSH Scripts in QuickTest

WScript Object

The **WScript** object, the top-level object in the **WSH** object model, provides information about the script host (that is, about **WScript.exe** or **CScript.exe**) and the script file it is executing as well as provides access to other objects. The **WScript** object was embedded on **QuickTest**, and replaced by the **QuickTest** engine. The **WScript** object direct properties and methods are not supported also you can't create the object **WScript**.

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. Taken from the **QuickTest/VBScript** help file in **Help->QuickTest Professional Help.**

As I said before, **QuickTest** already embeds the **WScript** object. That's why we can't create a **WScript** object and not use is properties and methods, the help can be very confusing for new users, when using the **vbs** samples.

WScript Properties and Methods

WScript.Application Property

Not supported in **QuickTest**.

WScript.Arguments Property

Not supported in **QuickTest**. Use instead the **Parameter** or **TestArgs** object

WScript.FullName Property

Not supported in **QuickTest**.

WScript.Name Property

Not supported in **QuickTest**. Use instead **ScriptEngine** Function

WScript.Path Property

Not supported in **QuickTest**, add the following code to an associated external vbs file.

```
arr = Split(Environment("TestDir"), "\")
ReDim Preserve arr(UBound(arr) - 1)
Environment("TestPath") = Join(arr,"\")
```

WScript.ScriptFullName Property

Not supported in **QuickTest**, use instead Environment("TestDir")

WScript.ScriptName Property

Not supported in **QuickTest**, use instead Environment("TestName")

WScript.StdErr, StdIn, StdOut Properties

Not supported in **QuickTest**

WScript.Version Property

Not supported in **QuickTest**. Use instead a concatenation of

- **ScriptEngineMajorVersion** Function
- **ScriptEngineMinorVersion** Function

WScript.CreateObject Method

Not supported in **QuickTest**. Use instead **CreateObject** Function

WScript.ConnectObject Method

Not supported in **QuickTest**. Use instead **ConnectObject** Function

WScript.DisconnectObject Method

Not supported in **QuickTest**.

WScript.Echo Method

Not supported in **QuickTest**. Use instead **MsgBox** Function

WScript.GetObject Method

Not supported in **QuickTest**. Use instead **GetObject** Function

WScript.Quit Method

Not supported in **QuickTest**. Use instead **ExitTest** or **ExitRun** statements

WScript.ShowUsage Method

Not supported in **QuickTest**.

WScript.Sleep Method

Not supported in **QuickTest**. Use instead the **Wait** statement

WshArguments Collection Object

Not supported in **QuickTest**.

WshNamed Object

Not supported in **QuickTest**.

WshUnNamed Object

Not supported in **QuickTest**.

WshShell Object

The **WshShell** object is a **COM** object, and using the following code statement can create an instance of the object

The **WshShell** object provides access to a wide variety of shell services, such as registry access, access to environment variables and to the location of system folders, and the ability to create shortcuts and to start processes.

You can instantiate a **WshShell** object with a code fragment like the following:

```
Dim oShell
Set oShell = CreateObject("WScript.Shell")
```

WshShell Capabilities

Category	Method or Property
Running Programs	Run, Exec
Working with Special Folders	SpecialFolders
Working with Shortcuts	CreateShortcut
Working with Environment Variables	Environment, ExpandEnvironmentStrings
Working with the Event Log	LogEvent
Working with the Registry	RegRead, RegWrite, RegDelete
Sending Keystrokes to an Application	AppActivate, SendKeys
Obtaining a Scripts Current Directory	CurrentDirectory
Creating Timed Dialog Boxes	Popup

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).

Running Programs

One of the most important lessons to learn about running advanced scripting in QuickTest s this: Advanced scripting is not the answer to all your system administration level tests.

For one thing, scripting does not provide 100 percent coverage for all the administrative tasks that can be performed under Windows. For example, you can use scripts to create shared folders. But what if you want to set the offline folder options for that share? This procedure cannot be scripted using **WSH** or **WMI**. Instead, you must configure offline folder options using either the **GUI** or the Net.exe command.

For another, there is no reason to write a script if a tool already exists that fills your needs. Suppose you want to see the list of files that are stored in a folder on the local computer. You can spend the time creating a script to return this

information. Alternatively, you can simply type **dir** at the command prompt and be done with it.

Using **WSH** scripts to help automate tasks does not require that you abandon the command-line tools

Although running command-line tools and running batch scripts are important uses of the **Run** and **Exec** methods, these methods are not restricted to running a particular type of program. The **Run** and **Exec** methods enable your scripts to run any Windows program, including **GUI** applications. In addition, you can also run other scripts from within a script.

Working with Shortcuts

Shortcuts are links to local or network programs, files, folders, computers, or Internet addresses. Each shortcut is a file with either an .lnk or a .url extension. The two extensions correspond to the two types of shortcuts: standard shortcuts and Uniform Resource Locator (URL) shortcuts. Standard shortcuts can link to local or network programs, files, folders, or computers, while URL shortcuts link only to entities that can be referenced by a URL most commonly, Web pages.

Shortcuts are used in a number of places within the Windows shell, particularly within menus and toolbars. The Start menu, the Programs menu, the Quick Launch bar, and the **SendTo** menu, for example, consist of a group of shortcuts located in special folders. The shortcuts that appear on the Quick Launch bar are located in the following folder:

C:\Documents and Settings\{user profile name}\Application Data\Microsoft\Internet Explorer\Quick Launch

Working with Special Folders

Special folders are folders that are or at least potentially can be present on all Windows computers; these include the My Documents, Fonts, and Start Menu folders. There are two types of special folders: those that map to standard directories and those that do not. The Favorites folder, for example, maps to a standard directory; the My Computer folder does not.

The **WScript SpecialFolders** collection contains the full path to each of the special folders that map to a standard directory.

The **SpecialFolders** collection enables your script to determine the path of any special folder that maps to a standard directory but does not enable your script to manipulate that folder or its contents. You must use a mechanism such as the **FileSystemObject** to actually work with the contents of the special folders.

Working with Environment Variables

Environment variables are a set of string values associated with a process. The Windows Shell process has a number of environment variables associated with it that contain useful information that you can use within your scripts, including:

- Directories searched by the shell to locate programs (the path).
- Number of processors, processor manufacturer, and processor architecture of the computer.
- User profile location.
- Temporary directory locations.

When a user logs on to Windows, the shell process starts and obtains its initial environment variables by loading both the computer-specific (system) and user-specific (user) environment variables from the registry.

In addition to the computer-specific and user-specific environment variables loaded from the registry, additional process environment variables are generated dynamically during each logon.

Logging an Event

Troubleshooting applications and services is simplified if these applications and services log important events to event logs. Your scripts will also be easier to troubleshoot if they do the same.

The **WshShell** object provides the **LogEvent** method for logging events to the Application event log. The **LogEvent** method enables you to write to the event log from within your scripts.

If you want to read and process event log information from within your scripts, you need to use **WMI**.

Reading From and Writing to the Local Registry

The **WshShell** object provides methods for reading from, writing to, and deleting from the registry.

Changing the registry with a script can easily propagate errors. The scripting tools bypass safeguards, allowing settings that can damage your system, or even require you to reinstall Windows. Before scripting changes to the registry, test your script thoroughly and back up the registry on every computer on which you will make changes.

The registry is the primary configuration database for the Windows operating system; the ability of an operating system component to run, and to run correctly, often depends on the configuration of one or more settings within the registry.

Sending Keystrokes to a Program

By providing scripts with access to most **COM** objects, **WSH** enables you to automate applications that have a **COM**-based object model. Unfortunately, some applications, especially older ones, do not have a **COM**-based object model. To automate these applications, **WSH** provides a way to send keystrokes to these applications.

When you use the **WshShell SendKeys** method to send keystrokes to an application, your script mimics a human typing on the keyboard. To send a single

keyboard character, you pass **SendKeys** the character itself as a string argument. For example, "**x**" to send the letter **x**. To send a space, send the string " ". This is exactly what a user would do if he or she was working with the application: to type the letter **x**, the user would simply press the **x** key on the keyboard.

Although **SendKeys** can be used effectively, there are several potential problems with this approach:

- The script might have difficulty determining which window to send the keystrokes to.
- Because the application runs in **GUI** mode, a user might close the application prematurely. Unfortunately, this will not terminate the script, and the script could end up sending keystrokes to the wrong application.
- The script might have difficulty synchronizing with the application.
- This timing issue is especially troublesome, simply because scripts tend to run much faster than **GUI** applications.

WshShell Properties and Methods

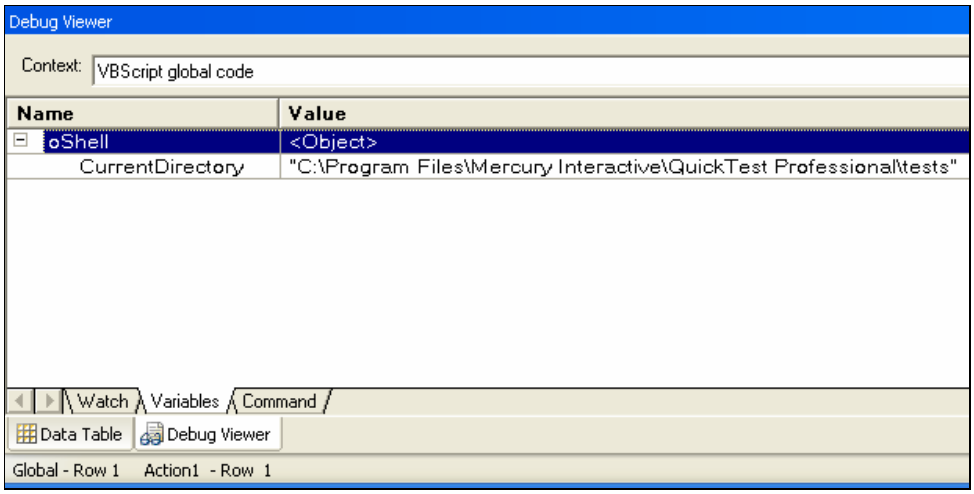

Figure 3 - Wshell Object View

WshShell.CurrentDirectory Property

Description

The **CurrentDirectory** property retrieves or changes the current active dir.

Data Type

String

Example

The following code displays the current active directory.

```
Option Explicit
Dim oShell
Set oShell = CreateObject("WScript.Shell")
MsgBox oShell.CurrentDirectory
```

```
Set oShell = Nothing
```

WshShell.Environment Property

Control Panel->System->Advanced->Environment Variables

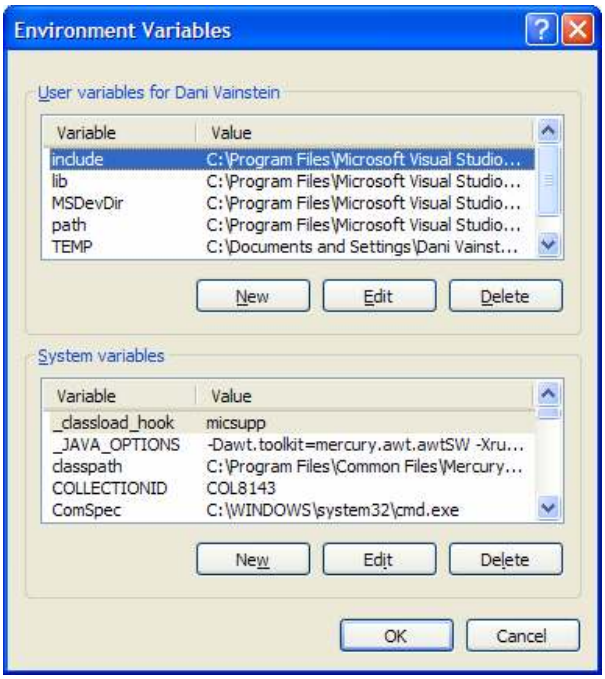

Figure 4 - Windows XP Environment Variable Dialog

Description

The **Environment** property returns the **WshEnvironment** object (a collection of environment variables).

Syntax

```
object.Environment(strType)
```

Data Type

String

Arguments

Parameter	Description
strType	Optional. Specifies the location of the environment variable.

