Windows Script Host

Windows Script Host

In This Section

Build: Topic Version 5.6.9309.1546

Windows Script Host

Getting Started

In This Section

Description of the syntax and conventions used in WSH 5.6 help documentation.

Related Sections

WSH Reference
List of elements that make up WSH Reference.
WSH Basics
Learn the basics of WSH.

Build: Topic Version 5.6.9309.1546

Windows Script Host

What's New In WSH 5.6

Several areas of functionality have been addressed in this latest version of the Windows Script Host (version 5.6).

- **Argument handling has been improved** — Handling and documenting command line arguments is simpler. The process of integrating your scripts with other command line scripts has been simplified, and it is easier to create scripts that can supply the user with help information. Refer to the following table for information on the WSH language features that connect you to this new functionality.

To Learn About	See
Grouping your script's switches together.	<runtime> Element
Defining your script's named switches.	<named> Element
Defining your script's unnamed switches.	<unnamed> Element
Making your script self-documenting.	<example> Element
	<description> Element
Sharing the environment of the current process (IOW, WSH) with a spawned	Exec Method

process.	
Accessing the standard streams programmatically.	Exec Method WshScriptExec Object
Accessing environment variables programmatically.	Environment Property WshEnvironment Object ExpandEnvironmentStrings Method Remove Method
Determining whether a spawned script process is currently running.	Status Property (WshScriptExec)
Accessing the spawned script process's StdIn input stream.	StdIn Property (WshScriptExec)
Accessing the spawned script process's StdOut output stream.	StdOut Property (WshScriptExec)
Accessing the spawned script process' StdErr output stream.	StdErr Property (WshScriptExec)
Terminating a spawned script process.	Terminate Method (WshScriptExec)
Accessing the named command-line script arguments.	WshNamed Object
Determining whether a specific key value exists in the WshNamed object.	Exists Method
Determining the number of switches in the WshNamed or WshUnnamed objects.	Count Method

- **You can run scripts remotely** — You can load scripts onto several remote computer systems and start them all running simultaneously. While a remote script is running, you can check its progress. After it has finished, you can ensure that it ran correctly or determine the cause of its premature termination. There is a new dispatch object used to create remote WSH objects — the **Controller** object. In addition, there is a new object that represents an instance of a running script — the **Remote WSH** object.

To Learn About	See
Creating a remote script object — the remote WSH interface.	WshController Object
Creating a remote script object — using remote WSH interface.	CreateScript Method
Creating a remote script object — getting a handle.	WshRemote Object
Starting a remote script process.	Execute Method

Determining whether a remote script is currently running.	Status Property (WshRemote)
Determining why a remote script terminated.	Description Property (WshRemoteError)
Identifying which statement in your remote script caused it to terminate.	Line Property (WshRemoteError)
Accessing error information after a remote script terminates.	WshRemoteError Object
Identifying the character in the line of code that contained the error.	Character Property
Identifying the error number representing a script error.	Number Property
Identifying the source of the script error.	Source Property
Identifying the line of source code that caused an error.	SourceText Property
Handling remote object events.	Start Event
	End Event
	Error Event

- **When you start new processes, you can treat them as objects** — You determine the status of spawned processes and access their standard I/O streams.

To Learn About	See
Spawning a process.	Exec Method
Accessing the object that represents running processes.	WshScriptExec Object
Accessing process status information.	Status Property (WshScriptExec)
Accessing the standard I/O streams.	StdOut Property (WshScriptExec)
	StdIn Property (WshScriptExec)
	StdErr Property (WshScriptExec)

- **You can access the current working directory —** You can determine/modify the active process's current working directory.

To Learn About	See
Accessing the active directory information.	CurrentDirectory Property

- **Security issues unique to scripts have been addressed** — A new security model makes distributing and running scripts safer.

To Learn About	See
Script signing and verification.	Security and Windows Script Host

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Upgrading Windows Script Host

If you are currently using Windows 2000 or Windows ME, you have version 2.0 of WSH installed on your computer system. If you are running Windows 95, 98, or Windows NT 4.0, you have version 1.0. To upgrade to WSH 5.6, visit the Microsoft Windows Script Technologies Web site at (http://msdn.microsoft.com/scripting/).

> **Note** The latest version of WSH is 5.6 due to a file versioning issue that was easiest to resolve by skipping some version numbers.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Document Conventions

Throughout the Windows Script Host 5.6 Help documentation, different fonts, weights, and colors are used to draw your attention to text of interest.

Examples of Text Styles

Type of Information	Sample
Code snippets appear in Courier blue	`WScript.Echo "Hello from VBScript"`
Featured elements in code snippets appear bolded in Courier blue	`WScript.Echo "Hello from VBScript"`
Keywords appear bolded	in the **Script** tab within the **Properties** dialog…
Links appear underlined	WshNetwork Object

Pop-up links appear italicized, and underlined <?job *error*="flag" *debug*="flag" ?>

Build: Topic Version 5.6.9309.1546

Windows Script Host

WSH Version Information

The following table lists the version of Windows Script Host implemented by Microsoft host applications.

Host Application	1.0	2.0	5.6
Microsoft Windows 98	x		
Microsoft Windows NT 4 Option Pack	x		
Microsoft Windows 2000		X	

The following table lists Windows Script Host language elements and the version in which they were introduced.

Language Element	1.0	2.0	5.6
<?job ?> Element		X	
<?XML ?> Element		X	
AddPrinterConnection Method	x		
AddWindowsPrinterConnection Method		X	
AppActivate Method		X	
Arguments Property	x		
AtEndOfLine Property		X	
AtEndOfStream Property		X	
Character Property			x
Close Method		X	
Column Property		X	
ComputerName Property	x		

ConnectObject Method		x	
Count Method	x		
CreateObject Method	x		
CreateScript Method			x
CreateShortcut Method	x		
Description Property	x		
Description Property (WshRemote)			x
DisconnectObject Method	x		
Echo Method	x		
EnumNetworkDrives Method	x		
EnumPrinterConnections Method	x		
Environment Property	x		
Error Property (WshRemote)			x
<example> Element			x
Exec Method			x
Execute Method			x
Exists Method			x
ExitCode Property			x
ExpandEnvironmentStrings Method	x		
FullName Property	x		
GetObject Method	x		
GetResource Method		x	
HotKey Property	x		
IconLocation Property	x		
Item Property	x		
Item Property (WshNamed)			x
Item Property (WshUnnamed)			x
<job> Element		x	x
Length Property	x		
Line Property		x	
Line Property (WshRemote)			x
LogEvent Method		x	
MapNetworkDrive Method	x		
Name Property	x		

<named> Element			x
Number Property			x
<object> Element		x	
<package> Element		x	
Path Property	x		
Popup Method	x		
ProcessID Property			x
Quit Method	x		
Read Method		x	
ReadAll Method		x	
ReadLine Method		x	
<reference> Element		x	
RegDelete Method	x		
RegRead Method	x		
RegWrite Method	x		
Remove Method	x		
RemoveNetworkDrive Method	x		
RemovePrinterConnection Method	x		
<resource> Element			
Run Method	x		
<runtime> Element			x
Save Method	x		
<script> Element		x	
ScriptFullName Property	x		
ScriptName Property	x		
SendKeys Method		x	
SetDefaultPrinter Method	x		
ShowUsage Method			x
Skip Method		x	
SkipLine Method		x	
Sleep Method		x	
Source Property			x
SourceText Property			x
SpecialFolders Property	x		

Status Property (WshRemote)			x
Status Property (WshScriptExec)			x
StdErr Property		x	
StdErr Property (WshScriptExec)			x
StdIn Property		x	
StdIn Property (WshScriptExec)			x
StdOut Property		x	
StdOut Property (WshScriptExec)			x
TargetPath Property	x		
Terminate Method (WshScriptExec)			x
<usage> Element			x
UserDomain Property	x		
UserName Property	x		
Version Property	x		
WindowStyle Property	x		
WorkingDirectory Property	x		
Write Method		x	
WriteBlankLines Method		x	
WriteLine Method		x	
WScript Object	x		
WshArguments Object	x		
WshController Object			x
WshEnvironment Object	x		
WshNamed Object			x
WshNetwork Object	x		
WshRemote Object			x
WshRemoteError Object			x
WshScriptExec Object			x
WshShell Object	x		
WshShortcut Object	x		
WshSpecialFolders Object	x		
WshUnnamed Object			x
WshUrlShortcut Object	x		

Build: Topic Version 5.6.9309.1546

Windows Script Host

Windows Script Host Basics

Microsoft® Windows® Script Host (WSH) is a language-independent scripting host for Windows Script compatible scripting engines. It brings simple, powerful, and flexible scripting to the Windows 32-bit 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.

In the Section

© 2001 Microsoft Corporation. All rights reserved.

Build: Topic Version 5.6.9309.1546

Windows Script Host

What Is WSH?

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 noninteractive scripting needs, such as logon scripting, administrative scripting, and machine automation.

WSH Objects and Services

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

Where Is WSH?

Windows Script Host is built into Microsoft Windows 98, 2000, and Millennium Editions. If you are running Windows 95, you can download Windows Script Host 5.6 from the Microsoft Windows Script Technologies Web site (http://msdn.microsoft.com/scripting).

> **Note** 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.

See Also

Windows Script Host Object Model | CreateObject | GetObject

Build: Topic Version 5.6.9309.1546

Windows Script Host

Scripts and Automating Windows

Computers are wonderful tools that perform repetitive tasks. But what if you need to perform a series of repetitive tasks? The answer lies in scripting.

What Is a Script?

A script is a program written in a scripting language, such as JScript and VBScript. Alternative script languages include Rexx, Python, and Perl. When compared to programming languages such as C++ and Visual Basic, scripting languages are better suited to creating short applications that provide quick solutions to small problems.

Automating Windows

In many cases, scripts are used to automate manual tasks, much like a macro. Scripts are well suited for:

- Manipulating the Windows environment
- Running other programs
- Automating logon procedures
- Sending key sequences to an application

For example, if you have several similar tasks, you can write one generalized script that can handle all of them.

You can write scripts that start an action in response to an event. You can write scripts that keep a running tally of events and trigger some

action only when certain criteria are met.

Scripts are also useful for nonrepetitive tasks as well. If a task requires you to do many things in sequence, you can turn that sequence of tasks into just one task by scripting it.

See Also

Types of Script Files

Build: Topic Version 5.6.9309.1546

Windows Script Host

Types of Script Files

Stand-alone scripts come in several varieties, and each has its own extension. The following table is a list of some common types.

Extension	Script Type	Description
.bat	MS-DOS batch file	MS-DOS operating system batch file
.asp	ASP page	Active Server Page file
.html	HTML file	Web page
.js	JScript file	Windows script
.vbs	VBScript file	Windows script
.wsf	Windows Script Host file	Container or project file for a Windows script; supported by WSH 2.0 and later.
.wsh	Windows Script Host files	Property file for a script file; supported by WSH 1.0 and later.

Each script type is suited to different application needs, and each has strengths and weaknesses. The script type you choose depends on your needs.

Still, there are certain scenarios where you could divide your overall problem into several smaller parts, writing a separate script for each part with each script written in the most suitable scripting language.

This is where Windows Script Host files (WSF files) are useful. WSF files may include other script files as part of the script. Consequently, multiple WSF files can reference libraries of useful functions, which may be created and stored in a single place.

See Also

Windows Script Host Object Model

Build: Topic Version 5.6.9309.1546

Windows Script Host

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 case of a script embedded in an HTML page, the engine component that interprets and runs the script code is loaded by the Web browser, such as Internet Explorer. In the case of a script embedded in an ASP page, the engine that interprets and runs the script code is built into Internet Information Services (IIS).

Windows Script Host executes scripts that exist outside an HTML or ASP page and that stand on their own as text files.

Available Script Engines

Generally, you write scripts in either Microsoft JScript or VBScript, the two script engines that ship with Microsoft Windows 98, 2000 and Millennium Editions. You can use other script engines, such as Perl, REXX, and Python, with Windows Script Host.

> **Note** For more information, see Microsoft Developer Network (MSDN) (http://msdn.microsoft.com/workshop/languages/clinic/vbsvjs.asp).

A stand-alone script written in JScript has the .js extension; a stand-alone script written in VBScript has the .vbs extension. These extensions are registered with Windows. When you run one of these types of files, Windows starts Windows Script Host, which invokes the associated script engine to interpret and run the file.

Note If you need to run another engine, that engine must be registered properly.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Creating Scripts that Can Be Used by WSH

A Windows script is a text file. You can create a script with any text editor as long as you save your script with a WSH-compatible script extension (.js, vbs, or .wsf).

The most commonly available text editor is already installed on your computer — Notepad. You can also use your favorite HTML editor, Microsoft Visual C++, or Visual InterDev.

To create a script with Notepad

1. Start Notepad.
2. Write your script. For example purposes, type `WScript.Echo("Hello World!");`
3. Save this text file with a .js extension (instead of the default .txt extension). For example, `Hello.js`.
4. Navigate to the file you just saved, and double-click it.
5. Windows Script Host invokes the JScript engine and runs your script. In the example, a message box is displayed with the message "Hello World!"

Build: Topic Version 5.6.9309.1546

Windows Script Host

Dividing Scripts into Reusable Parts

To simplify your script writing, you can divide a script into more than one part. With this approach, you would create a .wsf file and use it as the starting point of execution. The other parts could be .js or .vbs files. You would reference these files from the .wsf file.

This approach makes your code more robust because it isolates pieces of it, allowing you to debug one piece at a time. It also makes your code reusable because it allows you to create functions that can be called again and again.

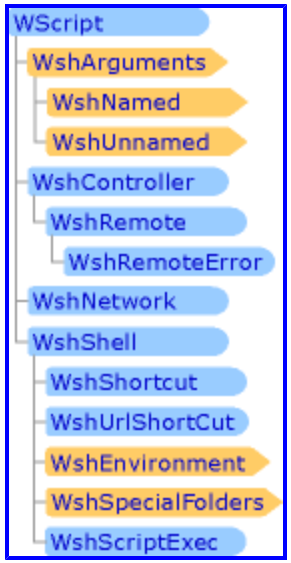
Build: Topic Version 5.6.9309.1546

Windows Script Host

Windows Script Host Object Model

The Windows Script Host object model consists of 14 objects. The root object is the **WScript** object.

The illustration that follows represents the Windows Script Host Object Model hierarchy. Click an object in the diagram to see its associated Help topic.



The Windows Script Host object model provides a logical, systematic way to perform many administrative tasks. The set of COM interfaces it provides can be placed into two main categories:

- Script Execution and Troubleshooting

  This set of interfaces allows scripts to perform basic manipulation of the Windows Script Host, output messages to the screen, and perform basic COM functions such as **CreateObject** and **GetObject**.

- Helper Functions

  Helper functions are properties and methods for performing actions, such as mapping network drives, connecting to printers, retrieving and modifying environment variables, and manipulating registry keys. Administrators can also use the Windows Script Host helper functions to create simple logon scripts.

WSH Objects and Associated Tasks

The following table is a list of the WSH objects and the typical tasks associated with them.

Object	What you can do with this object

Wscript	• Set and retrieve command line arguments • Determine the name of the script file • Determine the host file name (wscript.exe or cscript.exe) • Determine the host version information • Create, connect to, and disconnect from COM objects • Sink events • Stop a script's execution programmatically • Output information to the default output device (for example, a dialog box or the command line)
WshArguments	Access the entire set of command-line arguments
WshNamed	Access the set of named command-line arguments
WshUnnamed	Access the set of unnamed command-line arguments
WshNetwork	• Connect to and disconnect from network shares and network printers • Map and unmap network shares • Access information about the currently logged-on user
WshController	Create a remote script process using the Controller method **CreateScript**()
WshRemote	• Remotely administer computer systems on a computer network • Programmatically manipulate other programs/scripts
WshRemote Error	Access the error information available when a remote script (a WshRemote object) terminates as a result of a script error
WshShell	• Run a program locally • Manipulate the contents of the registry • Create a shortcut • Access a system folder • Manipulate environment variables (such as WINDIR, PATH, or PROMPT)
WshShortcut	Programmatically create a shortcut
WshSpecialfolders	Access any of the Windows Special Folders
WshURLShortcut	Programmatically create a shortcut to an Internet resource
WshEnvironment	Access any of the environment variables (such as WINDIR, PATH, or PROMPT)
WshScriptExec	Determine status and error information about a script run with Exec() Access the StdIn, StdOut, and StdErr channels

In addition to the object interfaces provided by Windows Script Host, administrators can use any ActiveX control that exposes automation interfaces to perform various tasks on the Windows platform. For example, administrators can write scripts to manage the Windows Active Directory Service Interface (ADSI).

Build: Topic Version 5.6.9309.1546

Windows Script Host

Running Your Scripts

There are three ways to run your scripts.

- In the Windows environment, double-click the icon of the script (you run script files the same way you run regular executable files).
- In the Windows environment, click the **Start** button, and then click **Run.** In the **Open** field of the **Run** dialog box, type the full path of the script, and click **OK**.
- From the command line, type the name of the script.

See Also

Running Scripts with WScript.exe | Running Scripts with CScript.exe | What to Include to Run a Script | Drag and Drop Support

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Using Windows Script Files (.wsf)

A Windows script (*.wsf) file is a text document containing Extensible Markup Language (XML) code. It incorporates several features that

offer you increased scripting flexibility. Because Windows script files are not engine-specific, they can contain script from any Windows Script compatible scripting engine. They act as a container.

With .wsf files, you can take advantage of the following features as you create your scripts:

.wsf files support	You can
Include statements	Incorporate functions from VBScript or JScript files into your Windows Script Host project.
Multiple engines	Use more than one scripting language per file.
Type libraries	Add constants to your code.
Tools	Edit files with any XML editor.
Multiple jobs in one file	Store all of your code in a single location.

Include Statements

If you have .js and .vbs files from previous Windows Script Host projects, a .wsf file enables you to use them with Windows Script Host. A .wsf file encapsulates a library of functions that can in turn be used by multiple .wsf files.

The following example shows a .wsf file that includes a JScript file (fso.js), plus a VBScript function that calls a function (**GetFreeSpace**) in the included file. The contents of fso.js are also shown.

```
<job id="IncludeExample">
   <script language="JScript" src="FSO.JS"/>
   <script language="VBScript">
      ' Get the free space for drive C.
      s = GetFreeSpace("c:")
      WScript.Echo s
   <sScript>
</job>
```

The fso.js file contains the following:

```
function GetFreeSpace(drvPath) {
   var fs, d, s;
   fs = new ActiveXObject("Scripting.FileSystemObject");
   d = fs.GetDrive(fs.GetDriveName(drvPath));
   s = "Drive " + drvPath + " - " ;
   s += d.VolumeName;
```

```
   s += " Free Space: " + d.FreeSpace/1024 + " Kbytes";
   return s;
}
```

Multiple-Engine Support

Since one scripting language may not have all the functionality you need, Windows Script Host allows you to combine multiple languages in a single .wsf file. The following example shows a .wsf file that includes both VBScript and PerlScript code:

```
<job id="PERLandVBS">
   <script language="PerlScript">
      sub PerlHello {
         my $str = @_[0];
         $WScript->Echo($str);
      }
   </script>

   <script language="VBScript">
      WScript.Echo "Hello from VBScript"
      PerlHello "Hello from PERLScript"
   </script>
</job>
```

Type Library Support

In the following example, "MyComponent" was developed with Microsoft Visual Basic 5.0. "MyComponent" defines the constant *MyError* with the following statement.

```
Public Const MyError = "You are not using MyComponent correctly"
```

The type library is contained in `mycomponent.lib`, which is installed in `C:\MyComponent`.

```
<job id="IncludeExample">
   <reference progid="MyComponent.MyClass">
   <script language="VBScript">
      Dim MyVar
      Set MyVar = CreateObject("MyComponent.MyClass")
      Currentreturn = MyVar.MyMethod
      If Currentreturn = False then
         WScript.Echo MyError
```

```
      End If
   </script>
</job>
```

Tools Support

Since the .wsf file is in XML format, you can use any editor that supports XML to edit .wsf files. This includes text editors, such as Notepad.

Multiple Jobs in One File

Instead of keeping all your scripts in separate files, you can incorporate them all into one .wsf file and break them into several different jobs. You can then run each job separately using syntax similar to the following example, where "MyFirstJob" is the name of the job contained in the MyScripts.wsf file.

```
CScript //Job:MyFirstJob MyScripts.wsf
```

Build: Topic Version 5.6.9309.1546

Windows Script Host

WSH Drag and Drop Support

You can drag files onto a WSH script. The file names are translated into arguments on the command line. These file names can be displayed in a list, which you can use to manipulate files with any scripting object.

To display a script's argument list

1. Create a file and give it a name with a script extension (for example, **DragDrop.vbs**).
2. Add code to the script file, for example:

```
Set objArgs = WScript.Arguments
For I = 0 to objArgs.Count - 1
   WScript.Echo objArgs(I)
Next
```

3. Save the file to your hard disk.
4. Drag and drop any file or files onto your saved file. In the example, the file names are echoed back to the screen.

The number of files you can drag onto a script is limited by the your system's maximum command-line length. If the total number of characters in all file names being dragged exceeds this limit, the drag and drop operation fails.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Setting and Customizing Script Properties (.wsh)

You can record specific settings for each of your individual scripts by means of a Windows Script Host control (.wsh) file. The .wsh file is a text document in which you can customize execution of one or more of your scripts. It is created automatically when you set the properties for a supported script file.

If you create multiple .wsh files for a single script, you can tailor the way the script runs to the needs of specific groups or even individuals within an organization. For example, you could create a single logon script that is invoked by two different .wsh files that contain different settings and parameters.

When you double-click a .wsh file or run it from the command line, CScript.exe or WScript.exe reads the .wsh file to determine the specific settings that should be used to execute the script. CScript/WScript executes the original script, passing in the properties that are defined within the .wsh file.

To create a .wsh file for a given script

1. Right-click the script file in Windows Explorer.
2. Click **Properties** on the shortcut menu.
3. Choose the settings you want for the script.
4. Click **OK** or **Apply**.

   A .wsh file is created with the same name as the script file you selected.

The following example illustrates a typical .wsh file:

```
[ScriptFile]
Path=C:\WINNT\Samples\WSH\showprop.vbs
[Options]
Timeout=0
DisplayLogo=1
BatchMode=0
```

The path information in the `[ScriptFile]` section identifies the script file that is associated with the .wsh file. The keys in the `[Options]` section correspond to settings in the **Script** tab within the **Properties** dialog box.

> **Note** You must have the original script file present when executing the .wsh file. If the .wsh file fails to run the script, check the `Path=` information in the .wsh file to ensure that it points to the script you are attempting to run.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Running Scripts from the Command Prompt

Windows Script Host enables you to run scripts from the command prompt. CScript.exe provides command-line switches for setting script properties.

To run scripts using CScript.exe

- Type a command at the command prompt using the following syntax:

```
cscript [host options...] [script name] [script options and parameters]
```

Host Options enable or disable various Windows Script Host features. Host options are preceded by two slashes (//).**Script name** is the name of the script file with extension and necessary path information, for example, `d:\admin\vbscripts\chart.vbs`. **Script options and parameters** are passed to the script. Script parameters are preceded by a single slash (/).

Each parameter is optional; however, you cannot specify script options without specifying a script name. If you do not specify parameters, CScript displays the CScript syntax and the valid host parameters.

CScript Example

Several sample scripts, which are installed along with Windows Script Host, are also available for download at (http://msdn.microsoft.com/scripting).

Suppose, for the purposes of this example, that you have copied the Chart.vbs sample script to the following folder on your computer:

```
c:\sample scripts\chart.vbs
```

You can run the script with and without a logo as follows.

To run a script with or without a logo

1. Start the MS-DOS command prompt.
2. Enter the following commands at the command prompt (modify accordingly if your sample scripts are located in a different folder):

```
cscript //logo c:\"sample scripts"\chart.vbs
cscript //nologo c:\"sample scripts"\chart.VBScript
```

See Also

Running Scripts from Windows | What to Include to Run a Script | WScript.exe and CScript.exe Options

Build: Topic Version 5.6.9309.1546

Windows Script Host

Running Scripts from Windows

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.

On initial installation, the default host is WScript. To change it to CScript, type the following at the command line:

```
cscript //h:cscript
```

Or, to change it from Cscript to Wscript:

```
wscript //h:cscript
```

To run a script using the default engine:

1. Double click the script in Windows Explorer or on the desktop.
2. Click **Start**, select **Run**, and enter the script name.

    Note On Windows NT and Windows 2000 only, simply enter the script name on a command line.

To run a script using a particular engine:

- Right-click the script in Windows Explorer and select **Open** to run in WScript or **Open in MS-DOS Window** (Windows 9x) or **Open in Command Window** (Windows NT and Windows 2000) to run in CScript.

-or-

- Click **Start**, select **Run**, enter "cscript" or "wscript" followed by the script name.

-or-

- Enter "cscript" or "wscript" on the command line, followed by the script name.

To run scripts using WScript.exe

- Double-click files or icons. These can be files or icons listed in **My Computer**, Windows Explorer, the **Find** window, the **Start** menu, or on the desktop.

-or-

1. Click the **Start** button, and then click **Run.**
2. In the **Open** field, type the full path of the script, and then click **OK**. You can also type `WScript` followed by the full name and path of the script you want to run.

If you double-click a script file whose extension has not yet been associated with WScript.exe, the **Open With** dialog box appears and asks which program to use to open the file. Choose WScript and check **Always use this program to open this file** to register WScript as the default application for all files with that extension.

The WScript.exe and CScript.exe properties dialog box provides the following options:

Property	Description
Stop script after specified number of seconds.	Specifies the maximum number of seconds that a script can run. The default is no limit. CScript.exe equivalent: //T:nn
Display logo when script is executed in command console.	Displays a banner before running the script. This is the default. The opposite is //`nologo`. CScript.exe equivalent: //logo or //nologo

Using the WScript.exe **Properties** dialog box, you can set global scripting options for all scripts that WScript runs on the local machine. You

can also set options for individual scripts using a .wsf file.

See Also

Running Scripts from the Command Prompt | What to Include to Run a Script | WScript.exe and CScript.exe Options

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

WScript.exe and CScript.exe Options

For the most part, options listed in the following table are applicable to both WScript.exe and CScript.exe. Exceptions are noted.

Parameter	Description
//B	Batch mode; suppresses command-line display of user prompts and script errors. Default is Interactive mode.
//D	Turns on the debugger.
//E:engine	Executes the script with the specified script engine.
//H:CScript or //H:Wscript	Registers CScript.exe or WScript.exe as the default application for running scripts. If neither is specified, WScript.exe is assumed as the default.
//I	Default. Interactive mode; allows display of user prompts and script errors Opposite of Batch mode.
//Job:<JobID>	Runs the specified JobID from the .wsf file.
//logo	Default. Displays a banner. Opposite of `nologo`.
//nologo	Prevents display of an execution banner at run time. Default is `logo`.
//S	Saves the current command-line options for this user.
//T:nn	Enables time-out: the maximum number of seconds the script can run. The default is no limit. The //T parameter prevents excessive execution of scripts by setting a timer. When execution time exceeds the specified value, CScript interrupts the script engine using the **IActiveScript::InterruptThread** method and terminates the process.

//U	Used with Windows NT and Windows 2000 to force the command line output to be in Unicode. There is no way for CScript to determine whether to output in Unicode or ANSI; it defaults to ANSI.
//X	Launches the program in the debugger.
//?	Displays a brief description of and usage information for command parameters (the usage information).

Build: Topic Version 5.6.9309.1546

Windows Script Host

What to Include to Run a Script

The information that you type when you run a script by typing its name depends on which version of Windows you are running and on the method you use to run the script.