Settings

Type of Env. Variable	Description
System	Applies to all users of the computer and is saved between logoffs and restarts. Stored at *HKLM\System\CurrentControlSet\Control\Session*

	Manager\Environment
User	Applies to the user currently logged on to the computer and is saved between logoffs and restarts. Stored at *HKCU\Environment*
Volatile	Applies to current logon session and is not saved between logoffs and restarts. Stored at *HKCU\VolatileEnvironment*
Process	Applies to current process and might be passed to child processes. Not stored in registry

Example

The following code displays all the environment variables in your system to the **DataTable.LocalSheet**.

```
Option Explicit
Dim oShell, oEnvProc, oEnvSys, oItem
Dim arrEnv
Dim nRow : nRow = 1
'--- Creating 2 new columns in local sheet
DataTable.LocalSheet.AddParameter "Type", vbNullString
DataTable.LocalSheet.AddParameter "Variable", vbNullString
DataTable.LocalSheet.AddParameter "Value", vbNullString
'--- Creating a WshShell object
Set oShell = CreateObject("WScript.Shell")
'--- Retrieve the System environments collection
Set oEnvSys = oShell.Environment("System")
'--- Retrieve the Process/User environments collection
Set oEnvProc = oShell.Environment("Process")
Reporter.ReportEvent micDone, "Sys Count", oEnvSys.Count
Reporter.ReportEvent micDone, "Proc Count", oEnvProc.Count
'--- Retrieve System env. variables
For Each oItem In oEnvSys
      DataTable.LocalSheet.SetCurrentRow nRow
      nRow = nRow + 1
      arrEnv = Split(oItem, "=")
      DataTable("Type", dtLocalSheet) = "System"
      DataTable("Variable", dtLocalSheet) = arrEnv(0)
      DataTable("Value", dtLocalSheet) = arrEnv(1)
      Erase arrEnv
Next
'--- Retrieve System env. variables
For Each oItem In oEnvProc
      DataTable.LocalSheet.SetCurrentRow nRow
      nRow = nRow + 1
      arrEnv = Split(oItem, "=")
      DataTable("Type", dtLocalSheet) = "Process"
      DataTable("Variable", dtLocalSheet) = arrEnv(0)
      DataTable("Value", dtLocalSheet) = arrEnv(1)
      Erase arrEnv
Next
Set oShell = Nothing
Set oEnvProc = Nothing : Set oEnvSys = Nothing
```

Run-Time Data Table

	Type	Variable	Value
1	System	ComSpec	%SystemRoot%\system32\cmd.exe
2	System	Path	%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem
3	System	windir	%SystemRoot%
4	System	FP_NO_HOST_CHECK	NO
5	System	OS	Windows_NT
6	System	PROCESSOR_ARCHITECTURE	x86
7	System	PROCESSOR_LEVEL	15
8	System	PROCESSOR_IDENTIFIER	x86 Family 15 Model 4 Stepping 3, GenuineIntel
9	System	PROCESSOR_REVISION	0403
10	System	NUMBER_OF_PROCESSORS	2
11	System	PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
12	System	TEMP	%SystemRoot%\TEMP
13	System	TMP	%SystemRoot%\TEMP
14	System	LSERVRC	C:\Program Files\Common Files\Mercury Interactive\License Manager\lservrc
15	System	INCLUDE	C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\include\
16	System	LIB	C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Lib\;C:\Program Files\Common Files
17	System	VS71COMNTOOLS	C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools\
18	Process	ALLUSERSPROFILE	C:\Documents and Settings\All Users
19	Process	APPDATA	C:\Documents and Settings\Daniel\Application Data
20	Process	CLASSPATH	C:\PROGRA~1\MERCUR~1\QUICKT~1\bin\;
21	Process	CommonProgramFiles	C:\Program Files\Common Files
22	Process	COMPUTERNAME	DANIVA
23	Process	ComSpec	C:\WINDOWS\system32\cmd.exe
24	Process	FP_NO_HOST_CHECK	NO
25	Process	HOMEDRIVE	C:
26	Process	HOMEPATH	\Documents and Settings\Daniel
27	Process	INCLUDE	C:\Program Files\Microsoft Visual Studio\VC98\atl\include;C:\Program Files\Microsoft Visual Studio
28	Process	LIB	C:\Program Files\Microsoft Visual Studio\VC98\mfc\lib;C:\Program Files\Microsoft Visual Studio\VC

Figure 5 – WshShell Environment

The following code retrieves the system environment variable NUMBER_OF_PROCESSORS.

```
Option Explicit
Dim oShell, oEnvColl
Set oShell = CreateObject("WScript.Shell")
Set oEnvColl = oShell.Environment("SYSTEM")
MsgBox oEnvColl("NUMBER_OF_PROCESSORS")
'--- Clear objects
Set oEnvColl = Nothing : Set oShell = Nothing
```

WshShell.SpecialFolders Property

Description

The **Environment** property returns a **SpecialFolders** object

Data Type

SpecialFolders collection object

Notes

- AllUsersDesktop
- AllUsersStartMenu
- AllUsersPrograms
- AllUsersStartup
- Desktop
- PrintHood
- Programs
- Recent
- SendTo
- Fonts
- MyDocuments
- NetHood
- StartMenu
- Startup
- Templates
- Favorites

Example

The following code enumerates all the special folders locations in your system

```
Option Explicit
Dim oShell, oSpecialFolders
Dim i
DataTable.LocalSheet.AddParameter "Location", vbNullString
```

```
'--- Creating a WshShell object
Set oShell = CreateObject("WScript.Shell")
'--- Retrieve the special folders collection
Set oSpecialFolders = oShell.SpecialFolders
'--- Counting items and reporting
Reporter.ReportEvent micDone, "Folders Count", oSpecialFolders.Count
'--- Looping thorough the collection
For i = 0 To oSpecialFolders.Count - 1
    DataTable.LocalSheet.SetCurrentRow i + 1
    DataTable("Location", dtLocalSheet) = oSpecialFolders.Item(i)
Next
Set oSpecialFolders = Nothing : Set oshell = Nothing
```

Run-Time Data Table

	Location
1	C:\Documents and Settings\All Users\Desktop
2	C:\Documents and Settings\All Users\Start Menu
3	C:\Documents and Settings\All Users\Start Menu\Programs
4	C:\Documents and Settings\All Users\Start Menu\Programs\Startup
5	C:\Documents and Settings\Daniel\Desktop
6	C:\Documents and Settings\Daniel\Application Data
7	C:\Documents and Settings\Daniel\PrintHood
8	C:\Documents and Settings\Daniel\Templates
9	C:\WINDOWS\Fonts
10	C:\Documents and Settings\Daniel\NetHood
11	C:\Documents and Settings\Daniel\Desktop
12	C:\Documents and Settings\Daniel\Start Menu
13	C:\Documents and Settings\Daniel\SendTo
14	C:\Documents and Settings\Daniel\Recent
15	C:\Documents and Settings\Daniel\Start Menu\Programs\Startup
16	C:\Documents and Settings\Daniel\Favorites
17	C:\Documents and Settings\Daniel\My Documents
18	C:\Documents and Settings\Daniel\Start Menu\Programs

Figure 6 - WshShell SpecialFolders

WshShell.AppActivate Method

Description

The **AppActivate** activates an application.

Syntax

```
object.AppActivate title
```

Arguments

Parameter	Description
title	Specifies which application to activate. This can be a string containing the title of the application (as it appears in the title bar) or the application's **Process ID**.

Return Value

Boolean

Notes

- The **AppActivate** method returns a **Boolean** value that identifies whether the procedure call is successful. This method changes the focus to the named application or window, but it does not affect whether it is maximized or minimized. Focus moves from the activated application window when the user takes action to change the focus (or closes the window).
- In determining which application to activate, the specified title is compared to the title string of each running application. If no exact match exists, any application whose title string begins with title is activated. If an application still cannot be found, any application whose title string ends with title is activated. If more than one instance of the application named by title exists, one instance is arbitrarily activated.

Example

The following code activates notepad and activates the **SendKey** method with intervals

```
Option Explicit
Dim oShell
Dim bActivate, bExec
Dim arrKeys
Dim i
'--- defining the keys array
arrKeys = Array("Scripting ", _
                "Q","u","i","c","k ","T","e","s","t ", _
                "P","r","o","f", "e", "s", "s", "i", "o", "nal")
Set oShell = CreateObject("WScript.Shell")
'--- Looping until notepad is activated
Do
      bActivate = oShell.AppActivate("Notepad")
      If Not bActivate Then
          bExec = oShell.Run("notepad.exe")
          If bExec = 0 Then Wait 1
      End If
Loop Until bActivate
'--- sending keys with 100 milliseconds interval
For i = 0 To UBound(arrKeys)
      Wait 0,100
      oShell.SendKeys arrKeys(i)
Next
Set oShell = Nothing
```

WshShell.CreateShortcut Method

Description

The **CreateShortcut** method creates a new shortcut, or opens an existing shortcut.

Syntax

```
object.CreateShortcut (sPathname)
```

Arguments

Parameter	Description
sPathname	String value indicating the pathname of the shortcut to create.

Return Value

WshShortcut (.lnk) object or a **WshURLShortcut** (.url) object depends on the file extension supplied with the *sPathname* argument.

Notes

- The **CreateShortcut** method returns either a **WshShortcut** object or a **WshURLShortcut** object.
- If *sPathName* points to a file that already exists, then the shortcut file is opened and can be modified via the shortcut object returned by **CreateShortcut**. Otherwise, the shortcut file will not be created until the **Save** method of the shortcut object is called.
- Any other file extension than (.lnk or .url) generates an automation error.
- Simply calling the **CreateShortcut** method does not result in the creation of a shortcut. The shortcut object and changes you may have made to it are stored in memory until you save it to disk with the **Save** method. To create a shortcut, you must:
  - Create an instance of a **WshShortcut** object.
  - Initialize its properties.
  - Save it to disk with the **Save** method.

Example

The following code displays all the My documents path in two different ways

```
Option Explicit

Dim oShell, oShLink
Dim sPath, sDesktop
Dim oFso
Set oFso = CreateObject("Scripting.FileSystemObject")
'--- Creating a WshShell object
Set oShell = CreateObject("WScript.Shell")
'--- Retrive the user desktop location
sDeskTop = oShell.SpecialFolders.Item("Desktop")
sPath = oFso.BuildPath(sDesktop, "Shortcut to QuickTest.lnk")
'--- Create a WshShortcut object
Set oShLink = oShell.CreateShortcut(sPath)
sPath = oFso.BuildPath(Environment("ProductDir"), "bin\QTPro.exe")
oShLink.TargetPath = sPath
oShLink.WindowStyle = 1
oShLink.Hotkey = "CTRL+SHIFT+Q"
sPath = oFso.BuildPath(Environment("ProductDir"), "bin\QTPro.exe")
oShLink.IconLocation = sPath & ", 0"
oShLink.Description = "Activates QuickTest Professional Testing Tool"
oShLink.WorkingDirectory= oFso.BuildPath(Environment("ProductDir"), "bin\")
oShLink.Save

Set oShell = Nothing : Set fso = Nothing
Set oShLink = Nothing
```

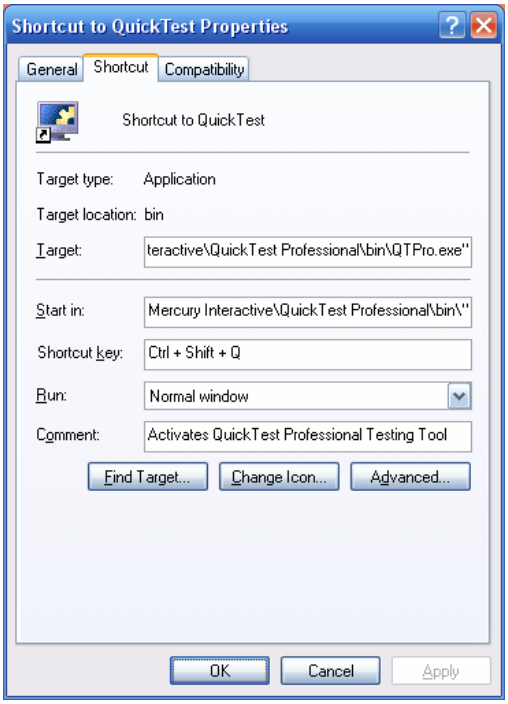

Figure 7 – Created Shortcut

WshShell.ExpandEnvironmentStrings Method

Description

The **ExpandEnvironmentStrings** method returns an environment variable's expanded value.

Syntax

```
object.ExpandEnvironmentString (strString)
```

Arguments

Parameter	Description
strString	String value indicating the name of the environment variable you want to expand.

Return Value

String

Notes

- The **ExpandEnvironmentStrings** method expands environment variables defined in the PROCESS environment space only. Environment variable names, which must be enclosed between "%" characters, are not case-sensitive.

Example

The following code displays the expanded environment variables.

```
Option Explicit
Dim oShell
'--- Creating a WshShell object
Set oShell = CreateObject("WScript.Shell")
MsgBox oShell.ExpandEnvironmentStrings("%SystemRoot%")
MsgBox oShell.ExpandEnvironmentStrings("%WinDir%")
MsgBox oShell.ExpandEnvironmentStrings("%ComputerName%")
MsgBox oShell.ExpandEnvironmentStrings("%UserName%")
MsgBox oShell.ExpandEnvironmentStrings("%ProgramFiles%")
```

WshShell.LogEvent Method

Description

The **LogEvent** method Adds an event entry to a log file.

Syntax

```
object.LogEvent(nType, sMessage ,sTarget)
```

Arguments

Parameter	Description
nType	Required. Integer value representing the event type.
sMessage	Required. String value containing the log entry text.
sTarget	Optional. String value indicating the name of the computer system where the event log is stored (the default is the local computer system). Applies to Windows NT/2000 only.

Return Value

Boolean

Notes

- The **LogEvent** method returns a **Boolean** value (true if the event is logged successfully, otherwise false). In Windows NT/2000, events are logged in the Windows NT Event Log.
- In Windows 9x/Me, events are logged in WSH.log (located in the Windows directory). There are six event types.