WSH Executable File	Windows Version	Include
Command prompt	Windows NT or 2000	Specify the script name without the file extension. Example: `myScript`. **Note** If you specify the WSH executable file name, you must also include the script's file extension. Example: `cscript myScript.wsf`.
Command prompt	Windows 9*x* or Millennium	Specify the script's file extension and precede the script name with the WSH executable filename. Example: `cscript myScript.wsf`
Run command from **Open** box	Windows NT, 2000, 9*x*, or Millennium	Specify the script's file extension.

See Also

Running Scripts with WScript.exe | Running Scripts with CScript.exe | Drag and Drop Support

Build: Topic Version 5.6.9309.1546

Windows Script Host

Setting up Remote WSH

Remote WSH, which is a new technology included in WSH 5.6, provides the ability to run a script on a remote machine or machines. With Remote WSH, the script is physically copied from the local machine to the remote machine before executing. In order to enable Remote WSH functionality, you must first set up the remote machine with the proper security settings. The steps below perform the tasks that enable Remote WSH.

> **Note** Both the remote and local machines must be running Windows NT 4 SP3 or greater in order to use Remote WSH.

To enable a machine to run remote scripts

1. Install WSH V5.6 on the machine. If you are using Windows 2001 or have installed Internet Explorer 6 or greater, WSH 5.6 has already been installed.

   > **Note** WSH 5.6 is available for download from the web at http://msdn.microsoft.com/scripting

2. Add yourself to the remote machine's Local Administrators group.
3. To enable Remote WSH, use Poledit.exe on the server.

   > **Note** An administrator who wants to enable Remote WSH must either acquire the Windows 2000 resource kit, or use http://msdn.microsoft.com/scripting to acquire the necessary windowsscript.adm file that contains the WSH settings. The windowsscript.adm file must be copied to the server that sets the gapplicabel group's policies. Although it is not necessary to copy the file to the server's \WINNT\INF directory, this is nonetheless where the default adm files are located.

   > **Note** For more information on Poledit.exe, see the Poledit.exe's online help system.

4. WSH should now be enabled on the machine. To test it, see Running Scripts Remotely.

See Also

Security and Windows Script Host | Running Scripts Remotely

Build: Topic Version 5.6.9309.1546

Windows Script Host

Basic Windows Script Host Tasks

This section contains several commonly used Windows Scripting Host 5.6 scripts, which demonstrate basic functions.

> **Note** The scripts presented in the following tasks are virtually the same for developers using JScript or VBScript. When applicable, differences between the scripting models are noted. In addition, each task is written in both JScript and VBScript.

In this Section

- Accessing Networks
- Creating an Automated Login Script
- Driving Applications
- Executing File Management Operations
- Managing Shortcuts
- Manipulating the System Registry
- Running Scripts Remotely
- Signing a Script
- WSH and Windows Management Instrumentation (WMI)

Build: Topic Version 5.6.9309.1546

Windows Script Host

Accessing Networks

With WSH you can easily access a network programmatically. The following tasks demonstrate some of these capabilities.

In this Section

Accessing Network Connections

Controlling Networked Printers

See Also

WSH Samples

© 2001 Microsoft Corporation. All rights reserved.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Accessing Network Connections

The Network object enables you to access information about your network. The following scripts demonstrate how to map a network drive. In the first step, the script creates a Network Object. Next, the **MapNetworkDrive** method, one of the Network object's methods, performs the mapping operation. The **MapNetworkDrive** method takes five arguments:

- The local drive assignment (I:, for example)
- The Universal Naming Convention (UNC) path to the mapped remote drive
- An optional Boolean indicating whether the drive will be persistently connected
- An optional user name if you want to use different credentials
- An optional password for use with the alternate user name

```
// JScript.
var net;
net = new ActiveXObject("WScript.Network");
net.MapNetworkDrive("I:", "\\\\computer2\\public","True","jdoe","jdoepassword");

' VBScript.
Dim net
Set net = CreateObject("WScript.Network")
net.MapNetworkDrive "I:", "\\computer2\public","True","jdoe","jdoepassword"
```

See Also

Accessing Networks

Build: Topic Version 5.6.9309.1546

Windows Script Host

Controlling Networked Printers

The **Network** object enables you to access printing devices on your network. The following scripts demonstrate the use of the **Network** object to control a network printer device.

Connecting to a Remote Printer

The following scripts demonstrate how to connect to a network shared printing device. In the first step, the script creates a **Network** Object. Next, the **AddWindowsPrinterConnection** method, one of the Network object's methods, performs the connection operation. The **AddWindowsPrinterConnection** method takes two parameters: the name you wish to call the printer and the Universal Naming Convention (UNC) path to the printing device.

```
// JScript.
var net;
net = new ActiveXObject("WScript.Network");
net.AddWindowsPrinterConnection("\\\\ServerName\\PrinterName");

' VBScript.
Dim net
Set net = CreateObject("WScript.Network")
net.AddWindowsPrinterConnection "\\ServerName\PrinterName"
```

Setting Default Printer

The following script demonstrates how to set the desired default printing device. In the first step, the script creates a **Network** Object. Next, the **SetDefaultPrinter** method, one of the Network object's methods, performs the operation. The **SetDefaultPrinter** method takes a single parameter, the name of the printer, which is either the local printer name or a remote printer name using the Universal Naming Convention (UNC) path to the printing device.

```
// JScript.
var net;
net = new ActiveXObject("WScript.Network");
net.SetDefaultPrinter("\\\\ServerName\\PrinterName");

' VBScript.
Dim net
Set net = CreateObject("WScript.Network")
net.SetDefaultPrinter "\\ServerName\PrinterName"
```

See Also

Accessing Networks | AddWindowsPrinterConnection Method | SetDefaultPrinter Method | RemovePrinterConnection Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Creating an Automated Login Script

With WSH you can create automated login scripts. The following example assumes that a company has two file servers (named "server1" and "server2"), and two print servers (named "printer1" and "printer2"). To balance usage of the servers, everyone whose login name starts with A - K goes to the first file and print server, and everyone whose login name starts with L - Z goes to the second one.

Note In Windows 9x, include a delay so user logon takes affect.

```
// JScript.
var oNet, sUser, cInitial, startTime;
oNet = new ActiveXObject("WScript.Network");
// Get the user name. On Windows 98 and Windows ME, the use may not be logged
// on when the script starts running; keep checking every 1/2 a
// second until they are logged on
sUser = oNet.UserName;
startTime = new Date();
while (sUser == "")
{
   var curTime = new Date();
   if (curTime – startTime > 30000) WScript.Quit();
   WScript.Sleep(500);
   sUser = oNet.UserName;
}
// Add a share for the "h" drive and the printer, based on the
// first letter of the user's name
cInitial = sUser.charAt(0).toUpperCase();
if (cInitial < "L")
{
   oNet.MapNetworkDrive("h:", "\\\\server1\\users\\" + sUser);
   oNet.AddWindowsPrinterConnection("\\\\printer1\\hp", "HP LaserJet 4");
}
else
{
   oNet.MapNetworkDrive("h:", "\\\\server2\\users\\" + sUser);
```

```
   oNet.AddWindowsPrinterConnection("\\\\printer2\\hp", "HP LaserJet 4");
}

' VBScript.

Option Explicit
Dim oNet, sUser, cInitial, startTime
' Helper object
Set oNet = CreateObject("WScript.Network")
' Get the user name. On Windows 9x, the use may not be logged
' on when the script starts running; keep checking every 1/2 a
' second until they are logged on.
sUser = oNet.UserName
startTime = Now
Do While sUser = ""
   If DateDiff("s", startTime, Now) > 30 Then Wscript.Quit
   Wscript.Sleep 500
   sUser = oNet.UserName
Loop
' Add a share for the "h" drive and the printer, based on the
' first letter of the user's name
cInitial = UCase(Left(sUser, 1))
If (cInitial < "L") Then
   oNet.MapNetworkDrive "h:", "\\server1\users\" & sUser
   oNet.AddWindowsPrinterConnection "\\printer1\hp", "HP LaserJet 4"
Else
   oNet.MapNetworkDrive "h:", "\\server2\users\" & sUser
   oNet.AddWindowsPrinterConnection "\\printer2\hp", "HP LaserJet 4"
End If
```

See Also

WSH Samples

Build: Topic Version 5.6.9309.1546

Windows Script Host

Driving Applications

With WSH you can start applications. The following scripts demonstrate some of these capabilities.

Creating a Local Server Application

Some applications, such as Microsoft Word, expose objects which can be accessed programmatically. The following script uses Word's spell checker.

```
// JScript.
var Word, Doc, Uncorrected, Corrected;
var wdDialogToolsSpellingAndGrammar = 828;
var wdDoNotSaveChanges = 0;
Uncorrected = "Helllo world!";
Word = new ActiveXObject("Word.Application");
Doc = Word.Documents.Add();
Word.Selection.Text = Uncorrected;
Word.Dialogs(wdDialogToolsSpellingAndGrammar).Show();
if (Word.Selection.Text.length != 1)
   Corrected = Word.Selection.Text;
else
   Corrected = Uncorrected;
Doc.Close(wdDoNotSaveChanges);
Word.Quit();

' VBScript.

Dim Word, Doc, Uncorrected, Corrected
Const wdDialogToolsSpellingAndGrammar = 828
Const wdDoNotSaveChanges = 0

Uncorrected = "Helllo world!"
Set Word = CreateObject("Word.Application")
Set Doc = Word.Documents.Add
Word.Selection.Text = Uncorrected
Word.Dialogs(wdDialogToolsSpellingAndGrammar).Show

If Len(Word.Selection.Text) <> 1 Then
```

```
Corrected = Word.Selection.Text
Else
   Corrected = Uncorrected
End If

Doc.Close wdDoNotSaveChanges
Word.Quit
```

Spawning Programs with Shell.Exec Command

The **Shell.Exec** command provides additional capability beyond the **Shell.Run** method. These abilities include:

- Improved environment variable passing
- Ability to access the standard streams of the executable

The following VBScript sample demonstrates how to use standard streams and the **Shell.Exec** command to search a disk for a file name that matches a regular expression.

First, here's a small script that dumps to **StdOut** the full path of every file in the current directory and below:

```
' VBScript.
'  MYDIR.VBS
Option Explicit
Dim FSO
Set FSO = CreateObject("Scripting.FileSystemObject")
DoDir FSO.GetFolder(".")
Sub DoDir(Folder)
   On Error Resume Next
   Dim File, SubFolder
   For Each File In Folder.Files
      WScript.StdOut.WriteLine File.Path
   Next
   For Each SubFolder in Folder.SubFolders
      DoDir SubFolder
   Next
End Sub
```

Next, this script searches **StdIn** for a pattern and dumps all lines that match that pattern to **StdOut**.

```
' MyGrep.VBS
```

```
Option Explicit
Dim RE, Line
If WScript.Arguments.Count = 0 Then WScript.Quit
Set RE = New RegExp
RE.IgnoreCase = True
RE.Pattern = WScript.Arguments(0)
While Not WScript.StdIn.AtEndOfStream
   Line = WScript.StdIn.ReadLine
   If RE.Test(Line) Then WScript.StdOut.WriteLine Line
WEnd
```

Together these two scripts do what we want — one lists all files in a directory tree and one finds lines that match a regular expression. Now we write a third program which does two things: it uses the operating system to pipe one program into the other, and it then pipes the result of that to its own **StdOut**:

```
// MyWhere.JS
if (WScript.Arguments.Count() == 0)
   WScript.Quit();
var Pattern = WScript.Arguments(0);
var Shell = new ActiveXObject("WScript.Shell");
var Pipe = Shell.Exec("%comspec% /c \"cscript //nologo mydir.vbs | cscript //nologo mygrep.vbs " + Pattern + "\"")
while(!Pipe.StdOut.AtEndOfStream)
   WScript.StdOut.WriteLine(Pipe.StdOut.ReadLine());
```

See Also

WSH Samples | Exec Method | Run Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Executing File Management Operations

With WSH you can easily create, copy, move, and delete files and folders programmatically. The following tasks demonstrate these capabilities.

In this Section

Copying Files and Folders

Mapping to a Special Folders

See Also

WSH Samples

Build: Topic Version 5.6.9309.1546

Windows Script Host

Copying Files and Folders

File system manipulation, such as copying files and folders, requires the use of the **File System Object** (**FSO**). The following scripts demonstrate the use of the FSO to copy both files and folders.

Copying Files

The following scripts demonstrate how to copy a file from one local folder to another. In the first step, the script creates a **File System Object**. The **CopyFile** method, a file system object method, performs the file copy operation. The **CopyFile** method takes two parameters, the source file and the destination.

```
// JScript.
var FSO = WScript.CreateObject("Scripting.FileSystemObject");
```

```
FSO.CopyFile("c:\\COMPlusLog.txt", "c:\\x\\");

' VBScript.
Dim FSO
Set FSO = CreateObject("Scripting.FileSystemObject")
FSO.CopyFile "c:\COMPlusLog.txt", "c:\x\"
```

Copying Folders

The following script demonstrates how to copy the contents of one local folder to another folder on the local machine.

> **Note** The destination folder must already exist for this method to succeed. For information on how to create a directory using WSH, see CreateFolder Method.

In the first step, the script creates a **File System Object**. The **CopyFolder** method, a file system object method, performs the folder copy operation. The **CopyFolder** method takes two parameters, the source folder and the destination.

```
// JScript.
var FSO = WScript.CreateObject("Scripting.FileSystemObject");
FSO.CopyFolder("c:\\x", "c:\\y");

' VBScript.
Dim FSO
Set FSO = CreateObject("Scripting.FileSystemObject")
FSO.CopyFolder "c:\x", "c:\y"
```

See Also

Executing File Management Operations | FileSystemObject | CopyFile Method | CopyFolder Method | CreateFolder Method | MoveFolder Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Mapping to a Special Folder

Mapping special folders, such as the My Documents desktop folder, requires the use of the **Shell** object. The following scripts demonstrate the use of the **Shell** object to create a shortcut to a folder on the desktop.

```
// JScript.
var Shell, DesktopPath, URL;
Shell = new ActiveXObject("WScript.Shell");
DesktopPath = Shell.SpecialFolders("Desktop");
URL = Shell.CreateShortcut(DesktopPath + "\\MSDN Scripting.url");
URL.TargetPath = "HTTP://MSDN.Microsoft.com/scripting/";
URL.Save();

' VBScript.
Dim Shell, DesktopPath, URL
Set Shell = CreateObject("WScript.Shell")
DesktopPath = Shell.SpecialFolders("Desktop")
Set URL = Shell.CreateShortcut(DesktopPath & "\MSDN Scripting.URL")
URL.TargetPath = "HTTP://MSDN.Microsoft.com/scripting/"
URL.Save
```

See Also

Executing File Management Operations

Build: Topic Version 5.6.9309.1546

Windows Script Host

Managing Shortcuts

With WSH you can easily create, delete, move, and copy shortcuts programmatically. The following tasks demonstrate some of these capabilities.

In this Section

See Also

WSH Samples

Build: Topic Version 5.6.9309.1546

Windows Script Host

Copying a Shortcut

Copying shortcuts requires the use of the **File System Object** (**FSO**). The following scripts demonstrate the use of the **File System Object** to copy shortcuts.

```
// JScript.
Shell = new ActiveXObject("WScript.Shell");
FSO = new ActiveXObject("Scripting.FileSystemObject");
DesktopPath = Shell.SpecialFolders("Desktop") + "\\MSDN Scripting url";
MyDocumentsPath = Shell.SpecialFolders("MyDocuments") + "\\"
```

```
FSO.CopyFile(DesktopPath, MyDocumentsPath);

' VBScript.
Set Shell = CreateObject("WScript.Shell")
Set FSO = CreateObject("Scripting.FileSystemObject")
DesktopPath = Shell.SpecialFolders("Desktop") + "\MSDN Scripting.url"
MyDocumentsPath = Shell.SpecialFolders("MyDocuments") + "\\"
FSO.CopyFile DesktopPath, MyDocumentsPath
```

See Also

Managing Shortcuts

Build: Topic Version 5.6.9309.1546

Windows Script Host

Creating a Shortcut

Creating shortcuts requires the use of the **File Shell** object. The following scripts demonstrate the use of the **File Shell** object to create shortcuts.

```
// JScript.
Shell = new ActiveXObject("WScript.Shell");
DesktopPath = Shell.SpecialFolders("Desktop");
link = Shell.CreateShortcut(DesktopPath + "\\test.lnk");
link.Arguments = "1 2 3";
link.Description = "test shortcut";
link.HotKey = "CTRL+ALT+SHIFT+X";
link.IconLocation = "foo.exe,1";
link.TargetPath = "c:\\blah\\foo.exe";
link.WindowStyle = 3;
link.WorkingDirectory = "c:\\blah";
link.Save();
```

```
' VBScript.
Set Shell = CreateObject("WScript.Shell")
DesktopPath = Shell.SpecialFolders("Desktop")
Set link = Shell.CreateShortcut(DesktopPath & "\test.lnk")
link.Arguments = "1 2 3"
link.Description = "test shortcut"
link.HotKey = "CTRL+ALT+SHIFT+X"
link.IconLocation = "foo.exe,1"
link.TargetPath = "c:\blah\foo.exe"
link.WindowStyle = 3
link.WorkingDirectory = "c:\blah"
link.Save
```

See Also

Managing Shortcuts

Build: Topic Version 5.6.9309.1546

Windows Script Host

Deleting a Shortcut

Deleting shortcuts requires the use of the **File System Object** (**FSO**). The following scripts demonstrate the use of the **File System Object** to delete shortcuts.

```
// JScript.
Shell = new ActiveXObject("WScript.Shell");
FSO = new ActiveXObject("Scripting.FileSystemObject");
DesktopPath = Shell.SpecialFolders("Desktop");
FSO.DeleteFile(DesktopPath + "\\test.lnk")

' VBScript.
```

```
Set Shell = CreateObject("WScript.Shell")
Set FSO = CreateObject("Scripting.FileSystemObject")
DesktopPath = Shell.SpecialFolders("Desktop")
FSO.DeleteFile DesktopPath & "\test.lnk"
```

See Also

Managing Shortcuts

Build: Topic Version 5.6.9309.1546

Windows Script Host

Moving a Shortcut

Moving shortcuts requires the use of the **File System Object** (**FSO**). The following scripts demonstrate the use of the **File System Object** to move shortcuts.

```
// JScript.

Shell = new ActiveXObject("WScript.Shell");
FSO = new ActiveXObject("Scripting.FileSystemObject");
DesktopPath = Shell.SpecialFolders("Desktop") + "\\test.lnk";
MyDocumentsPath = Shell.SpecialFolders("MyDocuments") + "\\test.lnk";
FSO.MoveFile(DesktopPath, MyDocumentsPath);
' VBScript.

Set Shell = CreateObject("WScript.Shell")
Set FSO = CreateObject("Scripting.FileSystemObject")
DesktopPath = Shell.SpecialFolders("Desktop") & "\test.lnk"
MyDocumentsPath = Shell.SpecialFolders("MyDocuments") & "\test.lnk"
FSO.MoveFile DesktopPath, MyDocumentsPath
```

See Also

Managing Shortcuts

Build: Topic Version 5.6.9309.1546

Windows Script Host

Manipulating the System Registry

With WSH you can manage the system registry. The following scripts demonstrate some of these capabilities.

```
// JScript.
Sh = new ActiveXObject("WScript.Shell");
key =  "HKEY_CURRENT_USER\\"
Sh.RegWrite( key + "WSHTest\\", "testkeydefault");
Sh.RegWrite( key + "WSHTest\\string1", "testkeystring1");
Sh.RegWrite( key + "WSHTest\\string2", "testkeystring2", "REG_SZ");
Sh.RegWrite( key + "WSHTest\\string3", "testkeystring3", "REG_EXPAND_SZ");
Sh.RegWrite( key + "WSHTest\\int", 123, "REG_DWORD");
WScript.Echo( Sh.RegRead(key + "WSHTest\\"));
WScript.Echo ( Sh.RegRead(key + "WSHTest\\string1"));
WScript.Echo ( Sh.RegRead(key + "WSHTest\\string2"));
WScript.Echo ( Sh.RegRead(key + "WSHTest\\string3"));
WScript.Echo ( Sh.RegRead(key + "WSHTest\\int"));
Sh.RegDelete(key + "WSHTest\\");

' VBScript.
Set Sh = CreateObject("WScript.Shell")
key =  "HKEY_CURRENT_USER\"
Sh.RegWrite key & "WSHTest\", "testkeydefault"
Sh.RegWrite key & "WSHTest\string1", "testkeystring1"
Sh.RegWrite key & "WSHTest\string2", "testkeystring2", "REG_SZ"
Sh.RegWrite key & "WSHTest\string3", "testkeystring3", "REG_EXPAND_SZ"
```

```
Sh.RegWrite key & "WSHTest\int", 123, "REG_DWORD"
WScript.Echo Sh.RegRead(key & "WSHTest\")
WScript.Echo Sh.RegRead(key & "WSHTest\string1")
WScript.Echo Sh.RegRead(key & "WSHTest\string2")
WScript.Echo Sh.RegRead(key & "WSHTest\string3")
WScript.Echo Sh.RegRead(key & "WSHTest\int")
Sh.RegDelete key & "WSHTest\"
```

See Also

WSH Samples

Build: Topic Version 5.6.9309.1546

Windows Script Host

Running Scripts Remotely

WSH 5.6 can run scripts that reside on remote systems. The following scripts demonstrate this capability. These scripts make the assumption that the files are located on a local machine directory called "c:\wsh5.6"; change the local path and the remote machine name as necessary.

After initially running RemoteTest.WSF on the local machine, there may be a small pause as DCOM verifies your identity. After you see the "Done" message, a file named "c:\beenhere.txt" on the remote machine indicates the time that you executed the command (from the remote computer's clock).

```
// JScript.
RemoteTest.WSF
------------------------------
<package>
<job>
<script language="JScript">
var oController = new ActiveXObject("WSHController");
var oProcess = oController.CreateScript("c:\\wsh5.6\\beenhere.wsf", "remmachine");
```

```
oProcess.Execute();
while (oProcess.Status != 2) WScript.Sleep(100);
WScript.Echo("Done");
</script>
</job>
</package>
------------------------------

BeenHere.WSF
------------------------------
<package>
<job>
<script language="JScript">
var fso = new ActiveXObject("Scripting.FileSystemObject");
var fout = fso.CreateTextFile("c:\\beenhere.txt", true);
fout.WriteLine(new Date);
fout.Close();
</script>
</job>
</package>
------------------------------

' VBScript.
RemoteTest.WSF
------------------------------
<package>
<job>
<script language="VBScript">
set oController = CreateObject("WSHController")
set oProcess = oController.CreateScript("c:\wsh5.6\beenhere.wsf", "remmachine")
oProcess.Execute
While oProcess.Status <> 2
   WScript.Sleep 100
WEnd
WScript.Echo "Done"
</script>
</job>
</package>
------------------------------

BeenHere.WSF
------------------------------
<package>
<job>
```

```
<script language="VBScript">
set fso = CreateObject("Scripting.FileSystemObject")
set fout = fso.CreateTextFile("c:\beenhere.txt", true)
fout.WriteLine Now
fout.Close
</script>
</job>
</package>
```

See Also

WSH Samples | Setting up Remote WSH | WshRemote Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Signing a Script

The following scripts demonstrate the creation of a signature, which is used in a verification process. The script uses the **Signer Object** and the **SignFile** method to create a digital signature.

```
// JScript.
<job>
<runtime>
   <named name="file" helpstring="the file to sign" required="true" type="string"/>
   <named name="cert" helpstring="the name of the signing certificate" required="true" type="string"/>
   <named name="store" helpstring="the name of the certificate store" required="false" type="string"/>
</runtime>
<script language="JScript">
   var Signer, File, Cert, Store;
   if (!(WScript.Arguments.Named.Exists("cert") && WScript.Arguments.Named.Exists("file")))
   {
      WScript.Arguments.ShowUsage();
```

```
      WScript.Quit();
   }
   Signer = new ActiveXObject("Scripting.Signer");
   File  = WScript.Arguments.Named("file");
   Cert  = WScript.Arguments.Named("cert");
   Store = WScript.Arguments.Named("store");
    Signer.SignFile(File, Cert, Store);
</script>
</job>

'VBScript
<job>
<runtime>
   <named name="file" helpstring="the file to sign" required="true" type="string"/>
   <named name="cert" helpstring="the name of the signing certificate" required="true" type="string"/>
   <named name="store" helpstring="the name of the certificate store" required="false" type="string"/>
</runtime>
<script language="VBScript">
   Dim Signer, File, Cert, Store
   If Not (WScript.Arguments.Named.Exists("cert")) And WScript.Arguments.Named.Exists("file")) Then
      WScript.Arguments.ShowUsage
      WScript.Quit
   End If
   Set Signer = CreateObject("Scripting.Signer")
   File  = WScript.Arguments.Named("file")
   Cert  = WScript.Arguments.Named("cert")
   Store = WScript.Arguments.Named("store")
    Signer.SignFile File, Cert, Store
</script>
</job>
```

See Also

WSH Samples | Verifying a Script | Signing a Script

Build: Topic Version 5.6.9309.1546

Windows Script Host

WSH and Windows Management Instrumentation (WMI)

With WSH you can easily use Windows Management Instrumentation (WMI). The following scripts demonstrate using WSH and WMI to retrieve a user's logon time via ADSI.

Note For more information on WMI, see the WMI SDK at (http://msdn.microsoft.com).

```
// JScript.
LoginProfiles = GetObject("winmgmts:").InstancesOf ("Win32_NetworkLoginProfile");
for(e = new Enumerator(LoginProfiles) ; !e.atEnd() ; e.moveNext())
{
   Profile = e.item();
   WScript.Echo(Profile.Name);
   WScript.Echo(Profile.LastLogon);
}

' VBScript.
Set LoginProfiles = GetObject("winmgmts:").InstancesOf ("Win32_NetworkLoginProfile")
   for each Profile in LoginProfiles
      WScript.Echo Profile.Name
      WScript.Echo Profile.LastLogon
   next
```

See Also

Basic Windows Script Host Tasks

Build: Topic Version 5.6.9309.1546

Windows Script Host

WSH Walkthrough

The following walkthrough describes how a typical Network Administrator or other IT professional might use WSH 5.6 to create procedures that accomplish useful tasks.

> **Note** The walkthrough is presented in VBScript. The process for creating these scripts is nearly the same for developers using VBScript or JScript.

During the course of this walkthrough, you will perform the following activities:

- Create a script that creates a common share on several remote machines and populate it with files.
- Create a script that creates a common Printing Device connection on several remote machines and establish it as the default printing device.

To complete the walkthrough, all remote machines must be properly configured to enable Remote WSH. For more information on enabling these security settings, see Setting up Remote WSH.

> **Note** The following code is from the sample included in this documentation. To view the entire sample, see WSH Network Administrator Sample Script.