Type	Value		Type	Value
0	SUCCESS		4	INFORMATION
1	ERROR		8	AUDIT_SUCCESS
2	WARNING		16	AUDIT_FAILURE

Example

The following code logs the events of **LogEvent** Method

```
Option Explicit
'--- Declaring Event types constants
Private Const LOG_SUCCESS = 0
Private Const LOG_ERROR = 1
Private Const LOG_WARNING = 2
Private Const LOG_INFORMATION = 4
Private Const LOG_AUDIT_SUCCESS = 8
Private Const LOG_AUDIT_FAILURE = 16

Dim oShell
'--- Creating a shell object
Set oShell = CreateObject("WScript.Shell")
oShell.LogEvent LOG_SUCCESS, "Logon Script Success."
oShell.LogEvent LOG_ERROR, "Logon Script Error."
oShell.LogEvent LOG_WARNING, "Logon Script Warning."
oShell.LogEvent LOG_INFORMATION, "Logon Script Information."
oShell.LogEvent LOG_AUDIT_SUCCESS, "Logon Script Audit Success."
oShell.LogEvent LOG_AUDIT_FAILURE, "Logon Script Audit Failure."
'--- Cleaning used objects
Set oShell = Nothing
```

To see the log Event Viewer in WinXP:
Control Panel->Administrative Tools->Event Viewer

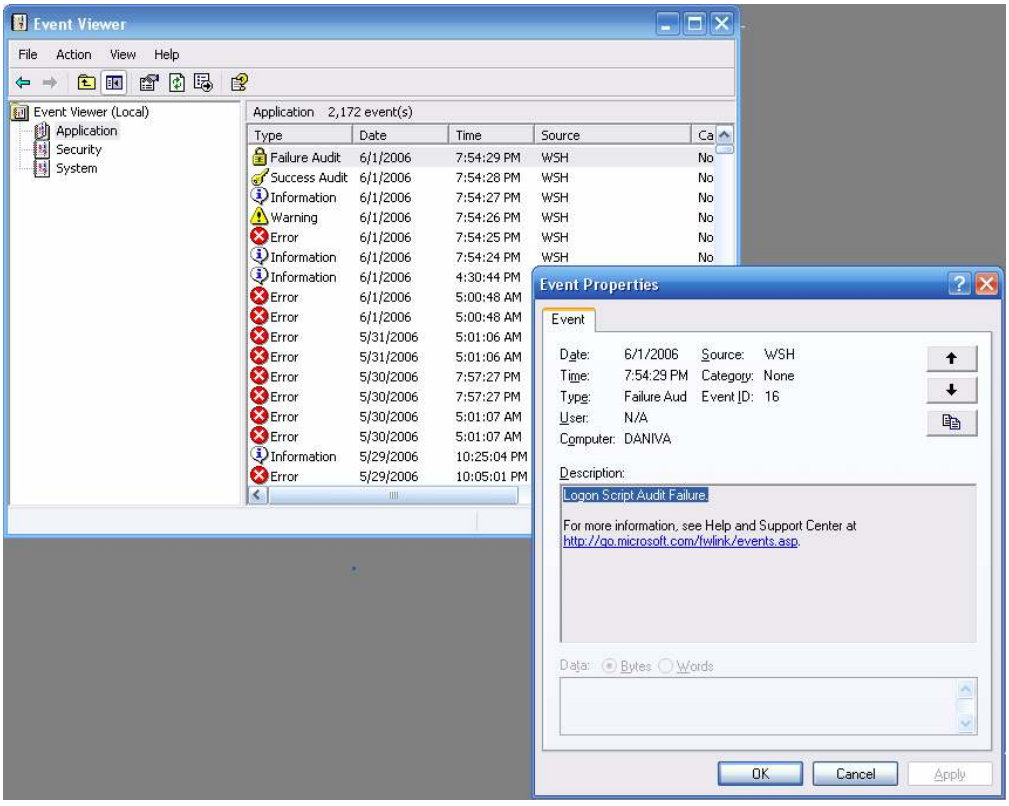

Figure 8 - Log Event Viewer

WshShell.Popup Method

Description

The **Popup** method Displays a timed text in a pop-up message box.

Syntax

```
nButton = object.Popup(sText, nSecondsToWait, sTitle, nType)
```

Arguments

Parameter	Description
sText	String value containing the text you want to appear in the pop-up message box.
nSecondsToWait	Optional. Numeric value indicating the maximum length of time (in seconds) you want the pop-up message box displayed.
sTitle	Optional. String value containing the text you want to appear as the title of the pop-up message box.
nType	Optional. Numeric value indicating the type of buttons and icons you want in the pop-up message box. These determine how the message box is used.
nButton	Integer value indicating the number of the button the user clicked to dismiss the message box. This is the value returned by the **Popup** method.

Return Value

Integer

Notes

- If *nSecondsToWait* is equals zero (the default), the pop-up message box remains visible until closed by the user.
- If *nSecondsToWaitis* is greater than zero, the pop-up message box closes after *nSecondsToWait* seconds.
- If you do not supply the argument *strTitle*, the title of the pop-up message box defaults to "Windows Script Host."
- The meaning of *nType* is the same as in the Microsoft Win32® application programming interface **MsgBox** function.

Tip

- To display text properly in RTL languages such as Hebrew or Arabic, add hex &h00100000 (decimal 1048576) to the *nType* parameter.

WshShell.RegDelete Method

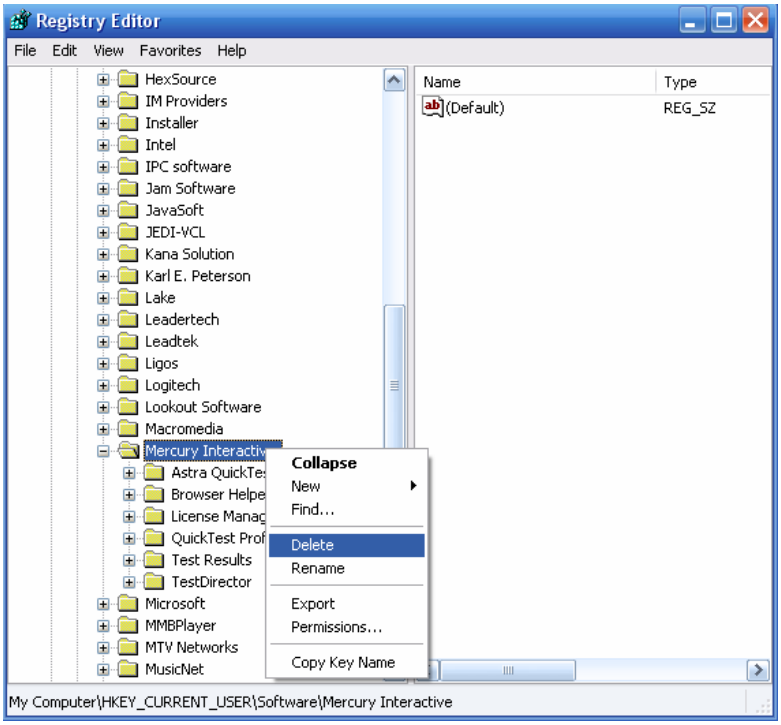

Figure 9 - Registry Delete key

Description

The **RegDelete** method deletes a key or one of its values from the registry.

Syntax

```
object.RegDelete(sName)
```

Arguments

Parameter	Description
sName	Required. String value indicating the name of the registry key or key value you want to delete.

Notes

- Specify a key-name by ending *strName* with a final backslash; leave it off to specify a value-name. Fully qualified key-names and value-names are prefixed with a root key.
- You may use abbreviated versions of root key names with the **RegDelete** method.

Settings – Root keys abbreviations

Root key name	Abbreviation
HKEY_CURRENT_USER	HKCU
HKEY_LOCAL_MACHINE	HKCU

HKEY_CLASSES_ROOT	HKCR
HKEY_USERS	HKEY_USERS
HKEY_CURRENT_CONFIG	HKEY_CURRENT_CONFIG

Tip

- If *sName* ends in a backslash, it denotes a key; otherwise, it denotes a value. The default (or unnamed) value of a key cannot be deleted; it must be replaced with an empty string ("") by using the **RegRead** method.

WshShell.RegRead Method

Description

The **RegRead** method returns the value of a key or value-name from the registry.

Syntax

```
object.RegRead(sName)
```

Arguments

Parameter	Description
sName	Required. String value indicating the key or value-name whose value you want.

Return Value

The **RegRead** method returns values of the following five types.

Type	Description	In The Form of
REG_SZ	A string	A string
REG_DWORD	A Number	An Integer
REG_BINARY	A binary value	A **VBArray** of integers
REG_EXPAND_SZ	An expandable string	A string
REG_MULTI_SZ	An array of strings	A **VBArray** of strings

Notes

- You can specify a key-name by ending *sName* with a final backslash.
- Do not include a final backslash to specify a value-name.
- A value entry has three parts: its name, its data type, and its value. When you specify a key-name (as opposed to a value-name),
- **RegRead** returns the default value. To read a key's default value, specify the name of the key itself.
- If your script attempts to use the **RegRead** method to retrieve the value of a registry entry with an unsupported data type, the call will result in an error.
- Unfortunately, **WSH** does not provide a way for you to verify the data type of a registry entry before you attempt to read it. However, you can use **WMI**

to verify data types.

WshShell.RegWrite Method

Description

The **RegWrite** method creates a new key, adds another value-name to an existing key, or changes the value of an existing value-name.

Syntax

```
object.RegWrite(sName, vValue [,sType])
```

Arguments

Parameter	Description
sName	Required. String value indicating the key-name, value-name, or value you want to create, add, or change.
vValue	Required. The name of the new key you want to create, the name of the value you want to add to an existing key, or the new value you want to assign to an existing value-name.
sType	Optional. String value indicating the value's data type.

Notes

- Specify a key-name by ending *sName* with a final backslash.
- Do not include a final backslash to specify a value name.
- The **RegWrite** method automatically converts the parameter *vValue* to either a string or an integer.
- The value of *sType* determines its data type (either a string or an integer). The options for *sType* are listed in the following table.

Converted to	sType
String	REG_SZ
String	REG_EXPAND_SZ
Integer	REG_DWORD
Integer	REG_BINARY

Notes

- **RegWrite** will write at most one **DWORD** to a **REG_BINARY** value. Larger values are not supported with this method.
- The **REG_MULTI_SZ** type is not supported for the **RegWrite** method.

Example

The following code creates a key and two values, reads them, and deletes them.

```
Option Explicit
Dim oShell
```

```
Dim sDefault, sBinary, sString, sDWord, sExpand
'--- Creating a shell object
Set oShell = CreateObject("WScript.Shell")
'--- Creating registry values
oShell.RegWrite "HKCU\Software\QTP\", "Default Key", "REG_SZ"
oShell.RegWrite "HKCU\Software\QTP\Binary", 180150008, "REG_BINARY"
oShell.RegWrite "HKCU\Software\QTP\String", "String", "REG_SZ"
oShell.RegWrite "HKCU\Software\QTP\DWord", &HABCDEF8, "REG_DWORD"
oShell.RegWrite "HKCU\Software\QTP\ExpandedString", "%SystemRoot%", "REG_EXPAND_SZ"
'--- Reading registry values
sDefault = oShell.RegRead("HKCU\Software\QTP\")
sBinary = oShell.RegRead("HKCU\Software\QTP\Binary")
sString = oShell.RegRead("HKCU\Software\QTP\String")
sDWord = oShell.RegRead("HKCU\Software\QTP\DWord")
sExpand = oShell.RegRead("HKCU\Software\QTP\ExpandedString")
'--- Deleting a key value
oShell.RegDelete "HKCU\Software\QTP\String"
'--- Deleting a whole key
oShell.RegDelete "HKCU\Software\QTP\"
Set oShell = Nothing
```

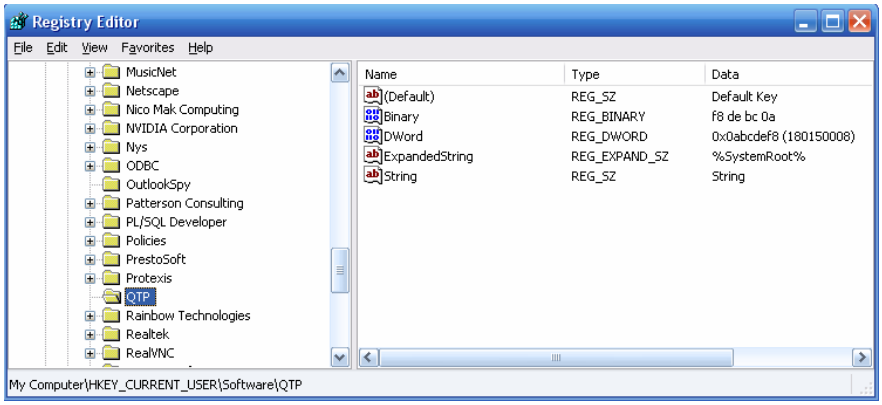

Figure 10 - RedWrite, RegRead and RegDelete methods

This is how the variables values looks in the debug viewer after reading values and before remove the registry keys.

Name	Value
oShell	<Object>
CurrentDirectory	"C:\Program Files\Mercury Interactive\QuickTest Professional\tests"
sDefault	"Default Key"
sBinary	<Array>
(0)	248
(1)	222
(2)	188
(3)	10
sString	"String"
sDWord	180150008
sExpand	"%SystemRoot%"

Figure 11 - RegRead View

WshShell.Run Method

Description

The **Run** method runs a program in a new process.