Create Variables and Constants

To create the necessary variables and constants

1. In your text-scripting editor, enter the variables.

```
Dim FSO
Dim Services
Dim SecDescClass
Dim SecDesc
Dim Trustee
Dim ACE
Dim Share
Dim InParam
Dim Network
```

2. In your text-scripting editor, enter the constants, changing the values to reflect the UNC names and paths applicable to your network environment.

```
Const FolderName = "C:\Public"
Const AdminServer = "\\AdminMachine"
Const ShareName = "Pubs"
Const PrinterShare = "\\CorpPrinters\PrinterShare"
```

Connecting to a printer and setting it as default

To connect the machine to a common printing device

- In your text-scripting editor, enter the code that creates a printing device. This code uses the **Network** variable and **PrinterShare** constant initialized in the previous step.

```
Set Network = CreateObject("Wscript.Network")
Network.AddWindowsPrinterConnection PrinterShare
```

To set the machines default printing device

- In your text-scripting editor, enter the code that sets the default printing device. This code uses the **Network** variable and **PrinterShare** constant initialized in the first step.

```
Network.SetDefaultPrinter PrinterShare
```

Creating a common share, copying files to it, and sharing it

To create a common share on the machine

- In your text-scripting editor, enter the code that creates a **File System Object** (**FSO**) and creates a folder. The script verifies the existence of the folder. If the folder does not exist, the script creates it. This code uses the **FSO** variable and the **FolderName** constant initialized in the first step.

```
Set FSO = CreateObject("Scripting.FileSystemObject")
If Not FSO.FolderExists(FolderName) Then
   FSO.CreateFolder(FolderName)
End If
```

To copy files to the newly created folder

- In your text-scripting editor, enter the code that creates a **File System Object** (**FSO**) and copies files from your local machine to the remote machine. This code uses the **FSO** variable and the **FolderName** constant initialized in the first step.

```
Call FSO.CopyFile(AdminServer & "\Public\Images\*.*", FolderName)
```

To establish the newly created folder as a share with WMI

- In your text-scripting editor, enter the code that creates a share using Windows Management Instrumentation (WMI). The share is established on the folder generated above. The script first connects to WMI. Next, it sets the security impersonation level and the Windows NT privilege that lets you set Discretionary Access Control Lists (DACLs) and Security Access Control Lists (SACLs). Next, it creates a new security descriptor and sets up a couple of Access Control Entries (ACEs) for the new share. Finally, it creates a new share with the new security descriptor. This code uses the **Services**, **SecDescClass**, **SecDesc**, **Trustee**, **ACE**, **Share**, and **InParam** variables, and the **FolderName**, **AdminShare**, and **ShareName** constants initialized in the first step.

  > **Note** WMI is a powerful, sophisticated technology based on Web Based Enterprise Management (WBEM). WMI is primarily used for accessing and instrumenting management information in an enterprise environment. For more information on WMI, see Microsoft Windows Management Instrumentation: Background and Overview at (http://msdn.microsoft.com/library/default.asp?URL=/library/backgrnd/html/wmixwdm.htm).

```
Set Services = GetObject("WINMGMTS:{impersonationLevel=impersonate,(Security)}!" & AdminServer & "\ROOT\CIMV2
Set SecDescClass = Services.Get("Win32_SecurityDescriptor")
Set SecDesc = SecDescClass.SpawnInstance_()
Set Trustee = Services.Get("Win32_Trustee").SpawnInstance_
Trustee.Domain = Null
Trustee.Name = "EVERYONE"
Trustee.Properties_.Item("SID") = Array(1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0)
Set ACE = Services.Get("Win32_Ace").SpawnInstance_
ACE.Properties_.Item("AccessMask") = 2032127
ACE.Properties_.Item("AceFlags") = 3
ACE.Properties_.Item("AceType") = 0
ACE.Properties_.Item("Trustee") = Trustee
SecDesc.Properties_.Item("DACL") = Array(ACE)
Set Share = Services.Get("Win32_Share")
Set InParam = Share.Methods_("Create").InParameters.SpawnInstance_()
InParam.Properties_.Item("Access") = SecDesc
InParam.Properties_.Item("Description") = "Public Share"
InParam.Properties_.Item("Name") = ShareName
InParam.Properties_.Item("Path") = FolderName
```

```
InParam.Properties_.Item("Type") = 0
Share.ExecMethod_("Create", InParam)
```

Running the Completed Script

The sample included in this documentation contains a complete, executable script with all of the functionality above. See WSH Network Administrator Sample Script.

Before running the script, ensure that all remote machines have been properly configured to run remote scripts. This is accomplished with Poledit.exe on the server. For more information, see Setting up Remote WSH.

When running remote WSH, the script is copied to the remote machines. Once the remote machine's security settings have been verified and the script is successfully copied, a return indicates success or failure. If successful, the script is then executed on the remote machines. For more information on running a remote WSH script, see Running Scripts Remotely.

See Also

Setting up Remote WSH | Accessing Networks | Running Scripts Remotely

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

WSH Network Administrator Sample Script

This WSH network sample demonstrates how a typical network administrator may use a script on several remote machines on the network. The sample script performs useful administrative tasks including:

- Connecting the machines to a network printing device
- Setting the newly connected printing device as the default printer

- Creating a common public folder on the machines
- Copying files to the newly created folder
- Establishing the newly created folder as a share using Windows Management Instrumentation (WMI).

> **Note** WMI is a powerful, sophisticated technology based on Web Based Enterprise Management (WBEM). WMI is primarily used for accessing and instrumenting management information in an enterprise environment. For more information on WMI, see Microsoft Windows Management Instrumentation: Background and Overview at (http://msdn.microsoft.com/library/default.asp?URL=/library/backgrnd/html/wmixwdm.htm).

The Administrator must establish the necessary security settings on the remote machines. For more information, see Setting up Remote WSH. Next the administrator must copy and paste the sample into the scripting editor and change the constants to reflect the corresponding network paths and machine names. Finally the administrator can run the script.

To run this sample

1. Establish the necessary security settings on the remote machines.
2. Copy the AdminScript.vbs script below into your scripting text editor.
3. Change the constants to reflect your network paths and machine names.
4. Replace **remmachine** with the applicable remote machine name and run the script:

```
var oController = new ActiveXObject"WSHController"
var oProcess = oController.CreateScript "c:\MyLocalDir\\AdminScript.vbs", "remmachine"
oProcess.Execute()
while (oProcess.Status != 2)
     WScript.Sleep(100)
WScript.Echo"Done"
```

AdminScript.vbs Sample

```
' Remote WSH Admin Sample  AdminScript.vbs
'
' This sample code does a few common administrative tasks which a
' network administrator might want to do to a number of the machines
' on his or her network:  it creates a public directory, populates
' it with some files and shares the directory out.  It also sets
' up the machines default printer connection.

' Note that in the interests of keeping this example code small, error
' handling has been omitted.  Actual production code should use
```

```
' appropriate error handling as many of these operations could fail;
' the disks could run out of space, for instance.

Option Explicit

Dim FSO
Dim Services
Dim SecDescClass
Dim SecDesc
Dim Trustee
Dim ACE
Dim Share
Dim InParam
Dim Network

Const FolderName = "C:\Public"
Const AdminServer = "\\AdminMachine"
Const ShareName = "Pubs"
Const PrinterShare = "\\CorpPrinters\PrinterShare"

' First we add a printer to this machine and make it the default.

Set Network = CreateObject("Wscript.Network")
Network.AddWindowsPrinterConnection PrinterShare
Network.SetDefaultPrinter PrinterShare

' Next we create a folder and populate it with some files.

Set FSO = CreateObject("Scripting.FileSystemObject")
If Not FSO.FolderExists(FolderName) Then
   FSO.CreateFolder(FolderName)
End If

Call FSO.CopyFile(AdminServer & "\Public\Images\*.*", FolderName)

' Make the folder into a share using WMI
' See the WMI SDK for information on how this code works.

Set Services = GetObject("WINMGMTS:{impersonationLevel=impersonate,(Security)}!" & AdminServer & "\ROOT\CIMV2")
Set SecDescClass = Services.Get("Win32_SecurityDescriptor")
Set SecDesc = SecDescClass.SpawnInstance_()
Set Trustee = Services.Get("Win32_Trustee").SpawnInstance_
Trustee.Domain = Null
Trustee.Name = "EVERYONE"
```

```
Trustee.Properties_.Item("SID") = Array(1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0)
Set ACE = Services.Get("Win32_Ace").SpawnInstance_
ACE.Properties_.Item("AccessMask") = 2032127
ACE.Properties_.Item("AceFlags") = 3
ACE.Properties_.Item("AceType") = 0
ACE.Properties_.Item("Trustee") = Trustee
SecDesc.Properties_.Item("DACL") = Array(ACE)
Set Share = Services.Get("Win32_Share")
Set InParam = Share.Methods_("Create").InParameters.SpawnInstance_()
InParam.Properties_.Item("Access") = SecDesc
InParam.Properties_.Item("Description") = "Public Share"
InParam.Properties_.Item("Name") = ShareName
InParam.Properties_.Item("Path") = FolderName
InParam.Properties_.Item("Type") = 0
Share.ExecMethod_("Create", InParam)

' And we're done.
```

See Also

WSH Walkthrough | Accessing Networks | Setting up Remote WSH | Running Scripts Remotely

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Security and Windows Script Host

Windows Script Host, which is a flexible tool for automating Windows, can be dangerous in the hands of someone who is bent on wreaking havoc.

To prevent abuse of Windows Script Host without stifling its power, Windows Script Host 5.6 employs a new security model.

Script users can now verify the authenticity of a script before running it. Script developers can sign their scripts to prevent unauthorized modifications. Administrators can enforce strict policies that determine which users have privileges to run scripts locally or remotely.

In this Section

CryptoAPI Tools

Signing a Script

Signature Verification Policy

Verifying a Script

Software Restriction Policies

Build: Topic Version 5.6.9309.1546

Windows Script Host

CryptoAPI Tools

The CryptoAPI Tools are used to create and verify digital signatures in *.exe, *.cab, *.dll, *.ocx, and script (*js, *.vbs, and *.wsf) files.

CryptoAPI Tools are used to digitally sign files to be used with Microsoft® Authenticode®, and to view and manage certificates, certificate revocation lists (CRLs), and certificate trust lists (CTLs). For more information on CrptoAPI Tools, see the CryptoAPI Start Page at (http://msdn.microsoft.com/library/psdk/crypto/portalapi_3351.htm?RLD=290).

See Also

Security and Windows Script Host | Signing a Script | Verifying a Script | Signature Verification Policy

Build: Topic Version 5.6.9309.1546

Windows Script Host

Signing a Script

Signing a script writes a digital signature block of comments in a script. The signature, which contains encoded information about the identity of the author, also encapsulates encoded information about the script itself. Consequently, any attempt to change the script invalidates the signature.

Script signing is programmatically accomplished with the **Scripting.Signer** object's **SignFile** method.

```
<job>
<runtime>
   <named name="file" helpstring="the file to sign" required="true" type="string"/>
   <named name="cert" helpstring="the name of the signing certificate" required="true" type="string"/>
   <named name="store" helpstring="the name of the certificate store" required="false" type="string"/>
</runtime>
<script language="JScript">
   var Signer, File, Cert, Store;
   if (!(WScript.Arguments.Named.Exists("cert") && WScript.Arguments.Named.Exists("file")))
   {
      WScript.Arguments.ShowUsage();
      WScript.Quit();
   }
   Signer = new ActiveXObject("Scripting.Signer");
   File  = WScript.Arguments.Named("file");
   Cert  = WScript.Arguments.Named("cert");
   if (WScript.Arguments.Named.Exists("store"))
   {
      Store = WScript.Arguments.Named("store");
   }
   else
   {
```

```
      Store = "";
   }
    Signer.SignFile(File, Cert, Store);
</script>
</job>
```

Note In order to sign a script, you must have a valid certificate. Ask your Administrator about your certification policy or contact a commercial certification authority.

See Also

Security and Windows Script Host | Verifying a Script | Signature Verification Policy | WinTrust | Signing a Script

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Software Restriction Policies

Software Restriction Policies are trust policies, which are regulations set by an administrator to restrict scripts that are not fully trusted from performing unauthorized actions within the operating system.

The following three criteria are used by Software Restriction Policies in determining a trust level:

- Any signature information in the script
- The path from which the script is running
- File content

Software Restriction Policies define the following default containers:

- Domain Administrator

- Machine Administrator
- Machine User
- Guest User
- Denied

Additionally, you can define your own custom container and set your own policies.

See Also

Security and Windows Script Host

Build: Topic Version 5.6.9309.1546

Windows Script Host

Signature Verification Policy

In WSH, administrators have the choice to turn signature verification either on or off. If an administrator turns signature verification on, then the machine will only run scripts signed by trusted authorities. With signature verification turned on, there are two possible scenarios:

- If the trust can't be determined, then the user is prompted to confirm that the script should run.
- If the trust can't be determined, then the script does not run.

If an administrator turns signature verification off, the machine permits users to run any script.

In Windows 2000, the signature verification policy is set through the Local Security Policy editor. For more information on the Local Security Policy Editor and WSH settings, see the online Windows help system.

The signature verification policy registry key is located in the following hive:

\HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows Script Host\Settings\TrustPolicy

The key is set to one of the following REG_DWORD values:

- **0** Run all scripts
- **1** Prompt user if script is untrusted
- **2** Run only trusted scripts

See Also

Security and Windows Script Host | Signing a Script | Verifying a Script | WinTrust

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Verifying a Script

Verifying a script determines whether the script that you are about to run is from a trusted source. It also allows you to confirm the integrity of the script. WSH verifies scripts before it attempts to run them, but you may have your own reason to verify a script.

Script verification is accomplished programmatically with the **Signer** object's **VerifyFile** method.

The **VerifyFile** method:

- Verifies the validity of the signature.
- Verifies that the signature belongs to a person who is trusted in your Trusted Publishers List.
- Verifies that the script has not been changed since it was signed.

Note Although the **VerifyFile** method programmatically confirms the digital signature, you should always ensure that you

really trust the trusted roots in the Trusted Publishers List.

See Also

Security and Windows Script Host | Signing a Script | Signature Verification Policy | WinTrust

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Reference

In this Section

XML Elements
List of WSH XML Elements.
Objects
List of WSH Objects.
Properties
List of WSH Properties.
Methods
List of WSH Methods.
Events
List of WSH Events
Error Messages
List of WSH Error Messages.

Related Sections

WSH Basics
Learn the basics of WSH.

Build: Topic Version 5.6.9309.1546

Windows Script Host

XML Elements

In this Section

XML element that contains script to define the behavior of a Windows Script component.

Related Sections

WSH Reference
List of elements that make up WSH Reference.
WSH Basics
Learn the basics of WSH.

Build: Topic Version 5.6.9309.1546

Windows Script Host

<?job?> Element

Specifies attributes for error handling.

```
<?job error="flag" debug="flag" ?>
```

Arguments

error
A Boolean value. False is the default value for all attributes. Set to true to allow error messages for syntax or run-time errors in the Windows Script (.wsf) file.
Debug
A Boolean value. False is the default value for all attributes. Set to true to enable debugging. If debugging is not enabled, you will be unable to launch the script debugger for a Windows Script file.