Syntax

```
object.Run(sCommand, [nWindowStyle], [bWaitOnReturn])
```

Arguments

Parameter	Description
sCommand	Required. String value indicating the command line you want to run. You must include any parameters you want to pass to the executable file.
nWindowStyle	Optional. Integer value indicating the appearance of the program's window. Note that not all programs make use of this information.
bWaitOnReturn	Optional. Boolean value indicating whether the script should wait for the program to finish executing before continuing to the next statement in your script. If set to **true**, script execution halts until the program finishes, and **Run** returns any error code returned by the program. If set to **false** (the default), the **Run** method returns immediately after starting the program, automatically returning 0 (not to be interpreted as an error code).

Return Value

Integer

Notes

- The **Run** method starts a program running in a new Windows process.
- The Run method is very similar to the **QuickTest SystemUtil.Run** method
- Do You can have your script wait for the program to finish execution before continuing. This allows you to run scripts and programs synchronously.
- Environment variables within the argument *sCommand* are automatically expanded.
- If a file type has been properly registered to a particular program, calling run on a file of that type executes the program. For example, if Word is installed on your computer system, calling **Run** on a *.doc file starts Word and loads the document.
- The following table lists the available settings for *nWindowStyle*.

Style	Description
0	Hides the window and activates another window.
1	Activates and displays a window. If the window is minimized or maximized, the system restores it to its original size and position. An application should specify this flag when displaying the window for the first time.
2	Activates the window and displays it as a minimized window.
3	Activates the window and displays it as a maximized window.
4	Displays a window in its most recent size and position. The active window remains active.
5	Activates the window and displays it in its current size and position.
6	Minimizes the specified window and activates the next top-level window in the Z

	order.
7	Displays the window as a minimized window. The active window remains active.
8	Displays the window in its current state. The active window remains active.
9	Activates and displays the window. If the window is minimized or maximized, the system restores it to its original size and position. An application should specify this flag when restoring a minimized window.
10	Sets the show-state based on the state of the program that started the application.

Example

The following code opens a command window, changes to the path to C:\ , and executes the **DIR** command.

```
Option Explicit
Dim oShell
Set oShell = CreateObject("WScript.shell")
'--- running a DOS command
oShell.Run "cmd /K CD C:\ & Dir /O"
Set oShell = Nothing
```

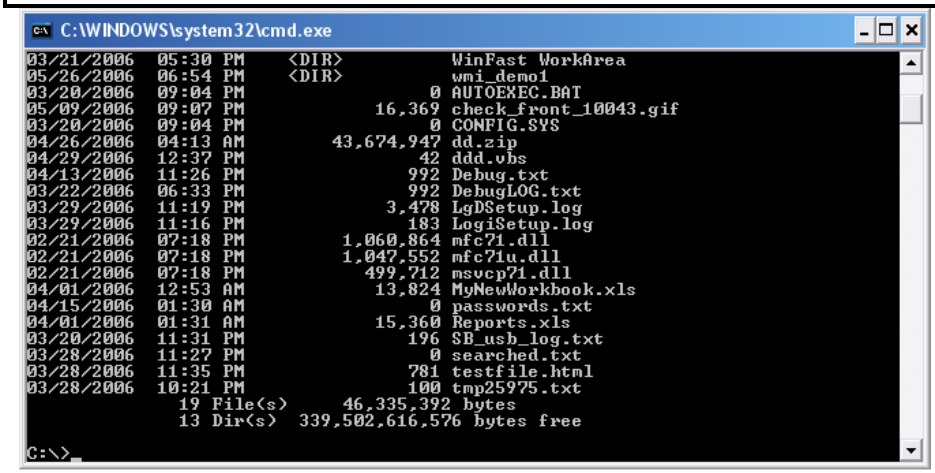

Figure 12 - Run method

Example 2

The following code opens a copy of the currently running script with Notepad it also specifies the window type, waits for Notepad to be shut down by the user, and saves the error code returned from Notepad when it is shut down.

```
Option Explicit
Dim oShell, oFso
Dim nRetval, sScriptPath
'--- Creating a shell object
Set oShell = CreateObject("WScript.Shell")
Set oFso = CreateObject("Scripting.FileSystemObject")
sScriptPath = oFso.BuildPath(Environment("TestDir"),"\Action1\Script.mts")
nRetval = oShell.Run("%windir%\notepad " & sScriptPath, 1, True  )
MsgBox "Notepad Closed, return code: " & nRetval
Set oFso = Nothing : Set oShell = Nothing
```

WshShell.SendKeys Method

Description

The **SendKeys** method sends one or more keystrokes to the active window (as if typed on the keyboard).

Syntax

```
object.SendKeys(string)
```

Arguments

Parameter	Description
string	String value indicating the keystroke(s) you want to send.

Notes

- Use the **SendKeys** method to send keystrokes to applications that have no automation interface.
- Most keyboard characters are represented by a single keystroke.
- Some keyboard characters are made up of combinations of keystrokes (CTRL+SHIFT+HOME, for example). To send a single keyboard character, send the character itself as the string argument.
- To send a space, send the string " ".
- You can use **SendKeys** to send more than one keystroke at a time. To do this, create a compound string argument that represents a sequence of keystrokes by appending each keystroke in the sequence to the one before it. For example, to send the keystrokes a, b, and c, you would send the string argument "abc".
- The **SendKeys** method uses some characters as modifiers of characters (instead of using their face-values).
  - plus sign "+"
  - caret "^"
  - percent "%"
  - tilde "~"
- Send these characters by enclosing them within braces "{}". For example, to send the plus sign, send the string argument "{+}". Brackets "[ ]" have no special meaning when used with SendKeys, but you must enclose them within braces to accommodate applications that do give them a special meaning (for dynamic data exchange (DDE) for example).
- To send bracket characters, send the string argument "{[}" for the left bracket and "{]}" for the right one.
- To send brace characters, send the string argument "{{}" for the left brace and "{}}" for the right one.
- Some keystrokes do not generate characters (such as ENTER and TAB). Some keystrokes represent actions (such as BACKSPACE and BREAK). To send these kinds of keystrokes, send the arguments shown in the following table:

Key	Argument	Key	Argument

BACKSPACE	{BACKSPACE}, {BS}, or {BKSP}	F1	{F1}
BREAK	{BREAK}	F2	{F2}
CAPS LOCK	{CAPSLOCK}	F3	{F3}
DEL or DELETE	{DELETE} or {DEL}	F4	{F4}
DOWN ARROW	{DOWN}	F5	{F5}
END	{END}	F6	{F6}
ENTER	{ENTER} or ~	F7	{F7}
ESC	{ESC}	F8	{F8}
HELP	{HELP}	F9	{F9}
HOME	{HOME}	F10	{F10}
INS or INSERT	{INSERT} or {INS}	F11	{F11}
LEFT ARROW	{LEFT}	F12	{F12}
NUM LOCK	{NUMLOCK}	F13	{F13}
PAGE DOWN	{PGDN}	F14	{F14}
PAGE UP	{PGUP}	F15	{F15}
PRINT SCREEN	{PRTSC}	F16	{F16}
RIGHT ARROW	{RIGHT}	TAB	{TAB}
SCROLL LOCK	{SCROLLLOCK}	UP ARROW	{UP}

To send keyboard characters that are comprised of a regular keystroke in combination with a SHIFT, CTRL, or ALT, create a compound string argument that represents the keystroke combination. You do this by preceding the regular keystroke with one or more of the following special characters:

Key	Special Character
SHIFT	+
CTRL	^
ALT	%

- When used this way, these special characters are not enclosed within a set of braces.
- To specify that a combination of SHIFT, CTRL, and ALT should be held down while several other keys are pressed, create a compound string argument with the modified keystrokes enclosed in parentheses. For example, to send the keystroke combination that specifies that the SHIFT key is held down while:
  - **e** and **c** are pressed, send the string argument "**+**(**ec**)".
  - **e** is pressed, followed by a lone **c** (with no SHIFT), send the string argument "**+ec**".
- You can use the **SendKeys** method to send a pattern of keystrokes that consists of a single keystroke pressed several times in a row. To do this, create a compound string argument that specifies the keystroke you want to repeat, followed by the number of times you want it repeated. You do this using a compound string argument of the form {keystroke number}. For example, to send the letter "x" ten times, you would send the string argument "{x 10}". Be sure to include a space between keystroke and

number.

- The only keystroke pattern you can send is the kind that is comprised of a single keystroke pressed several times. For example, you can send "x" ten times, but you cannot do the same for "Ctrl+x".
- You cannot send the PRINT SCREEN key {PRTSC} to an application.

Example

I Find that **SendKeys** method is very useful in **QuickTest**, I use it a lot, especially on "no Web applications".

I have experience that sometimes when selecting a menu-item, is not always activated.

i.e. `Window("Notepad").WinMenu("Menu").Select "File;New Ctrl+N"`

In other applications the menu is not recognized by **QuickTest**, in WEB usually the menu is a collection of Web-Elements.

Every Windows application should support Keyboard usage, so, why don't we use it?

In the following sample I will show you how to handle all the menus in notepad, using the **SendKeys** method, and implement all the menu in a one single line.

External vbs/qfl file.

```
Option Explicit
RegisterUserFunc "Window", "SelectMenuItem", "MySelectMenuItem"
Public Sub MySelectMenuItem( ByVal obj, ByVal sMenuItem)
     Dim oShell
     Dim sSendKey
     Select Case LCase(sMenuItem)
     Case "about notepad"      : sSendKey = "%ha"
     Case "help topics"        : sSendKey = "%hs"
     Case "status bar"         : sSendKey = "%vs"
     Case "font"               : sSendKey = "%of"
     Case "word wrap"          : sSendKey = "%ow"
     Case "time date"          : sSendKey = "{F5}"
     Case "select all"         : sSendKey = "^a"
     Case "go to"              : sSendKey = "^g"
     Case "replace"            : sSendKey = "^h"
     Case "find next"          : sSendKey = "{F3}"
     Case "find"               : sSendKey = "^f"
     Case "delete"             : sSendKey = "{DEL}"
     Case "paste"              : sSendKey = "^v"
     Case "copy"               : sSendKey = "^c"
     Case "cut"                : sSendKey = "^x"
     Case "undo"               : sSendKey = "^u"
     Case "exit"               : sSendKey = "%fx"
     Case "print"              : sSendKey = "^p"
     Case "page setup"         : sSendKey = "%fu"
     Case "save as"            : sSendKey = "%fa"
     Case "save"               : sSendKey = "^s"
     Case "open"               : sSendKey = "^o"
     Case "new"                : sSendKey = "^n"
     Case Else
          Reporter.ReportEvent micFail, "SelectMenuItem", _
```

```
                                "Menu item '" & sMenuItem & "' does not exist."
            Err.Raise VbObjectError + 1002, "SelectMenuItem",  _
                                "Menu item '" & sMenuItem & "' does not exist."
            Exit Sub
        End Select
        Set oShell = CreateObject("WScript.Shell")
        obj.Activate : Wait 0, 200
        oShell.SendKeys sSendKey
        Set oShell = Nothing
End Sub
```

- After the function is ready, move the function to an external **vbs** file.
- Associate the **vbs** file to the test setting resources in **QuickTest**; Test->Settings->Resources->Add a new file to the list and select the **vbs** file you have just created.
- Of course we can call this function from our script using the follow syntax:

```
Call MySelectMenuItem(Window("Notepad", "New")
```

- But, I think, is more easy to use this function as a new user-defined method for the Window object.
- To create a new user-defined method we have to use the **QuickTest** statement

```
RegisterUserFunc "Window", "SelectMenuItem", "MySelectMenuItem"
```

To Use the example usage:

```
Dim oShell, oFso
Dim sPath
'--- Creating a shell object
Set oShell = CreateObject("WScript.Shell")
Set oFso = CreateObject("Scripting.FileSystemObject")
'--- Building the notepad file path and file
sPath = oFso.BuildPath(Environment("TestDir"),"SendKeys.txt")
oFso.CreateTextFile sPath, True
Set oFso = Nothing
'--- activating the new created text file
oShell.Run "%windir%\notepad " & sPath
Set oShell = Nothing
With Window("text:=SendKeys.txt - Notepad")
        .Activate
        .SelectMenuItem("Time Date")
        Wait 0,500
        .SelectMenuItem("Select All")
        Wait 0,500
        .SelectMenuItem("Copy")
        Wait 0,500
        .SelectMenuItem("Paste")
        Wait 0,500
        .SelectMenuItem("Paste")
        Wait 0,500
        .SelectMenuItem("Undo")
        Wait 0,500
        .SelectMenuItem("Save")
```

```
        Wait 0,500
        .SelectMenuItem("Exit")
End With
```

The advantage of save the ***RegisterUserFunc*** statement in the **vbs** file, instead that in the script itself, is that the method will auto-recognized by **QuickTest**.
Insert->Step->Step Generator->Category=Test Objects->Object=”Notepad”

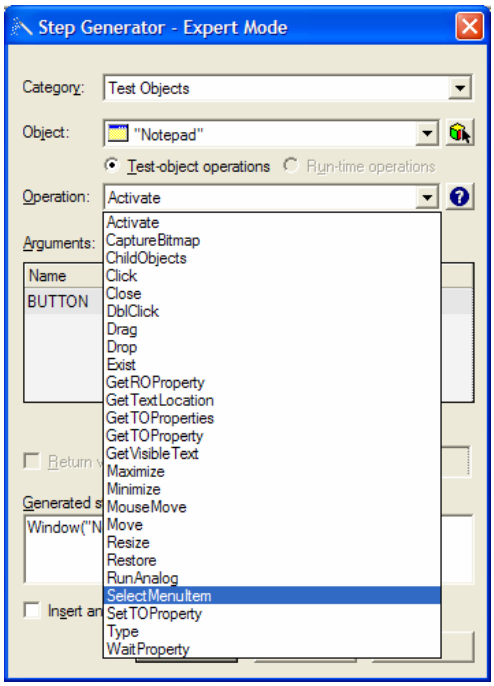

Figure 13 – Step generator

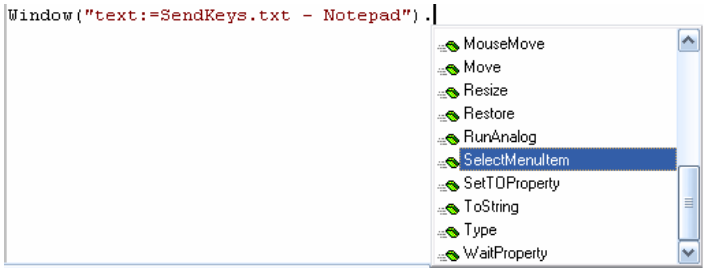

Figure 14 – QuickTest Intellicense

WshShell.Exec Method

Description

The **Exec** method Runs an application in a child command-shell, providing access to the **StdIn**/**StdOut**/**StdErr** streams.

Syntax

```
object.Exec(sCommand)
```

Arguments

Parameter	Description
sCommand	String value indicating the command line used to run the script. The command line should appear exactly as it would if you typed it at the command prompt.

Return Value

WshScriptExec object

Notes

- The **Exec** method returns a **WshScriptExec** object, which provides status and error information about a script run with **Exec** along with access to the **StdIn**, **StdOut**, and **StdErr** channels.
- The **Exec** method allows the execution of command line applications only.
- The **Exec** method cannot be used to run remote scripts. Do not confuse the **Exec** method with the **Execute** method (of the **WshRemote** object).

Comparing methods Exec and Run

The fact that there are two ways to run programs from a script leads to an obvious question: which method should you use in your scripts? The answer to that question depends on the script and what it needs to accomplish.

script can use either the **Run** method or the **Exec** method to run a program in a manner similar to using the Run dialog box from the Start menu. Regardless of the method used, the program starts, and runs in a new process.

However, when you use the **Run** method, your script will not have access to the standard input, output, and error streams generated by the program being run. A script cannot use the **Run** method to run a command-line tool and retrieve its output.

For example, suppose you want to run **Ping.exe** and then examine the output to see whether the computer could be successfully contacted. This cannot be done using the **Run** command. Instead, you would need to ping the computer, save the results of the ping command to a text file, open the text file, read the results, and then parse those results to determine the success or failure of the command.

The following script uses the **Run** method to call **Ping.exe**, redirecting the output to a temporary file. The script opens and reads the text file, checks to see whether the command succeeded (by determining whether any of the lines of output begin with the word Reply), and then closes and deletes the temporary file:

```
Option Explicit
Dim oShell, oFSO, oTextFile
Dim sTempName, sText
Dim sComputer : sComputer = "62.241.53.16"
'--- Creating a shell object
Set oShell = CreateObject("WScript.Shell")
Set oFSO = CreateObject("Scripting.FileSystemObject")
sTempName = oFSO.GetTempName
oShell.Run "%comspec% /c ping -n 3 -w 1000 " & sComputer & " > " & sTempName, 0, True
'--- Reading from output file
Set oTextFile = oFSO.OpenTextFile(sTempName, 1)
Do While oTextFile.AtEndOfStream <> True
    sText = oTextFile.ReadLine
    If Instr(sText, "Reply") > 0 Then
        MsgBox "Reply received."
        Exit Do
    End If
Loop
'--- Closing and deleting temporary file
oTextFile.Close : oFSO.DeleteFile(sTempName)
'--- Cleaning used objects
Set oTextFile = Nothing : Set oFSO = Nothing
Set oShell = Nothing
```

Although this approach works, it is somewhat complicated. If you need access to command-line output, you should use the **Exec** method instead. The following script also parses the output generated by **Ping.exe**. However, it does so by using the **Exec** method and by directly reading the output. There is no need to create, open, read, and delete a temporary file, and the script is only 9 lines long, compared with the 15 lines required to perform this same task using the **Run** method:

```
Option explicit
Dim oShell, oExec
Dim sText
Dim sComputer : sComputer = "62.241.53.16"
'--- Creating a shell object
Set oShell = CreateObject("WScript.Shell")
Set oExec = oShell.Exec("cmd /c ping -n 3 -w 1000 " & sComputer)
'--- Reading from output stream
Do While Not oExec.StdOut.AtEndOfStream
    sText = oExec.StdOut.ReadLine()
    If Instr(sText, "Reply") > 0 Then
        MsgBox "Reply received."
        Exit Do
    End If
Loop
'--- Cleaning used objects
Set oShell = Nothing
```

- In many respects, this makes the **Exec** method a better choice than the **Run** method. However, the **Run** method is still useful in a number of situations:
  - You might want to run the application in a specified window type, such as a minimized window. **Exec** offers no control over window style; **Run** offers

more options.

- You might need to run a script on computers that do not have **WSH 5.6** installed. **Exec** is supported only on **WSH 5.6**.
- You might want to wait for the application being called to finish running before the script resumes. This can be done with either **Run** or **Exec** but requires less coding with **Run**.

WshShortcut Object

- Microsoft offers two technologies that the **Visual Basic** developer can use to create shortcuts. The first is the **Shell** Library, a collection of **COM** automation objects and standard C/C++ functions found in **Shell32.dll**. The library, however, is really geared toward the C/C++ developer.
- Much more accessible and straightforward is the **Windows Scripting Host Object Library** (wshom.dll), which in this case, merely wraps the **IShellLink** interface of Shell32.dll. Although **Windows Script Host** (**WSH**) is most commonly called by scripted languages, such as **VBScript** or JScript, most of its object model is also accessible from **Visual Basic**.
- **WSH** has the advantage of being a very "flat" object model; you don't have to navigate through an extensive hierarchy of objects to either instantiate or retrieve the object in which you're interested.
- The **WshShortcut** object represents a shortcut, that is, a link to a file or other resource on the local system or local network. A new or existing **WshShortcut** object is returned by the **CreateShortcut** method of the **WshShell** object.
- A **WshShortcut** object exists in memory only and not in the filesystem until it is saved by calling the object's **Save** method.

WshShortcut Properties and Methods

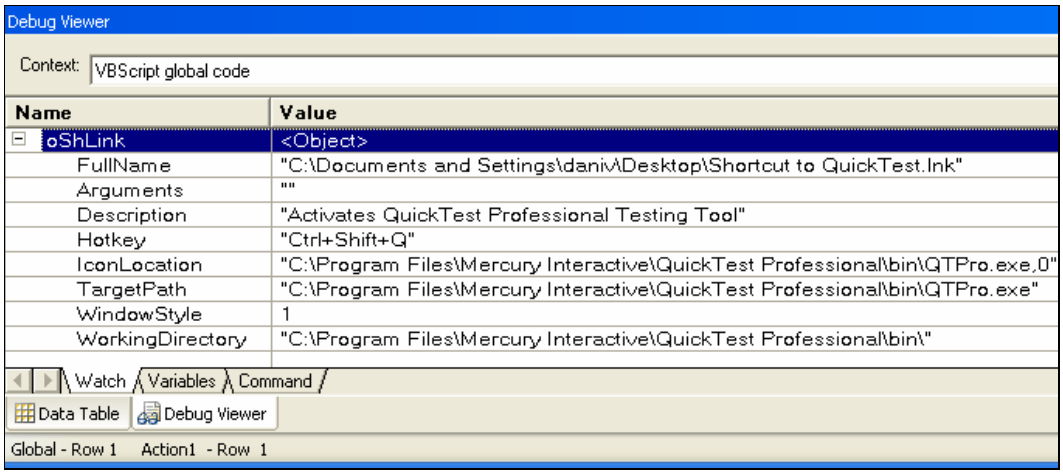

Figure 15 - WshShortcut object view

WshShortcut.Arguments Property

Description

The **Arguments** property sets or returns a single String representing the arguments passed to the shortcut.

Syntax

```
object.Arguments [= sArgs]
```

Arguments

Parameter	Description
sArgs	Any command-line arguments that are to be passed to the executable (defined by the **TargePath** property). To be meaningful, the target must be capable of recognizing the argument or arguments.

Return Value

String

Notes

- Under **Windows 2000**, this property corresponds to the "Comment" field that can be set when the properties of a **Shortcut** are viewed.
- Under **Windows 2000**, the value of this property is also displayed in a tool-tip when the mouse is held over the shortcut icon.

WshShortcut.Description Property

Description

The **Arguments** property sets or returns a String representing a description of the shortcut. The **Description** property is not visible from the Windows user interface.

Arguments

Parameter	Description
sDescription	The **Description** property contains a string value describing a shortcut.

Return Value

String

WshShortcut.FullName Property

Description

The **FullName** property returns a String containing the full path and filename of the shortcut file. Shortcut files have a file extension of *.lnk.

Return Value

String

Notes

- The **FullName** property contains a read-only string value indicating the fully qualified path to the shortcut's target.

WshShortcut.HotKey Property

Description

The **Hotkey** property assigns a key-combination to a shortcut, or identifies the key-combination assigned to a shortcut.

Return Value

String

Notes

- Sets or returns a String containing the keyboard shortcut that executes the shortcut file; hotkeys apply only to shortcuts located on the Windows desktop or on the Start menu. Multiple keys are joined by a "+" sign. For example, a Hotkey value of "Alt+Ctrl+A" indicates that the shortcut's hotkey is the Alt + Ctrl + A key combination.
- According to the documentation, strings indicating alphabetic keys are case-sensitive ("A" is an uppercase A, but "a" is lowercase), although this does not appear to be the case.
- strings that represent some common non-alphanumeric hotkeys are listed in the following table:

Hotkey String	Description
Alt	Alt key
Back	Backspace Key
Ctrl	Ctrl key
Escape	Escape key
Shift	Shift key
Space	Space key
Tab	Tab key

WshShortcut.IconLocation Property

Description

The **IconLocation** property assigns an icon to a shortcut, or identifies the icon assigned to a shortcut.

Return Value

String

Notes

- Defines the location of the shortcut's icon. Typically, its value is the complete path and filename to the file containing the icon followed by a comma and the zero- based position of the icon within the file. If the default icon is used, the value of **IconLocation** is " ,0".

WshShortcut.TargetPath Property

Description

The **TargetPath** property sets or returns the path and filename to the shortcut's executable file.

Return Value

String

Notes

- Note that the value of the **TargetPath** property can also include a data file that's associated with an executable file.
- This property is for the shortcut's target path only. Any arguments to the shortcut must be placed in the Argument's property.

WshShortcut.WindowStyle Property

Description

The **WindowStyle** property assigns a window style to a shortcut, or identifies the type of window style used by a shortcut.

Return Value

Integer

Notes

- The following table lists the available settings for nWindowStyle

Style	Description
1	Activates and displays a window. If the window is minimized or maximized, the system restores it to its original size and position.
3	Activates the window and displays it as a maximized window.
7	Minimizes the window and activates the next top-level window.

WshShortcut.WorkingDirectory Property

Description

The **WorkingDirectory** property assign a working directory to a shortcut, or identifies the working directory used by a shortcut.

Return Value

String

Notes

- Defines the shortcut's working directory (i.e., the directory in which the shortcut will start).

WshShortcut.Save Method

Description

The **Save** method saves a shortcut object to disk.

Syntax

```
object.Save
```

Notes

- After using the **CreateShortcut** method to create a shortcut object and set the shortcut object's properties, the **Save** method must be used to save the shortcut object to disk.
- **Save** method uses the information in the shortcut object's *FullName* property to determine where to save the shortcut object on a disk.
- You can only create shortcuts to system objects. This includes files, directories, and drives (but does not include printer links or scheduled tasks).

WshUrlShortcut Object

- The **WshURLShortcut** object is similar to the **WshShortcut** object, except that it has only two properties: the **FullName** property, which defines the path and name of the Internet shortcut; and the **TargetPath** directory, which defines the resource to which the Internet shortcut points.
- The **WshURLShortcut** object also supports the **Load** method (which, as in the case of the **WshShortcut** method, is private and cannot be called from **Visual Basic** code) and the **Save** method.
- The **WshUrlShortcut** object represents an Internet shortcut, an Internet link to an Internet resource. A new or an existing **WshUrlShortcut** object is returned by the **CreateShortcut** method of the **WshShell** object.
- A **WshUrlShortcut** object exists in memory only and not in the filesystem until it is saved by calling the object's **Save** method.
- Although the **WshShortcut** object allows you to select the shortcut file's icon from an executable or DLL containing icon resources, the **WshURLShortcut** object does not.
- Instead, the location of a default icon for the shortcut is retrieved from the system registry. This location is determined by the default value stored to the HKEY_CLASSES_ROOT\InternetShortcut\DefaultIcon key.