Remarks

Although most Windows Script files normally run silently during production, you might find it useful to be notified of errors in the Windows

Script (.wsf) file as you are developing it.

Example

The following example incorporates two jobs into one .wsf file that uses two different scripting languages.

```
<package>
   <job id="DoneInVBS">
      <?job debug="true"?>
      <script language="VBScript">
         WScript.Echo "This is VBScript"
      </script>
   </job>

   <job id="DoneInJS">
      <?job debug="true"?>
      <script language="JScript">
         WScript.Echo("This is JScript");
      </script>
   </job>
</package>
```

See Also

<runtime> Element | <named> Element | <description> Element | <example> Element | <object> Element | <package> Element | <resource> Element | <?XML?> Element

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

<?XML?> Element

Indicates that a file should be parsed as XML.

```
<?XML version="version" [standalone="DTDflag"] ?>
```

Arguments

version
A string in the form *n.n* specifying the XML level of the file. Use the value 1.0.

DTDflag
Optional. Boolean value indicating whether the XML file includes a reference to an external Document Type Definition (DTD). Script component XML files do not include such a reference, so the value for this attribute is always "yes."

Remarks

This declaration must be the first element in the file and cannot be preceded by any blank lines.

The existence of this declaration puts the script component compiler into strict XML mode, where element types and attribute names are case-sensitive, attribute values are enclosed in single or double quotation marks, and all elements are parsed. If the declaration is not included, the compiler allows syntax that is less strict.

You should include this declaration and follow XML conventions if your script component file will be edited in an editor that supports XML.

Example

The following example demonstrates the use of the <?XML?> Element:

```
<?XML version="1.0" standalone="yes" ?>
```

See Also

<runtime> Element | <named> Element | <description> Element | <example> Element | <object> Element | <package> Element | <resource> Element | <?job?> Element

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

<description> Element

Marks the descriptive text that is displayed when the **ShowUsage** method is executed or when the script is run with the **/?** command line switch.

```
<description>
    This section describes the script
</description>
```

Remarks

Your description can be more than one line long.

Do not include quotes in your description if you do not want them to appear in the usage. All text between the description tags appears in the usage listing. This includes, among others, tabs, new lines, and special characters.

The **<description>** element is similar to the **<example>** element, except it appears at the start of the usage, whereas the **<example>** element appears at the end.

Example

The following script demonstrates the use of the <description> Element:

```
<runtime>
    <description>
        This script reboots a server
    </description>
    <!--...etc.....->
</runtime>
```

See Also

ShowUsage Method | <runtime> Element | <named> Element | <unnamed> Element | <example> Element

Build: Topic Version 5.6.9309.1546

Windows Script Host

<example> Element

Provides an example of usage for the job.

```
<example>
   Example text
</example>
```

Remarks

The <**example**> element is enclosed within the <**runtime**> element.

Example

The following script demonstrates the use of the <example> Element:

```
<job>
    <runtime>
        <description>This script reboots a server</description>
        <named
            name = "Server"
            helpstring = "Server to run the script on"
            type = "string"
            required = "true"
        />
        <example>Example: reboot.wsf /Server:scripting</example>
    </runtime>
</job>
```

Everything between <example> and </example> tags gets picked up, including new lines and extra white space. Calling the **ShowUsage** method from this script results in the following output:

This script reboots a server

Usage: reboot.wsf /server:value

Options:

server : Server to run the script on

Example:

reboot.wsf /server:scripting

See Also

ShowUsage Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

<job> Element

Marks the beginning and the end of a job within a Windows Script file (*.wsf).

```
<job [id=JobID]>
    job code
</job>
```

Arguments

JobID
Optional. Uniquely identifies the job within the scope of the Windows Script file.

Remarks

Each jobID within a Windows Script file must be unique.

Each script within a set of job tags is executed in sequence, from top to bottom.

A job contains one or more script blocks. A script block is any script code between a set of <script> tags. A script block can contain several scripts, and each script can be in a different scripting language.

To run a specific job or to run multiple jobs, use the //Job switch. If you specify more than one job, the jobs are executed in sequential order. (This is shown in the example below.). If you do not specify a job, only the first job is run. If you have two or more jobs in your Windows Script file, they must be enclosed in a <package> tag.

Example

The following script example is a Windows Script file called `myScript.wsf`. This file contains two separate jobs, each written in a different scripting language. The first job, written in VBScript, is given the identifier `DoneInVBS`. The second job, written in JScript, is given the identifier `DoneInJS`.

```
<package>
    <job id="DoneInVBS">
    <?job debug="true"?>
        <script language="VBScript">
           WScript.Echo "This is VBScript"
        </script>
    </job>
    <job id="DoneInJS">
    <?job debug="true"?>
        <script language="JScript">
           WScript.Echo("This is JScript");
        </script>
    </job>
</package>
```

To run the second job in the Windows Script file, myScript.wsf, type the following at the command prompt.

```
cscript myScript.wsf //job:DoneInJS
```

To run both jobs in `myScript.wsf`, type the following at the command prompt.

```
cscript myScript.wsf //job:DoneInVBS //job:DoneInJS
```

See Also

ShowUsage Method | <runtime> Element | <named> Element | <unnamed> Element | <description> Element | <example> Element

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

<named> Element

Describes a named argument for the script.

```
<named
   name = namedname
   helpstring = helpstring
   type = "string|boolean|simple"
   required = boolean
/>
```

Arguments

name
String that represents the name of the argument you are describing. Defines the argument at the command line and in the script.
helpstring

String that represents the help description for the argument. The WSH runtime provides the help description using the **ShowUsage** method or the /? argument.

type

Optional. Describes the type of argument, which defines how the argument will be parsed from the command line. The default value is `simple`.

required

Optional. A Boolean value that indicates whether an argument is required or not. Affects the display of the usage only.

Remarks

The <**named>** element is contained by (enclosed within) a set of **runtime** tags.

An argument with the name `server` would provide a `/server` argument at the command line as well as an argument named `server` in the **WSHNamed** arguments collection.

If the type is `string`, the argument is a string. The argument is passed to the script as `/named:stringvalue`.

If the type is `Boolean`, the argument is Boolean. The argument is passed to the script as `/named+` to turn it on, or `/named-` to turn it off.

If the type is `simple`, the argument takes no additional value and is passed as just the name, `/named`.

Example

The following script demonstrates the use of the <named> Element:

```
<job>
<runtime>
  <named
    name="server"
    helpstring="Server to access"
    type="string"
    required="true"
  />
  <named
    name="user"
    helpstring="User account to use on server. Default is current account."
    type="string"
    required="false"
  />
```

```
  <named
    name="enable"
    helpstring="If true (+), enables the action. A minus(-) disables."
    type="boolean"
    required="true"
  />
  <named
    name="verbose"
    helpstring="If specified, output will be verbose."
    type="boolean"
    required="false"
  />
</runtime>
<script language="JScript">
  WScript.Arguments.ShowUsage();
</script>
</job>

This will produce the following output when usage is shown:

Usage: example.wsf /server:value [/user:value] /enable[+|-] [/verbose]

Options:

server    : Server to access
user      : User account to use on server. Default is current account.
enable    : If true (+), enables the action. A minus(-) disables.
verbose   : If specified, output will be verbose.
```

See also

ShowUsage Method | <runtime> Element | <unnamed> Element | <description> Element | <example> Element

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

<object> Element

Defines objects in Windows Script (.wsf) files that can be referenced by a script.

```
<object id="objID" [classid="clsid:GUID" | progid="progID"] />
```

Arguments

objID
Required. A name that references the object in your script. Object ID values must begin with a letter and can include letters, digits, and underscores (_). The object ID must be unique throughout the scope of the Windows Script file.

GUID
Optional. The class ID (GUID) of the object.

progID
Optional. Program ID of the object, which can be specified as an alternative to the class ID.

Remarks

The **<object>** element provides a way to expose objects globally for use in scripting within the Windows Script file without using functions such as **CreateObject()**. Using an **<object>** element makes the object available with global scope and enables scripting tools to provide statement completion for the object's members.

You must specify either a `classid` or a `progid`.

Example

```
<job>
<obect id="fso" progid="Scripting.FileSystemObject"/>
<script language="Jscript">

var a = fso.CreateTextFile("c:\\testfile.txt", true);
a.WriteLine("This is a test.");
a.Close();

</script>
</job>
```

See Also

<runtime> Element | <named> Element | <description> Element | <example> Element | <package> Element

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

<package> Element

Encloses multiple job definitions in a Windows Script (.wsf) file.

```
<package>
   code for one or more jobs
</package>
```

Remarks

The **<package>** element is optional when a .wsf file contains only one job.

Example

The following example incorporates two jobs into one .wsf file that uses two different scripting languages:

```
<package>
   <job id="DoneInVBS">
   <?job debug="true"?>
      <script language="VBScript">
         WScript.Echo "This is VBScript"
      </script>
   </job>
```

```
   <job id="DoneInJS">
   <?job debug="true"?>
      <script language="JScript">
         WScript.Echo("This is JScript");
      </script>
   </job>
</package>
```

See Also

<runtime> Element | <named> Element | <description> Element | <example> Element | <object> Element

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

<reference> Element

Includes a reference to an external type library.

<reference [object="*progID*"|guid="*typelibGUID*"] [version="*version*"] **/>**

Arguments

progID
Program ID from which the type library can be derived. It can include either a version number (for example, ADO.Recordset.2.0), the explicit program ID of a type library, or the program ID of the executable (such as a .DLL) that incorporates the type library. If you use the object attribute, you do not need to specify a version attribute because the version can be inferred from the program ID. If the object attribute is specified, you cannot also specify a GUID attribute.

typelibGUID
The GUID of the type library to reference. If the GUID attribute is specified, you cannot also specify an object attribute.

version

Optional. Version number of the type library to use. It must be in the form <major version>[.<minor version>]. If a version is not specified, the default version is 1.0. If the object attribute is used to specify the type library and the version is not specified, the version is derived from the registry key for the specified program ID. If none can be found, the default is 1.0.

Remarks

Referencing a type library in your Windows Script (.wsf) file enables you to use constants defined in the type library in your scripts. The **<reference>** element looks up and makes available the type library associated with a specific program ID or type library name. Type library information can be available in .tlb, .olb, or .dll files.

See Also

<runtime> Element | <named> Element | <description> Element | <example> Element | <object> Element | <package> Element

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

<resource> Element

Isolates textual or numeric data that should not be hard-coded into a script.

```
<resource id="resourceID">
   text or number
</resource>
```

Arguments

resourceID
Unique identifier for the resource within the script.

Remarks

The **<resource>** element allows you to isolate strings or numbers in your Windows Script (.wsf) file that you want to reference in your scripts. For example, resource elements are typically used to maintain strings that may be localized into other languages.

To get the value of a resource, call the **getResource** method, passing it the ID of the resource you want to use.

Everything within the elements gets used including white space (tabs, new lines, etc.).

Example

The following code defines a resource (`errNonNumeric`) and returns its value if the variable `upperBound` is not numeric.

```
<resource id="errNonNumeric">
   Non-numeric value passed
</resource>

<script language="VBScript">
   <![CDATA[
      Function getRandomNumber(upperBound)
         If IsNumeric(upperBound) Then
            getRandomNumber = CInt(upperBound * Rnd + 1)
         Else
            getRandomNumber=getResource("errNonNumeric")
         End If
      End Function
   ]]>
</script>
```

See Also

<runtime> Element | <named> Element | <description> Element | <example> Element | <object> Element | <package> Element | getResource Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

<runtime> Element

Groups together the set of run-time arguments for a script.

```
<runtime>
    <named attributes etc. />
    <unnamed attributes etc. />
    <example>Example Text</example>
    …
</runtime>
```

Remarks

The **ShowUsage** method uses the information enclosed by the <**runtime**> element to display the runtime parameters for a script.

Since the <**runtime**> element is enclosed within a set of **job** tags, the defined run-time arguments apply to that job only.

> **Note** With version 5.6, the data enclosed by the **<runtime>** element is used only for self-documentation and to format the data displayed by ShowUsage. The **<runtime>** element does not enforce the values set for the arguments it contains (i.e. a "required" argument that is missing from the command line does not cause an error). If the **<runtime>** element is included in a .wsf file, then running the script with the "/?" argument will show the usage and quit.

Example

The following script demonstrates the use of the <runtime> Element:

```
<job>
    <runtime>
        <named
            name="server"
            helpstring="The server to run the script on"
            type="string"
            required="true"
        />
```

```
    </runtime>
<script language="JScript">
  if (!WScript.Arguments.Named.Exists("server"))
  {
    WScript.Arguments.ShowUsage();
  }
// ... some script here
</script>
</job>
```

See Also

ShowUsage Method | <named> Element | <unnamed> Element | <description> Element | <example> Element

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

<script> Element

Contains script that defines the behavior of a Windows Script (.wsf) file.

```
<script language="language" [src="strFile"]>
     script here
</script>
```

Arguments

language
The name of the scripting language, such as VBScript or JScript, used in the script block.
strFile
The name of the script file to include into the script block.

Remarks

If XML validation is not enabled, the XML parser ignores all lines inside the <script> element. However, if XML validation is enabled by including the <?XML?> element at the top of the Windows Script (.wsf) file, the XML parser can misinterpret greater than (>), less than (<), and other symbols used in script as XML delimiters.

If you are creating a file that conforms closely to XML syntax, you must ensure that characters in your script element are not treated as XML-reserved characters. To do this, enclose the actual script in a <![CDATA[...]]> section. This applies to all data blocks - <example>, <description>, and <resource>. All may need CDATA markers if <?XML?> is specified and if they include XML-reserved characters.

> **Note** Do not include a CDATA section unless you also include the <?XML?> declaration.