WshUrlShortcut Properties and Methods

WshUrlShortcut.FullName Property

Description

The **FullName** property returns the fully qualified path of the shortcut object's target.

Return Value

String

Notes

- The **FullName** property is a read-only string representing the fully qualified path to the shortcut's target.

WshUrlShortcut.TargetPath Property

Description

The **TargetPath** property sets or returns a string containing the complete URL of the Internet resource to which the Internet shortcut is linked.

Return Value

String

WshUrlShortcut.Save Method

Description

The **Save** method saves a shortcut object to disk.

Syntax

```
object.Save
```

Notes

- After using the **CreateShortcut** method to create a shortcut object and set the shortcut object's properties, the **Save** method must be used to save the shortcut object to disk.
- **Save** method uses the information in the shortcut object's *FullName* property to determine where to save the shortcut object on a disk.
- You can only create shortcuts to system objects. This includes files, directories, and drives (but does not include printer links or scheduled tasks).

WshEnvironment Object

- The **WshEnvironment** object is a collection object returned by the Environment property of the **WshShell** object; it cannot be created by calls to the object's **CreateObject** or **GetObject** methods
- **WshEnvironment** is a collection of strings containing a set of environment variables. Windows systems maintain two such sets of environment variables, either of which can be returned by the **Environment** property of the **WshShell** object.
- A system table, which is retrieved by supplying the string System as an argument to the **Environment** property of the **WshShell** object. The system table contains the environment variables available to all processes running on the system.
- A process table, which is retrieved by supplying the string Process as an

argument to the Environment property of the **WshShell** object. The process table contains the environment variables defined for the individual process. It also includes the environment variables in the system table.

- Since the **WshEnvironment** object is a child of the **WshShell** object, it requires that a **WshShell** object be instantiated. This requires that you access the **WshEnvironment** collection through a code fragment like the following:

```
Dim oShell, oEnvColl
Set oShell = CreateObject("WScript.Shell")
Set oEnvColl = oShell.Environment
```

- You can then iterate the collection as follows:

```
Dim sResult, sTmp
For Each sTmp in oEnvColl
   sResult = sResult & sTmp & vbCrLf
Next
Msgbox sResult
```

WshEnvironment Properties and Methods

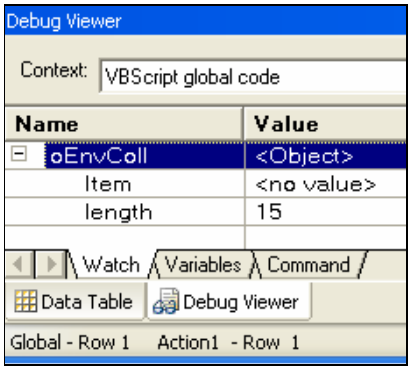

Figure 16 - WshEnvironment collection view

WshEnvironment.Count Property

Description

The **Count** property Indicates the number of environment variables in the collection.

Syntax

```
object.Count
```

Data Type

Integer

Example

```
Option Explicit
Dim oShell, oEnvColl
Set oShell = CreateObject("WScript.Shell")
Set oEnvColl = oShell.Environment
MsgBox oEnvColl.Count
```

WshEnvironment.Item Property

Description

The **Item** property returns an environment variable's name/value pair (separated by an equals sign) if passed a string containing its name (or key). Item is the default member of the **WshEnvironment** collection.

Syntax

```
object.Item(natIndex)
```

Data Type

Integer

Arguments

Parameter	Description
natindex	Item to retrieve.

Example

```
oEnvironment.Item("strName") = strValue
oEnvironment("strName") = strValue
```

WshEnvironment.Remove method

Description

The **Remove** method removes an existing environment variable.

Syntax

```
object.Remove(sName)
```

Data Type

Integer

Arguments

Parameter	Description
sName	String value indicating the name of the environment variable you want to remove.

Notes

- The **Remove** method removes environment variables from the following types of environments: PROCESS, USER, SYSTEM, and VOLATILE. Environment variables removed with the **Remove** method are not removed

permanently; they are only removed for the current session.

Example

The following code removes the **Process** environment variable *TestVar*.

```
Option Explicit
Dim oShell, oEnv

Set oShell = CreateObject("WScript.Shell")
Set oEnv = oShell.Environment("PROCESS")
oEnv("TestVar") = "Windows Script Host"
MsgBox
   oShell.ExpandEnvironmentStrings("The value of the test variable is: '%TestVar%'")
oEnv.Remove "TestVar"
MsgBox
   oShell.ExpandEnvironmentStrings("The value of the test variable is: '%TestVar%'")
```

WshScriptExec Object

- The **WshScriptExec** object represents a local script or application launched by calling the **WshShell**.**Exec** method. Its members provide status information and allow you to access the script or application's standard input, output, and error streams. The **WshScriptExec** object is new to WSH 5.6.
- Provides status information about a script run with **Exec** along with access to the **StdIn**, **StdOut**, and **StdErr** streams.
- The **WshScriptExec** object is returned by the **Exec** method of the **WshShell** object. The **Exec** method returns the **WshScriptExec** object either once the script or program has finished executing, or before the script or program begins executing.

WshScriptExec Properties and Methods

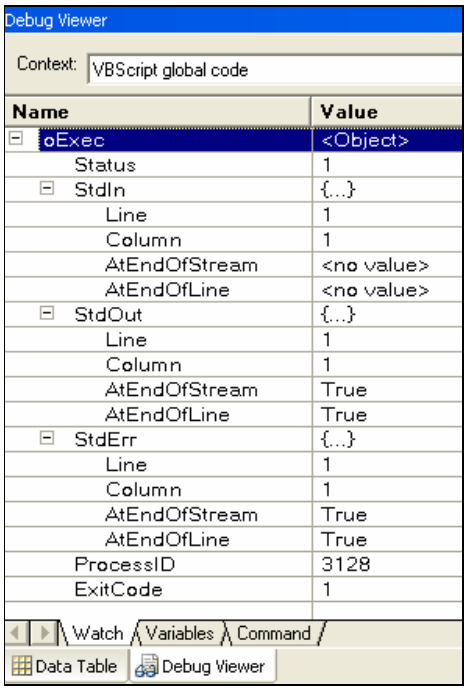

Figure 17 - WsScriptExec object view

WshScriptExec.Status Property

Description

The **Status** property provides status information about a script run with the **Exec** method.

Data Type

Integer (0 = The job is still running, 1 = The job has completed.)

Notes

- The **Status** property is used when a program is run asynchronously.

Example

The following code runs explorer.exe and display the final status to the Reporter.

```
Option Explicit
Dim oShell, oExec
Set oShell = CreateObject("WScript.Shell")
Set oExec = oShell.Exec("Explorer.exe C:\Program Files")
Do While oExec.Status = 0
   Reporter.ReportEvent micDone, "Status", oExec.status
   Wait 0,100
Loop
Reporter.ReportEvent micDone, "Status", oExec.status
```

WshScriptExec.StdOut Property

Description

The **StdOut** property exposes the write-only output stream of the **Exec** object.

Syntax

```
object.StdOut
```

Notes

- The **StdOut** property contains a read-only copy of any information the script may have sent to the standard output.

Example

The following code demonstrates the **StdOut** after executing a DIR command

```
Option Explicit
Dim oShell, oExec
Dim sOut
'--- Creating a Shell object
Set oShell = CreateObject("WScript.Shell")
'--- executing 'notexist' command
Set oExec = oShell.Exec("%comspec% /c Dir /B/O")
'--- looping
Do While True
    '--- Retrieving Standard Error if AtEndOfStream
    If Not oExec.StdErr.AtEndOfStream Then
        sOut = "STDERR: " & oExec.StdErr.ReadAll
        Exit Do
    End If
    '--- Retrieving Standard Output if AtEndOfStream
    If Not oExec.StdOut.AtEndOfStream Then
        sOut = oExec.StdOut.ReadAll
        Exit Do
    End If
Loop
MsgBox sOut
```

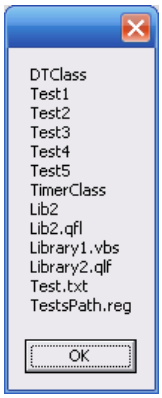

Figure 18 – Dir command – oExec.StdOut

WshScriptExec.StdIn Property

Description

The **StdIn** property exposes the input stream of the **Exec** object.

Notes

- Use the **StdIn** property to pass data to a process started using **Exec**.

WshScriptExec.StdErr Property

Description

The **StdErr** property provides access to the output stream of the **Exec** object.

Notes

- Use the **StdIn** property to pass data to a process started using **Exec**.

Example

The following code demonstrates the **StdErr** object by attempting to execute a non-existent command and displaying the results.

```
Option Explicit
Dim oShell, oExec
Dim sOut
'--- Creating a Shell object
Set oShell = CreateObject("Wscript.Shell")
'--- executing 'notexist' command
Set oExec = oShell.Exec("%comspec% /c notexist")
'--- looping
Do While True
    '--- Retrieving Standard Error if AtEndOfStream
    If Not oExec.StdErr.AtEndOfStream Then
        sOut = "STDERR: " & oExec.StdErr.ReadAll
        Exit Do
    End If
    '--- Retrieving Standard Output if AtEndOfStream
    If Not oExec.StdOut.AtEndOfStream Then
        sOut = oExec.StdOut.ReadAll
        Exit Do
    End If
Loop
MsgBox sOut
```

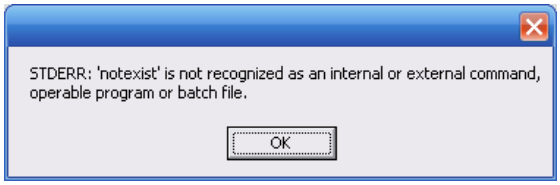

Figure 19 - Invalid command (StdErr)

WshScriptExec.Terminate Method

Description

The **Terminate** method Instructs the script engine to end the process started by the **Exec** method.

Syntax

```
object.Terminate
```

Notes

- Sends a **WM_CLOSE** message to a process (a script or an application) launched by calling the **WshShell.Exec** method. How the message is handled depends on the application: it can ignore the message, or it can terminate.

WshSpecialFolders Object

- The **WshSpecialFolders** object is a read-only collection object that contains all the special Windows folders such as the desktop folder, the Start Menu folder, and the personal document folder.
- This object cannot be instantiated directly. It can only be accessed through the **WshShell.SpecialFolders** object
- As with all collection objects, the **WshSpecialFolders** collection can be iterated using the **VBScript For Each…Next** statement

WshSpecialFolders Properties and Methods

WshSpecialFolders.Count Property

Description

The **Count** property Indicates the number of items in the collection..

Example

```
Option Explicit
Dim oShell, oSpFoldersColl
Set oShell = CreateObject("WScript.Shell")
Set oSpFoldersColl = oShell.SpecialFolders
MsgBox oSpFoldersColl.Count
```

WshSpecialFolders.Item Property

Description

The **Item** property returns an individual item from the collection; each item is a string that indicates the location of a particular special folder.

Syntax

```
object.Item(natIndex)
```

Arguments

Parameter	Description
natindex	Item to retrieve.

Notes

- The **Item** property is the default of the **WshSpecialFolders** object. It takes a single argument of type Long, and returns the element in the collection at the specified position. Indexing starts at zero. If there is no element at the requested index, a "Subscript out of range" error is
- generated.
- A key string can also be used to access the items in the **WshSpecialFolders**.
- If the member doesn't exist, the Item property returns an empty variant.
- An item is retrieved from the collection either by its ordinal position in the collection or by its key; valid key values are:

- AllUsersDesktop
- AllUsersStartMenu
- AllUsersPrograms
- AllUsersStartup
- Desktop
- PrintHood
- Programs
- Recent
- SendTo
- Fonts
- MyDocuments
- NetHood
- StartMenu
- Startup
- Templates
- Favorites

WshSpecialFolders.length Property

Description

The **length** property Indicates the number of items in the collection..

WshNetwork Object

- Provides access to the shared resources on the network to which your computer is connected.
- The **WshNetwork** is a **COM** object. The object represents network resources that are available to a client computer. You can create a **WshNetwork** object with a code fragment like the following:

```
Option Explicit
Dim oNet
Set oNet = CreateObject("WScript.Network")
```

- 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.

Managing Network Drives

Network drives remain an important part of the computing infrastructure. Users prefer mapped network drives to **Universal Naming Convention** (**UNC**) path names; it is easier to remember that financial records are stored on drive X than to remember that financial records are stored on \\atl-fs-01\departments\accounting\admin\financial_records\2002_archive. Is possible to map network drives as well; if financial records or tests need to be moved to a new server, it is far easier to remap drive X than to expect users to memorize the new location.

Your scripts can use the methods in this section to manage network drive connections as part of a logon script or in any **WSH** script that has a need to connect to or disconnect from a network share. In fact, this represents one area where **WSH** has system administration capability not found in **WMI**. While both **WSH** and **WMI** let you enumerate mapped network drives (**WMI** by using the **Win32_MappedLogicalDisk** class), only **WSH** enables you to create and delete drive mappings.

WshNetwork provides three methods to work with network drive connections: **MapNetworkDrive**, **RemoveNetworkDrive**, and **EnumNetworkDrives**. You can use the three methods to manage network connections as part of a users logon script or in any **WSH** script that needs to connect to or disconnect from a network share.