Example

The following example incorporates two jobs into one .wsf file, using two different scripting languages:

```
<package>
   <job id="DoneInVBS">
   <?job debug="true"?>
      <script language="VBScript">
         WScript.Echo "This is VBScript"
      </script>
   </job>

   <job id="DoneInJS">
   <?job debug="true"?>
      <script language="JScript">
         WScript.Echo("This is JScript");
      </script>
   </job>
</package>
```

See Also

<runtime> Element | <named> Element | <description> Element | <example> Element | <object> Element | <package> Element | <resource> Element | <?XML?> Element

Build: Topic Version 5.6.9309.1546

Windows Script Host

<unnamed> Element

Describes an unnamed argument for the script.

```
<unnamed
   name       = unnamedname
   helpstring = helpstring
   many       = boolean
   required   = boolean or integer
/>
```

Arguments

name
The string that is used in the usage to represent this argument. This value is not used elsewhere.

helpstring
String that represents the help description for the argument. The WSH runtime provides the help description using the **ShowUsage** method or the /? argument.

many
Optional. Boolean value. If true, then this argument may be repeated more times than specified by the required attribute. Otherwise, the required attribute represents the exact number of the argument that is required. See the example below for more details.

required
`Optional. An integer value indicating how many times this argument should appear in the command line.`

Remarks

The <**unnamed**> element is contained by (enclosed within) a set of **runtime** tags.

An argument with the name `server` would provide a `/server` argument at the command line and an argument named `server` in the **WSHNamed** arguments collection.

Note The name attribute of the unnamed element is just for display purposes.

When setting the "required" attribute, a Boolean value will be converted to an integer; "true" becomes 1 and "false" becomes 0.

Example

Here are a couple of examples of how the various attributes affect the usage with unnamed elements. First, a simple case:

```
<runtime>
<unnamed
  name="filename"
  helpstring="The file to process"
  many="false"
  required="true"
</>
</runtime>
```

This would produce the following:

```
Usage: example.wsf filename
```

Options:

filename : The file to process

Change it to:

```
<runtime>
<unnamed
  name="filename"
  helpstring="The files to process"
  many="false"
  required="3"
</ >
</runtime>
```

and the output changes to:

Usage: example.wsf filename1 filename2 filename3

Options:

filename : The files to process

The many switch will display ellipses to indicate you can enter more files than indicated. If the example changes to:

```
<runtime>
<unnamed
  name="filename"
  helpstring="The file(s) to process"
  many="true"
  required="1"
</>
</runtime>
```

then the output changes to:

Usage: example.wsf filename1 [filename2...]

Options:

filename: The file to process.

See also

ShowUsage Method | <runtime> Element | <named> Element | <description> Element | <example> Element

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

<usage> Element

Allows user to override default usage display.

```
<usage>
    This section describes the script
</usage>
```

Remarks

The <usage> element is similar to the <example> and <description> elements — it contains text that the developer can include in the usage. The difference is that if a <usage> element exists, everything else in the <runtime> element is ignored, and only the text in the <usage> element is displayed. This is so you can completely override the usage display.

> **Note** The <usage> Element should always be enclosed by a <runtime> Element.

Example

The following script demonstrates the use of the <usage> Element:

```
<job>
<runtime>
<named name="arg1" helpstring="the first arg"/>
<usage>
Your usage text goes here.
</usage>
</runtime>
<script language="vbscript">
WScript.Arguments.ShowUsage
</script>
</job>
```

This produces the following:

```
Your usage text goes here.
```

See Also

<runtime> Element | <named> Element | <description> Element | <example> Element | <package> Element | <resource> Element | <?XML?> Element

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Objects

In this Section

WScript Object
: Provides access to most of the objects, methods, and properties in the WSH object model.

WshArguments Object
: Gives you access to the entire collection of command-line parameters — in the order in which they were originally entered.

WshController Object
: Exposes the method **CreateScript**() that creates a remote script process.

WshEnvironment Object
: Gives you access to the collection of Microsoft Windows system environment variables.

WshNamed Object
: Provides access to the named command-line script arguments within the **WshArguments** object.

WshNetwork Object
: Gives you access to the shared resources on the network to which your computer is connected.

WshRemote Object
: Provides access to the remote script process.

WshRemoteError Object
: Exposes the error information available when a remote script (a **WshRemote** object) terminates as a result of a script error.

WshScriptExec Object
: Provides status and error information about a script run with **Exec**, along with access to the stdIn, stdOut, and stdErr channels.

WshShell Object
: Gives you access to the native Windows shell functionality.

WshShortcut Object
Allows you to create a shortcut programmatically.
WshSpecialFolders Object
Allows you to access the Windows Special Folders.
WshUnnamed Object
Provides access to the unnamed command-line script arguments within the **WshArguments** object.
WshUrlShortcut Object
Allows you to create a shortcut to an Internet resource, programmatically.

Related Sections

WSH Reference
List of elements that make up WSH Reference.
WSH Basics
Learn the basics of WSH.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Scripting.Signer Object

The Scripting.Signer object enables an author to sign a script with a digital signature and a recipient to verify the signature's authenticity and trustworthiness.

Remarks

The **Scripting.Signer** object requires a valid certificate.

Example

The following JScript code shows the **Scripting.Signer** object digitally signing a script.

```
<job>
<runtime>
   <named name="file" helpstring="the file to sign" required="true" type="string"/>
   <named name="cert" helpstring="the name of the signing certificate" required="true" type="string"/>
   <named name="store" helpstring="the name of the certificate store" required="false" type="string"/>
</runtime>
<script language="JScript">
   var Signer, File, Cert, Store = "my";
   if (!(WScript.Arguments.Named.Exists("cert") && WScript.Arguments.Named.Exists("file")))
   {
      WScript.Arguments.ShowUsage();
      WScript.Quit();
   }
   Signer = new ActiveXObject("Scripting.Signer");
   File  = WScript.Arguments.Named("file");
   Cert  = WScript.Arguments.Named("cert");
   if (WScript.Arguments.Named.Exists("store"))
   {
      Store = WScript.Arguments.Named("store");
   }

    Signer.SignFile(File, Cert, Store);
</script>
</job>
```

See Also

Signing a Script | Sign Method | SignFile Method | Verify Method | VerifyFile Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Scripting.Signer Object Methods

Methods

Sign Method

SignFile Method

Verify Method

VerifyFile Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WScript Object

Provides access to root object for the Windows Script Host object model.

Remarks

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

- command-line arguments,
- the name of the script file,

- the host file name,
- and host version information.

The **WScript** object allows you to:

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

The **WScript** object can be used to set the mode in which the script runs (either interactive or batch).

Example

Since the WScript object is the root object for the Windows Script Host object model, many properties and methods apply to the object. For examples of specific syntax, visit the properties and methods links.

Properties

Arguments Property | FullName Property (WScript Object) | Interactive Property | Name Property | Path Property | ScriptFullName Property | ScriptName Property | StdErr Property | StdIn Property | StdOut Property | Version Property

Methods

CreateObject Method | ConnectObject Method | DisconnectObject Method | Echo Method | GetObject Method | Quit Method | Sleep Method

See Also

Running Your Scripts | WshShell Object | WshNetwork Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

WScript Object Properties and Methods

Properties

Arguments Property

FullName Property

Name Property

Path Property

ScriptFullName Property

ScriptName Property

StdErr Property

StdIn Property

StdOut Property

Version Property

Methods

CreateObject Method

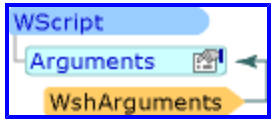
ConnectObject Method

DisconnectObject Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshArguments Object



Provides access to the entire collection of command-line parameters — in the order in which they were originally entered.

Remarks

The **WshArguments** object is a collection returned by the **WScript** object's **Arguments** property *(WScript.Arguments)*. Two of the **WshArguments** object's properties are filtered collections of arguments — one contains the named arguments (querying this property returns a **WshNamed** object), the other contains the unnamed arguments (querying this property returns a **WshUnnamed** object). There are three ways to access sets of command-line arguments.

- You can access the entire set of arguments (those with and without names) with the **WshArguments** object.
- You can access the arguments that have names with the **WshNamed** object.
- You can access the arguments that have no names with the **WshUnnamed** object.

Example

The following code displays the command-line parameters in the **WshArguments** object.

[VBScript]

```
Set objArgs = WScript.Arguments
For I = 0 to objArgs.Count - 1
   WScript.Echo objArgs(I)
Next
```

[JScript]

```
objArgs = WScript.Arguments;
for (i = 0; i < objArgs.length; i++)
{
   WScript.Echo(objArgs(i));
}
```

Properties

Item Property | Length Property (WshArguments object) | Count Property | Named Property | Unnamed Property

Methods

Count Method | ShowUsage Method

See Also

Arguments Property | WshNamed Object | WshUnnamed Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshArguments Object Properties and Methods

Properties

Item Property

Length Property

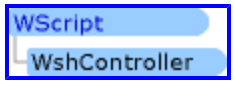
Methods

Count Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshController Object



Provides access to the **CreateScript**() method (for creating a remote script process).

Example

The following example uses a controller object to create a **WshRemote** object.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("remote1.js")
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}
```

Methods

CreateScript Method

See Also

CreateObject Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshController Object Methods

Methods

CreateScript Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshEnvironment Object

Provides access to the collection of Windows environment variables.

Remarks

The **WshEnvironment** object is a collection of environment variables that is returned by the **WshShell** object's **Environment** property. This collection contains the entire set of environment variables (those with names and those without). To retrieve individual environment variables (and their values) from this collection, use the environment variable name as the index.

Example

The following code displays an environment variable.

[VBScript]

```
Set WshShell = WScript.CreateObject("WScript.Shell")
Set WshSysEnv = WshShell.Environment("SYSTEM")
WScript.Echo WshSysEnv("NUMBER_OF_PROCESSORS")
```

[JScript]

```
var WshShell = WScript.CreateObject("WScript.Shell");
var WshSysEnv = WshShell.Environment("SYSTEM");
WScript.Echo(WshSysEnv("NUMBER_OF_PROCESSORS"));
```

Properties

Item Property | Length Property (WshEnvironment object)

Methods

Count Method | Remove Method

See Also

Environment Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshEnvironment Object Properties and Methods

Properties

Item Property

Length Property

Methods

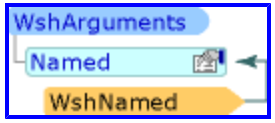
Count Method

Remove Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshNamed Object



Provides access to the named arguments from the command line.

Remarks

The **Named** property of the **WshArguments** object returns the **WshNamed** object, which is a collection of arguments that have names. This collection uses the argument name as the index to retrieve individual argument values. There are three ways to access sets of command-line arguments.

- You can access the entire set of arguments (those with and without names) with the **WshArguments** object.
- You can access the arguments that have names with the **WshNamed** object.
- You can access the arguments that have no names with the **WshUnnamed** object.

Example

The following code displays the number of named and unnamed command-line arguments.

```
<package>
<job id="JS">
<script language="JScript">

var argsNamed = WScript.Arguments.Named;
var argsUnnamed = WScript.Arguments.Unnamed;

WScript.Echo("There are " + argsNamed.length + " named arguments.");
WScript.Echo("There are " + argsUnnamed.length + " unnamed arguments.");

</script>
</job>

<job id="VBS">
<script language="VBScript">

Dim argsNamed, argsUnnamed
Set argsNamed = WScript.Arguments.Named
Set argsUnnamed = WScript.Arguments.Unnamed

WScript.Echo "There are " & argsNamed.Count & " named arguments."
WScript.Echo "There are " & argsUnnamed.Count & " unnamed arguments."

</script>
</job>
</package>
```

Properties

Item Property | Length Property

Methods

Count Method | Exists Method

See Also

WshArguments Object | WshUnnamed Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshNamed Object Properties and Methods

Properties

Item Property

Length Property

Methods

Count Method

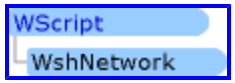
Exists Method

© 2001 Microsoft Corporation. All rights reserved.

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshNetwork Object



Provides access to the shared resources on the network to which your computer is connected.

Remarks

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.

Example

The following example demonstrates displaying the domain name, computer name, and user name for the current computer system using the WshNetwork object.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         Set WshNetwork = WScript.CreateObject("WScript.Network")
         WScript.Echo "Domain = " & WshNetwork.UserDomain
         WScript.Echo "Computer Name = " & WshNetwork.ComputerName
         WScript.Echo "User Name = " & WshNetwork.UserName
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshNetwork = WScript.CreateObject("WScript.Network");
         WScript.Echo("Domain = " + WshNetwork.UserDomain);
         WScript.Echo("Computer Name = " + WshNetwork.ComputerName);
         WScript.Echo("User Name = " + WshNetwork.UserName);
      </script>
   </job>
</package>
```

Properties

ComputerName Property | UserDomain Property | UserName Property

Methods

AddWindowsPrinterConnection Method | AddPrinterConnection Method | EnumNetworkDrives Method | EnumPrinterConnection Method | MapNetworkDrive Method | RemoveNetworkDrive Method | RemovePrinterConnection Method | SetDefaultPrinter Method

See Also

Running Your Scripts | WScript Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshNetwork Object Properties and Methods

Properties

ComputerName Property

UserDomain Property

UserName Property

Methods

AddPrinterConnection Method

EnumNetworkDrives Method

EnumPrinterConnection Method

MapNetworkDrive Method

RemoveNetworkDrive Method

RemovePrinterConnection Method

SetDefaultPrinter Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshRemote Object

Provides access to the remote script process.

Remarks

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.