Managing Network Printers

Managing printer connections on user computers is an important part of system administration. When a new printer comes online, you do not have to send instructions on how to connect to this device; instead, you can simply include code in a logon script that automatically makes this connection for a user. Likewise, when a printer is removed from the network, you can remove the printer connection, preventing the problems likely to arise when users try to print to a printer that no longer exists.

The **WshNetwork** object provides methods that enable your scripts to add or remove printer connections, to set the default printer, and to list the current printer connections on a computer.

WshNetwork Properties and Methods

WshNetwork.ComputerName Property

Description

The **ComputerName** property returns the name of the computer system.

Return Value

String

Notes

- The property is equivalent to `Environment.Value("LocalHostName")`

Example

The following example demonstrates the use of the **ComputerName** property.

```
Option Explicit
Dim oNet
Set oNet = CreateObject("WScript.Network")
MsgBox "Domain = " & oNet.UserDomain
MsgBox  "Computer Name = " & oNet.ComputerName
MsgBox  "User Name = " & oNet.UserName
Set oNet = Nothing
```

WshNetwork.UserDomain Property

Description

The **UserDomain** property returns a user's domain name.

Return Value

String

Example

The following example demonstrates the use of the **UserDomain** property.

```
Option Explicit
Dim oNet
Set oNet = CreateObject("WScript.Network")
MsgBox "Domain = " & oNet.UserDomain
MsgBox  "Computer Name = " & oNet.ComputerName
MsgBox  "User Name = " & oNet.UserName
Set oNet = Nothing
```

WshNetwork.UserName Property

Description

The **UserName** property returns the name of a user.

Return Value

String

Notes

- The property is equivalent to `Environment.Value("UserName")`

Example

The following example demonstrates the use of the **UserName** property.

```
Option Explicit
Dim oNet
Set oNet = CreateObject("WScript.Network")
MsgBox "Domain = " & oNet.UserDomain
MsgBox  "Computer Name = " & oNet.ComputerName
MsgBox  "User Name = " & oNet.UserName
```

```
Set oNet = Nothing
```

WshNetwork.AddWindowsPrinterConnection Method

Description

The **AddWindowsPrinterConnection** method adds a Windows-based printer connection to your computer system.

Syntax

```
object.AddWindowsPrinterConnection(sPrinterPath)
```

Arguments

Parameter	Description
sPrinterPath	String value indicating the path to the printer connection.

Return Value

String

Notes

- Using this method is similar to using the Printer option on Control Panel to add a printer connection.
- Unlike the **AddPrinterConnection** method, this method allows you to create a printer connection without directing it to a specific port, such as LPT1. If the connection fails, an error is thrown.

WshNetwork.AddPrinterConnection Method

Description

The **AddPrinterConnection** method adds a remote **MS-DOS**-based printer connection to your computer system.

Syntax

```
object.AddPrinterConnection(
   sLocalName, sRemoteName, bUpdateProfile, sUser, sPassword )
```

Arguments

Parameter	Description
sLocalName	Sting value indicating the local name to assign to the connected printer.
sRemoteName	Sting value indicating the name of the remote printer.
bUpdateProfile	Optional. Boolean value indicating whether the printer mapping is stored in the current user's profile. If *bUpdateProfile* is supplied and is **true**, the mapping is stored in the user profile. The default value is **false**.

sUser	Optional. String value indicating the user name. If you are mapping a remote printer using the profile of someone other than current user, you can specify *sUser* and *sPassword*.
sPassword	Optional. String value indicating the user password. If you are mapping a remote printer using the profile of someone other than current user, you can specify *sUser* and *sPassword*.

Return Value

String

Notes

- The **AddPrinterConnection** method adds a network printer to an MS-DOS printer port, such as LPT1.
- You cannot use this method to add a remote Windows-based printer connection.
- To add a remote Windows-based printer connection, use the **AddWindowsPrinterConnection** method.

WshNetwork.EnumNetworkDrives Method

Description

The **EnumNetworkDrives** method returns the current network drive mapping information.

Syntax

```
object.EnumNetworkDrives
```

Return Value

Collection of Strings

Notes

- Returns a zero-based collection of strings containing the current network drive mappings. All members having even index values are local names (drive letters), and all having odd index values are the remote names of the immediately preceding local drive.
- The collection returned by the method supports the following properties:
  - Count - The number of items in the collection
  - Item - Returns an individual item from the collection.
  - length - The number of items in the collection.

Example

The following example uses **EnumNetworkDrives** to generate a list of the networked drives and displays the mapping information.

```
Option Explicit
Dim oNet, oDrives
Dim i
'--- Adding to new entries to datatable
DataTable.LocalSheet.AddParameter "Drive", vbNullString
DataTable.LocalSheet.AddParameter "Value", vbNullString
```

```
'--- Creating a WshNetwork object
Set oNet = CreateObject("WScript.Network")
Set oDrives = oNet.EnumNetworkDrives
For i = 0 to oDrives.Count - 1 Step 2
        DataTable.LocalSheet.SetCurrentRow i + 1
        DataTable("Drive", dtLocalSheet) = oDrives.Item(i)
        DataTable("Value", dtLocalSheet) = oDrives.Item(i + 1)
Next
Set oDrives = Nothing : Set oNet = Nothing
```

WshNetwork.EnumPrinterConnections Method

Description

The **EnumPrinterConnections** method returns the current network printer mapping information.

Syntax

```
object.EnumPrinterConnections
```

Return Value

Collection of Strings

Notes

- Returns a zero-based collection of strings containing the current network printer mappings.
- All members having even index values are the ports, and all members having odd index values are the network mappings of the preceding port.
- The collection returned by the method has the following members:
  - Count - The number of items in the collection
  - Item - Returns an individual item from the collection
  - length - The number of items in the collection

Example

The following example uses the **EnumPrinterConnections** method to generate a list of networked printers and displays this mapping information.

```
Option Explicit
Dim oNet, oPrinters
Dim i
'--- Adding to new entries to datatable
DataTable.LocalSheet.AddParameter "Port", vbNullString
DataTable.LocalSheet.AddParameter "Value", vbNullString
'--- Creating a WshNetwork object
Set oNet = CreateObject("WScript.Network")
Set oPrinters = oNet.EnumPrinterConnections
For i = 0 to oPrinters.Count - 1 Step 2
        DataTable.LocalSheet.SetCurrentRow i + 1
        DataTable("Port", dtLocalSheet) = oPrinters.Item(i)
        DataTable("Value", dtLocalSheet) = oPrinters.Item(i + 1)
Next
```

```
Set oPrinters = Nothing : Set oNet = Nothing
```

WshNetwork.MapNetworkDrive Method

Description

The **MapNetworkDrive** method Adds a shared network drive to your computer system.

Syntax

```
object.MapNetworkDrive( _
      sLocalName, sRemoteName, bUpdateProfile, sUser, sPassword )
```

Arguments

Parameter	Description
sLocalName	String value indicating the name by which the mapped drive will be known locally.
sRemoteName	String value indicating the share's UNC name (\\xxx\yyy).
bUpdateProfile	Optional. Boolean value indicating whether the mapping information is stored in the current user's profile. If *bUpdateProfile* is supplied and has a value of **true**, the mapping is stored in the user profile (the default is **false**).
sUser	Optional. String value indicating the user name. You must supply this argument if you are mapping a network drive using the credentials of someone other than the current user.
sPassword	Optional. String value indicating the user password. You must supply this argument if you are mapping a network drive using the credentials of someone other than the current user.

Notes

- An attempt to map a non-shared network drive results in an error.

WshNetwork.RemoveNetworkDrive Method

Description

The **RemoveNetworkDrive** method removes a shared network drive from your computer system.

Syntax

```
object.RemoveNetworkDrive(sName, bForce, bUpdateProfile)
```

Arguments

Parameter	Description

sName	String value indicating the name of the mapped drive you want to remove. The *sName* parameter can be either a local name or a remote name depending on how the drive is mapped.
bForce	Optional. Boolean value indicating whether to force the removal of the mapped drive. If *bForce* is supplied and its value is **true**, this method removes the connections whether the resource is used or not.
bUpdateProfile	Optional. String value indicating whether to remove the mapping from the user's profile. If *bUpdateProfile* is supplied and its value is true, this mapping is removed from the user profile. *bUpdateProfile* is false by default.

Notes

- If the drive has a mapping between a local name (drive letter) and a remote name (UNC name), then *sName* must be set to the local name.
- If the network path does not have a local name (drive letter) mapping, then *sName* must be set to the remote name.

Tips

- Before removing a network drive, check if exist, using **EnumNetworkDrives**, to avoid runtime error

WshNetwork.RemovePrinterConnection Method

Description

The **RemovePrinterConnection** method removes a shared network printer connection from your computer system.

Syntax

```
object.RemovePrinterConnection( sName, bForce, bUpdateProfile )
```

Arguments

Parameter	Description
sName	String value indicating the name that identifies the printer. It can be a UNC name (in the form \\xxx\yyy) or a local name (such as LPT1).
bForce	Optional. Boolean value indicating whether to force the removal of the mapped printer. If set to **true** (the default is **false**), the printer connection is removed whether or not a user is connected.
bUpdateProfile	Optional. Boolean value. If set to **true** (the default is **false**), the change is saved in the user's profile.

Notes

- The **RemovePrinterConnection** method removes both Windows and MS-DOS based printer connections.
- If the printer was connected using the method **AddPrinterConnection**, *sName* must be the printer's local name.
- If the printer was connected using the **AddWindowsPrinterConnection** method or was added manually (using the Add Printer wizard), then *sName* must be the printer's UNC name.

WshNetwork.SetDefaultPrinter Method

Description

The **SetDefaultPrinter** method assigns a remote printer the role Default Printer.

Syntax

```
object.SetDefaultPrinter( sPrinterName )
```

Arguments

Parameter	Description
sPrinterName	String value indicating the remote printer's UNC name.

Notes

- The **SetDefaultPrinter** method fails when using a DOS-based printer connection.
- You cannot use the **SetDefaultPrinter** method to determine the name of the current default printer.

WshController Object

One limitation of **WSH** has always been the fact that scripts could only be run locally; they could not be run against remote computers. For instance, suppose you want to use a script to add printer connections to a number of computers. You would have to run that script on each of those computers; historically, you could not run a script on Computer A and get it to add a printer connection to Computer B. Because of that, **WSH** has primarily been relegated to use in logon scripts.

Clearly, it would be useful to automate the running of a script on remote computers and on multiple computers. The **WshController** object, introduced in WSH 5.6, provides that capability.

The **WshController** object allows you to create a controller script that can run worker scripts against remote computers. The controller script initiates, monitors, and, if necessary, terminates the worker script. The worker script, meanwhile, is simply the script that carries out the administrative task for example, adding a printer connection or mapping a network drive.

The worker scripts do not have to be located on the same computer as the controller script, although their location must be specified relative to the computer on which the controller script runs. For instance, you can create a controller script that accesses a worker script in a shared folder on another computer and runs that worker script on yet another computer.

The **WshController** object provides communication among all the local and remote computers involved. The script that runs on the remote computer is never saved to disk on the remote computer. Instead, the **WshController** object starts

the script within a **WSH** process in the memory of the remote computer.

- In addition of **WMI**, for the most part, works exactly the same on remote computers as it does on the local computer.
- To run **WSH** scripts remotely, you must use the WshController object and actually create two scripts: the script to be run and the script that allows that script to run remotely.
- The **WshController** is a COM object, which is new to WSH 5.6, allows for the creation of a remote script process. **WshController** is a creatable object that must be instantiated with a code fragment like the following:

```
Dim oCtrl
Set oCtrl = CreateObject("WSHController")
```

- The **WshController** object has a single method, **CreateScript**, this method that accesses the script to be run remotely and returns a **WshRemote** object that provides some control over the resulting script process.
- The **ProgID** of the **WshController** object does not follow the same naming convention as the **WshNetwork** (**WScript.Network**) and **WshShell** (**WScript.Shell**) objects. The **ProgID** of the **WshController** object has no dot (.) and does not begin with **WScript**. It is simply the name of the object: **WshController**.

Setup Required to Run Scripts Remotely

Before you can use the **WshController** object to run scripts on remote computers, you must first ensure that your environment is configured as follows:

- Both the local and target remote computers must be running **WSH** version 5.6.
- You must add a string-valued entry (REG_SZ) named Remote to the registry subkey HKEY_LOCAL_ MACHINE\SOFTWARE\Microsoft\Windows Script Host\Settings and set its value to 1 on all target remote computers. You do not need to add this entry to the registry of the local computer from which you run the controller script.

The following code, is adding a registry entry to enable running scripts remotely using **WMI**

```
Option Explicit

Const HKEY_LOCAL_MACHINE = &H80000002
Dim oRegProv
Dim sKeyPath

sComputer = "RemoteComputerName"
Set oRegProv = GetObject("winmgmts:{impersonationLevel=Impersonate}" & _
 "!\\" & sComputer & "\root\default:StdRegProv")

sKeyPath = "SOFTWARE\Microsoft\Windows Script Host\Settings"
oRegProv.SetStringValue HKEY_LOCAL_MACHINE,strKeyPath, "Remote", "1"
Set oRegProv = Nothing
```

WshController Capabilities

The **WshController** object enables you to run scripts on remote computers, monitor the status of remotely running scripts, and examine errors produced by these scripts. The following table, lists these categories along with the methods, properties, and events of the **WshController** object, the **WshRemote** object, and the **WshRemoteError** object that your scripts can use to access this functionality.

Task Category	Method, Property or Event
Running Scripts on Remote Computers	CreateScript, Execute, Terminate
Monitoring Status of Remotely Running Scripts	Status, Start, End, Error (event)
Examining Errors Produced by Remotely Running Scripts	Error (event), Error (property), Character, Description, Line, Number, Source, SourceText

WshController Limitations

Remote **WSH** has two important limitations. First, there is no easy way to retrieve the output from a remotely run script. Second, remotely run scripts cannot access shared folders using the credentials of the user who ran the controller script.

To work around the first problem, you can create a text file that holds the results of your worker script. You might create the file on a shared folder on each of the remote computers you are running the worker scripts on and then retrieve the files and store them in a central location.

However, there is no way around the second problem, at least not without creating potential security vulnerabilities.

WshNetwork Properties and Methods

WshController.CreateScript Method

Description

The **CreateScript** method Creates a **WshRemote** object.

Syntax

```
object.CreateScript(CommandLine, MachineName)
```

Arguments

Parameter	Description
CommandLine	Required. String value indicating the script's path and switches as they would be typed at the command prompt. The path to the script should appear as seen from the controller computer system rather than the computer system on which you want to run the script.
MachineName	Optional. String value indicating the name of the remote computer system (the computer on which you want to run the remote script). It

	is specified in the Uniform Naming Convention (**UNC**).

Return Value

WshRemote object

Notes

- The **CreateScript** method returns a handle to an instance of a **WshRemote** object.
- The path part of the script name does not need to be local — it can refer to a script on a network share. This makes it possible to sit at one computer system, retrieve a script from another computer system, and run it on a third computer system.
- If a machine name is not provided, the remote script object runs on the controller computer system (this is the default).
- If a machine name is provided, the remote script object runs on the named computer system.
- The **CreateScript** method establishes a connection with the remote computer system and sets it up to run the script, but the script does not actually start until you call the **Execute** method of the **WshRemote** object.

Example

The following code runs the hypothetical worker script MapNetworkDrive.vbs on the remote computer, Server01. Although the location of the worker script can be specified using either a local file path or a **UNC** path, because no path is specified, this means the worker script must be located in the same directory as the controller script.

```
Option Explicit
Dim oController, oRemoteScript
Dim sRemote, sWorkerScript

sRemote = "Server01"
sWorkerScript = "MapNetworkDrive.vbs"
Set oController = CreateObject("WshController")
Set oRemoteScript = oController.CreateScript(sWorkerScript, sRemote)
oRemoteScript.Execute
Do While Not oRemoteScript.Status = 2
        Wait 0, 500
    Reporter.ReportEvent micDone, "Status", "script not yet complete."
Loop
Reporter.ReportEvent micPass, "Status", "Remote script was complete."
```

WshRemote Object

- The WshRemote object allows you to remotely administer computer systems on a computer network.
- It represents an instance of a **WSH** script, i.e., a script file with one of the following extensions: .wsh, .wsf, .js, .vbs, .jse, .vbe, and so on. An instance of a running script is a process.
- You can run the process either on the local machine or on a remote machine.
- If you do not provide a network path, it will run locally. When a **WSHRemote**

object is created (by using the **CreateScript**() method), the script is copied to the target computer system. Once there, the script does not begin executing immediately; it begins executing only when the **WSHRemote** method **Execute** is invoked.

- Through the **WshRemote** object interface, your script can manipulate other programs or scripts.
- Additionally, external applications can also manipulate remote scripts. The **WshRemote** object works asynchronously over **DCOM**.

WshRemote Properties and Methods

WshRemote.Status Property

Description

The **Status** property returns the current status of the remote-executing script.

Return Value

Integer

Constant	Value	Description
wshNoTask	0	indicates that the remote script has not yet started to run.
wshRunning	1	indicates that the remote script is currently running.
wshFinished	2	The job has completed.

Notes

- Running scripts on remote computers is most useful if you can be sure that the scripts actually ran successfully. You can do so by using the **WshRemote** object and monitoring events that are generated when the remote script runs.

WshRemote.Error Property

Description

The **Error** property exposes the **WshRemoteError** object, which holds information about the error that caused the remote script to terminate prematurely.

Return Value

WshRemoteError object.

Notes

- Fired when an error occurs in the remote script. No parameters are passed to the event handler.

WshRemote.Execute Method

Description

The **Execute** method starts execution of a remote script object.

Syntax

```
object.Execute
```

Notes

- The Start event of the **WshRemote** object is fired when the script starts executing. Do not confuse the **Execute** method with the **Exec** method (of the **WScript** object).

WshRemote.Terminate Method

Description

The **Terminate** method instructs the script engine to end the process started by the **Exec** method.

Syntax

```
object.Terminate
```

Notes

- The **Terminate** method does not return a value.
- Use the **Terminate** method only as a last resort since some applications do not clean up properly. As a general rule, let the process run its course and end on its own.
- The **Terminate** method attempts to end a process using the WM_CLOSE message.
- If that does not work, it kills the process immediately without going through the normal shutdown procedure.

WshRemote.Start, End and Error Events

Events are not accessible, when using **QuickTest**.

WshRemoteError Object

Provides access to the error information available when a remote script (a **WshRemote** object) terminates as a result of a script error.

Knowing that a remotely executing script has encountered an error is important. However, once you know the error has occurred, you next need to determine the cause and correct the problem. If an error occurs in a remotely executing script, the **WshRemoteError** object can be accessed as a property of the **WshRemote** object.

The **WshRemoteError** object includes a number of properties that describe the error that occurred and can help you troubleshoot the problem.

The **WshRemoteError** object is returned by the **Error** property of the **WshRemote** object.

WshRemoteError Properties

WshRemoteError.Character Property

Description

The **Character** property Reports the specific character in a line of code that contains an error.

Notes

- Some errors are not associated with a particular character position. For example, consider the error Expected End If.
- In this case, there is no line (a line of code is missing). In such a case, the Character property returns zero (0).
- The character position is based on an offset of one (1) (the first character in a line resides at position one).

WshRemoteError.Description Property

Description

The **Description** property contains a brief description of the error that caused the remote script to terminate.

Notes

- Returns a string containing a brief description of the error, or an empty string if none is available.

WshRemoteError.Line Property

Description

The **Line** property Identifies the line in a script that contains an error.

Notes

- Notice that some errors do not occur on a particular line. For example, consider the error Expected **End If**.
- In this case, there is no line (a line of code is missing). In such a case, the Line property returns zero (0).

WshRemoteError.Number Property

Description

The **Number** property reports the error number representing a script error.

WshRemoteError.Source Property

Description

The **Source** property Identifies the **COM** object responsible for causing the

script error.

WshRemoteError.SourceText Property

Description

The **SourceText** property contains the line of source code that caused an error.

Notes

- It is not always possible to obtain the source text.
- If that is the case, the **SourceText** property returns an empty string.

Q&A

How to add an item to the Quick-Launch bar?

The following code uses a **URL** shortcut to create a Quick Launch button that opens the Microsoft® TechNet Web site. The Quick Launch button is visible only to users who run the script because the shortcut is created in the personal Quick Launch bar for the user.

```
Option Explicit
Dim oShell, oEnvVarsCol, oURLShortcut
Dim sFolder

Set oShell = CreateObject("WScript.Shell")
Set oEnvVarsCol = oShell.Environment("Volatile")
sFolder = oEnvVarsCol.Item("APPDATA") & "\Microsoft\Internet Explorer\Quick Launch"
Set oURLShortcut = oShell.CreateShortcut(sFolder & "\TechNet.url")
oURLShortcut.TargetPath = "http://www.microsoft.com/technet"
oURLShortcut.Save
Set oURLShortcut = Nothing :  Set oEnvVarsCol = Nothing
Set oShell = Nothing
```

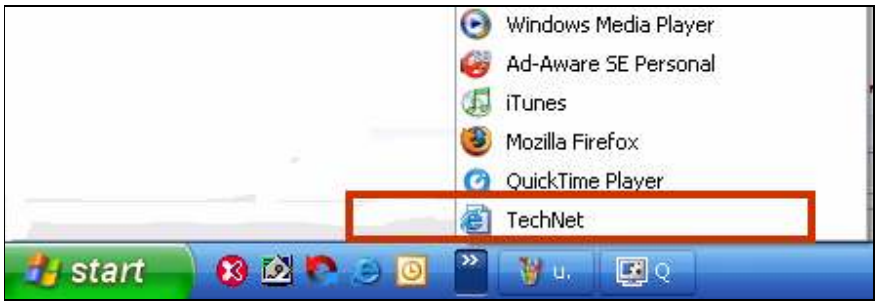

Figure 20 - Creating a Quick Launch Button to Open the TechNet Online Web Site

How to remove an item shortcut?

Shortcuts are files that can be deleted in the same way you delete any other file. For example, the following script deletes the shortcut created in the previous sample.

```
Option Explicit
Dim oShell, oEnvVarsCol, oFSO
Dim sFile
Set oShell = CreateObject("WScript.Shell")
Set oEnvVarsCol = oShell.Environment("Volatile")
Set oFSO = CreateObject("Scripting.FileSystemObject")
sFile = oEnvVarsCol.Item("APPDATA") & _
               "\Microsoft\Internet Explorer\Quick Launch\TechNet.URL"
oFSO.DeleteFile(sFile)
Set oFSO = Nothing :  Set oEnvVarsCol = Nothing
Set oShell = Nothing
```

How to create a shortcut to a Special Folder?

When you install an application, that application often associates itself with a particular file type. For example, when you install the Microsoft® FrontPage® Web site creation and management tool, **FrontPage** associates itself with all .asp files. If you right-click on a .asp file and click Edit, the file will open in **FrontPage**.

Of course, there might be times when you simply want to view the .asp file in Notepad. Recognizing this fact, Windows includes a special folder named **SendTo**. The **SendTo** option presents a menu of applications or locations to which you can send the selected file. You can add to the options available in the **SendTo** menu by adding shortcuts to the **SendTo** special folder. The next time you want to open an .asp file (or any file) in Notepad, you can simply right-click the file, select **SendTo**, and then click Notepad.

```
Option Explicit
Dim oShell, oShortcut
Dim sSendToFld, sNotePad

Set oShell = CreateObject("WScript.Shell")
sSendToFld = oShell.SpecialFolders("SendTo")
sNotePad = oShell.ExpandEnvironmentStrings("%SystemRoot%/system32/notepad.exe")
Set oShortcut = oShell.CreateShortcut(sSendToFld & "\notepad.lnk")
oShortcut.TargetPath = sNotePad
oShortcut.Save
Set oShortcut = Nothing : Set oShell = Nothing
```

Listing Current Network Drives?

The **EnumNetworkDrives** method returns a collection that holds pairs of items: network drive local names and their associated **UNC** names. The collection is zero-indexed; the even-numbered items in the collection are the local drive names, and the odd-numbered items are the associated **UNC** paths.

```
Option Explicit
Dim oNet, oDrives
Dim i
Set oNet = CreateObject("WScript.Network")
Set oDrives = oNet.EnumNetworkDrives
For i = 0 to oDrives.Count - 1 Step 2
    Reporter.ReportEvent micDone, oDrives.Item(i), oDrives.Item (i + 1)
Next
Set oDrives = Nothing : Set oNet = Nothing
```

How to examine errors produced by a remote script?

The following code includes a subroutine that handles the Error event. This script uses the **WshRemote** object **Error** property to retrieve an instance of the **WshRemoteError** object. The script then displays all of the properties of the **WshRemoteError** object, providing you with a great deal of information that is useful for troubleshooting the problem.

```
Option Explicit
Dim sRemoteComp, sScript
Dim oController, oRemote, oRemoteErr
sRemoteComp = "Server01"
sScript = "CreateTestFile.vbs"
'--- Create a controller object and a remote script object
Set oController = CreateObject("WshController")
Set oRemote = oController.CreateScript(sScript, sRemoteComp)
oRemote.Execute
'--- Looping the remote script status
Do While Not oRemote.Status = 2
   Wait 0, 200
   Set oRemoteErr = oRemote.Error
   If Not oRemoteErr Is Nothing Then
      '--- Reporting the error
      Call ErrorReport(oRemoteErr)
      oRemote.Terminate
      Exit Do
   End If
Loop
Set oRemote = Nothing : Set oController = Nothing
'--- Report remote error procedure.
Sub ErrorReport( ByVal oError )
   Dim sMsg
   sMsg = "Error Running Remote Script." & vbCrLf
   sMsg = sMsg & "Character :" & oError.Character & vbCrLf
   sMsg = sMsg & "Description :" & oError.Description & vbCrLf
   sMsg = sMsg & "Line :" & oError.Line & vbCrLf
   sMsg = sMsg & "Number :" & oError.Number & vbCrLf
   sMsg = sMsg & "Source :" & oError.Source & vbCrLf
   sMsg = sMsg & "SourceText :" & oError.SourceText
   Reporter.ReportEvent micDone, "Error", sMsg
End Sub
```