Example

The following example demonstrates how the **WshRemote** object is used to start a remote script.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}
```

Properties

Status Property | Error Property

Methods

Execute Method | Terminate Method

Events

Start Event | End Event | Error Event

See Also

WshController Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshRemote Object Properties, Methods, and Events

Properties

Status Property

Error Property

Methods

Execute Method

Terminate Method

Events

Start Event

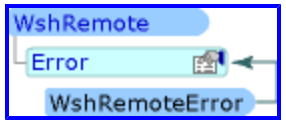
End Event

Error Event

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshRemoteError Object



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

Remarks

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

Example

The following example demonstrates how the **WshRemoteError** object is used to show where the error occurred in the script along with a description of the error.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
WScript.ConnectObject RemoteScript, "remote_"
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop

WScript.DisconnectObject RemoteScript
```

```
Sub remote_Error
    Dim theError
    Set theError = RemoteScript.Error
    WScript.Echo "Error " & theError.Number & " - Line: " & theError.Line & ", Char: " & theError.Character & vbCr
    WScript.Quit -1
End Sub
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
WScript.ConnectObject(RemoteScript, "remote_");
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}

WScript.DisconnectObject(RemoteScript);

function remote_Error()
{
    var theError = RemoteScript.Error;
    WScript.Echo("Error " + theError.Number + " - Line: " + theError.Line + ", Char: " + theError.Character + "\nD
    WScript.Quit(-1);
}
```

Properties

Description Property | Line Property | Character Property | SourceText Property | Source Property | Number Property

See Also

WshRemote Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshRemoteError Object Properties

Properties

Description Property

Line Property

Character Property

SourceText Property

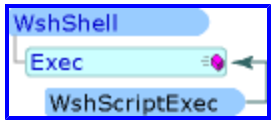
Source Property

Number Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshScriptExec Object



Provides status information about a script run with **Exec** along with access to the **StdIn**, **StdOut**, and **StdErr** streams.

Remarks

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.

Example

The following code runs calc.exe and echoes the final status to the screen.

[VBScript]

```
Dim WshShell, oExec
Set WshShell = CreateObject("WScript.Shell")

Set oExec = WshShell.Exec("calc")

Do While oExec.Status = 0
     WScript.Sleep 100
Loop

WScript.Echo oExec.Status
```

[JScript]

```
var WshShell = new ActiveXObject("WScript.Shell");
var oExec = WshShell.Exec("calc");

while (oExec.Status == 0)
{
     WScript.Sleep(100);
}

WScript.Echo(oExec.Status);
```

Properties

Status Property | StdOut Property | StdIn Property | StdErr Property

Methods

Terminate Method

See Also

WScript Object | Exec Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshScriptExec Object Properties and Methods

Properties

ExitCode Property

ProcessID Property

Status Property

StdOut Property

StdIn Property

StdErr Property

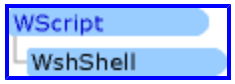
Methods

Terminate Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshShell Object



Provides access to the native Windows shell.

Remarks

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).

Example

The following example demonstrates the creation of a shortcut to the script being run and a URL shortcut to www.microsoft.com:

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         strDesktop = WshShell.SpecialFolders("Desktop")
         set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
         oShellLink.TargetPath = WScript.ScriptFullName
         oShellLink.WindowStyle = 1
         oShellLink.Hotkey = "CTRL+SHIFT+F"
```

```
         oShellLink.IconLocation = "notepad.exe, 0"
         oShellLink.Description = "Shortcut Script"
         oShellLink.WorkingDirectory = strDesktop
         oShellLink.Save
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         strDesktop = WshShell.SpecialFolders("Desktop");
         var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
         oShellLink.TargetPath = WScript.ScriptFullName;
         oShellLink.WindowStyle = 1;
         oShellLink.Hotkey = "CTRL+SHIFT+F";
         oShellLink.IconLocation = "notepad.exe, 0";
         oShellLink.Description = "Shortcut Script";
         oShellLink.WorkingDirectory = strDesktop;
         oShellLink.Save();
      </script>
   </job>
</package>
```

Properties

CurrentDirectory Property | Environment Property | SpecialFolders Property

Methods

AppActivate Method | CreateShortcut Method | ExpandEnvironmentStrings Method | LogEvent Method | Popup Method | RegDelete Method | RegRead Method | RegWrite Method | Run Method | SendKeys Method | Exec Method

See Also

Running Your Scripts

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshShellObject Properties and Methods

Properties

CurrentDirectory Property

Environment Property

SpecialFolders Property

Methods

AppActivate Method

CreateShortcut Method

ExpandEnvironmentStrings Method

LogEvent Method

Popup Method

RegDelete Method

RegRead Method

RegWrite Method

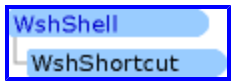
Run Method

SendKeys Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshShortcut Object



Allows you to create a shortcut programmatically.

Example

The following example demonstrates the creation of a shortcut to the script being run:

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         strDesktop = WshShell.SpecialFolders("Desktop")
         set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
         oShellLink.TargetPath = WScript.ScriptFullName
         oShellLink.WindowStyle = 1
         oShellLink.Hotkey = "CTRL+SHIFT+F"
         oShellLink.IconLocation = "notepad.exe, 0"
         oShellLink.Description = "Shortcut Script"
         oShellLink.WorkingDirectory = strDesktop
         oShellLink.Save
      </script>
   </job>

   <job id="js">
      <script language="JScript">
```

```
        var WshShell = WScript.CreateObject("WScript.Shell");
        strDesktop = WshShell.SpecialFolders("Desktop");
        var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
        oShellLink.TargetPath = WScript.ScriptFullName;
        oShellLink.WindowStyle = 1;
        oShellLink.Hotkey = "CTRL+SHIFT+F";
        oShellLink.IconLocation = "notepad.exe, 0";
        oShellLink.Description = "Shortcut Script";
        oShellLink.WorkingDirectory = strDesktop;
        oShellLink.Save();
      </script>
   </job>
</package>
```

Properties

Arguments Property | Description Property | FullName Property (WshShortcut Object) | Hotkey Property | IconLocation Property | TargetPath Property | WindowStyle Property | WorkingDirectory Property

Methods

Save Method

See Also

Running Your Scripts | CreateShortcut Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshShortcut Object Properties and Methods

Properties

Arguments Property

Description Property

FullName Property

Hotkey Property

IconLocation Property

TargetPath Property

WindowStyle Property

WorkingDirectory Property

Methods

Save Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshSpecialFolders Object

Provides access to the collection of Windows special folders.

Remarks

The **WshShell** object's **SpecialFolders** property returns the **WshSpecialFolders** object, which is a collection of special folders. This collection contains references to Windows special folders (for example, the **Desktop** folder, **Start Menu** folder, and **Personal Documents** folder). This collection retrieves paths to special folders using the special folder name as the index. A special folder's path depends on the user environment. The information stored in a special folder is unique to the user logged onto the computer system. If several different users have accounts on the same computer system, several different sets of special folders are stored on the hard disk.

The following special folders are available:

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

Example

The following script demonstrates the use of the **WshSpecialFolders** Object:

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         strDesktop = WshShell.SpecialFolders("Desktop")
         set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
         oShellLink.TargetPath = WScript.ScriptFullName
         oShellLink.WindowStyle = 1
         oShellLink.Hotkey = "CTRL+SHIFT+F"
         oShellLink.IconLocation = "notepad.exe, 0"
         oShellLink.Description = "Shortcut Script"
         oShellLink.WorkingDirectory = strDesktop
         oShellLink.Save
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         strDesktop = WshShell.SpecialFolders("Desktop");
         var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
         oShellLink.TargetPath = WScript.ScriptFullName;
         oShellLink.WindowStyle = 1;
         oShellLink.Hotkey = "CTRL+SHIFT+F";
         oShellLink.IconLocation = "notepad.exe, 0";
         oShellLink.Description = "Shortcut Script";
         oShellLink.WorkingDirectory = strDesktop;
         oShellLink.Save();
      </script>
   </job>
</package>
```

Properties

Item Property

Methods

Count Method

See Also

Running Your Scripts | SpecialFolders Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshSpecialFolders Object Properties and Methods

Properties

Item Property

Length Property

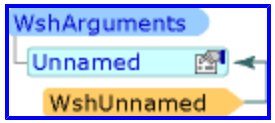
Methods

Count Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshUnnamed Object



Provides access to the unnamed arguments from the command line.

Remarks

The **WshUnnamed** object is a read-only collection that is returned by the **Unnamed** property of the **WshArguments** object. Individual argument values are retrieved from this collection using zero-based indexes. Two of the **WshArguments** object's properties are filtered arguments collections — one contains the named arguments (the **WshNamed** object), the other contains the unnamed arguments (the **WshUnnamed** object). In total, this gives you three ways to access sets of command-line arguments.

- You can access the entire set of arguments (those with and without names) with the **WshArguments** object.
- You can access the arguments that have names with the **WshNamed** object.
- You can access the arguments that have no names with the **WshUnnamed** object.

Example

The following code displays the number of named and unnamed command-line arguments.

```
<package>
<job id="JS">
<script language="JScript">

var argsNamed = WScript.Arguments.Named;
var argsUnnamed = WScript.Arguments.Unnamed;

WScript.Echo("There are " + argsNamed.length + " named arguments.");
WScript.Echo("There are " + argsUnnamed.length + " unnamed arguments.");

</script>
</job>

<job id="VBS">
<script language="VBScript">

Dim argsNamed, argsUnnamed
Set argsNamed = WScript.Arguments.Named
```

```
Set argsUnnamed = WScript.Arguments.Unnamed

WScript.Echo "There are " & argsNamed.Count & " named arguments."
WScript.Echo "There are " & argsUnnamed.Count & " unnamed arguments."

</script>
</job>
</package>
```

Properties

Item Property | Length Property (WshArguments object)

Methods

Count Method

See Also

WshArguments Object | WshNamed Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshUnnamed Object Properties and Methods

Properties

Item Property

Length Property

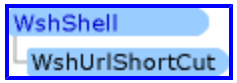
Methods

Count Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshUrlShortcut Object



Allows you to create a shortcut to an Internet resource programmatically.

Remarks

The **WshUrlShortcut** object is a child object of the **WshShell** object — you must use the **WshShell** method **CreateShortcut** to create a **WshUrlShortcut** object (e.g., `WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")`).

Example

The following example demonstrates the creation of a URL shortcut to www.microsoft.com.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         set oUrlLink = WshShell.CreateShortcut(strDesktop & "\Microsoft Web Site.url")
         oUrlLink.TargetPath = "http://www.microsoft.com"
         oUrlLink.Save
```

```
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         var oUrlLink = WshShell.CreateShortcut(strDesktop + "\\Microsoft Web Site.url");
         oUrlLink.TargetPath = "http://www.microsoft.com";
         oUrlLink.Save();
      </script>
   </job>
</package>
```

Properties

FullName Property (WshUrlShortcut Object) | TargetPath Property

Methods

Save Method

See Also

Running Your Scripts | CreateShortcut Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WshUrlShortcut Object Properties and Methods

Properties

FullName Property

TargetPath Property

Methods

Save Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Properties

In this Section

Arguments Property
: Returns the **WshArguments** object.

AtEndOfLine Property
: Returns a Boolean value indicating whether the end of an input line has been reached.

AtEndOfStream Property
: Returns a Boolean value indicating whether the end of an input stream has been reached.

Character Property
: Reports the specific character in the line of code that contained the error.

Column Property
: Returns the column number of the current character position in an input stream.

ComputerName Property
: Returns the name of the computer.

CurrentDirectory Property
: Allows you to set or retrieve the active script's current working folder.

Description Property

Returns the description of a shortcut.

Environment Property

Returns the **WshEnvironment** object.

Error Property (WshRemote)

Exposes a **WshRemoteError** object.

ExitCode Property

Returns the exit code set by a script/program run using the **Exec()** method.

FullName Property

Returns a fully qualified path name.

Hotkey Property

Allows you to assign a key combination to a shortcut and determine the key combination to a shortcut.

IconLocation Property

Allows you to assign an icon to a shortcut and determine which icon has been assigned to a shortcut.

Interactive Property

Allows you to set the script mode programmatically, as well as determine the script mode programmatically.

Item Property

Exposes a specified item from a collection.

Item Property (WshNamed)

Provides access to the items in the **WshNamed** object.

Item Property (WshUnnamed)

Returns an item using a zero-based index.

Length Property

Returns the number of items in a collection.

Line Property (WScript)

Returns the current line number in an input stream.

Line Property (WshRemote)

Identifies the line in a script that contains an error-causing statement.

Name Property

Returns the friendly name of the **WScript** object (the host executable file).

Number Property

Reports the error number representing a script error.

Path PropertyPath

Returns the name of the directory containing the **WScript** object (the host executable file).

ProcessID Property

Reports the process ID (PID) of a process started with the **WshScriptExec** object.

ScriptFullName Property

Returns the full path-name of the currently running script.

ScriptName Property
: Returns the file name of the currently running script.

Source Property
: Identifies the COM object responsible for causing the script error.

SourceText Property
: Contains the line of source code that caused an error.

SpecialFolders Property
: Returns the **WshSpecialFolders** object.

Status Property (WshRemote)
: Reports the current running-condition of the remote script.

Status Property (WshScriptExec)
: Provides status information about a script run with the **Exec()** method.

StdErr Property (WScript)
: Exposes the write-only error output stream for the current script.

StdErr Property (WshScriptExec)
: Exposes the read-only **stderr** output stream of the **Exec** object.

StdIn Property (WScript)
: Exposes the read-only input stream for the current script.

StdIn Property (WshScriptExec)
: Exposes the write-only **stdin** input stream of the **Exec** object.

StdOut Property (WScript)
: Exposes the write-only output stream for the current script.

StdOut Property (WshScriptExec)
: Exposes the write-only **stdout** output stream of the **Exec** object.

TargetPath Property
: Allows you to assign a path to the executable file to which a shortcut points, as well as a determine the path to the executable file pointed to by a shortcut.

UserDomain Property
: Returns the user's domain name.

UserName Property
: Returns the name of the user.

Version Property
: Returns the version of WSH.

WindowStyle Property
: Allows you to assign a window style to a shortcut, as well as determine the type of window style used by a shortcut.

WorkingDirectory Property
: Allows you to assign a working directory to a shortcut, as well as determine the working directory used by a shortcut.

Related Sections

WSH Language
List of elements that make up WSH Reference.
WSH Basics
Learn the basics of WSH.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Arguments Property (Shortcut Object)

Sets the arguments for a shortcut, or identifies a shortcut's arguments.

object.**Arguments**

Arguments

object
WshShortcut object.

Remarks

The **Arguments** property returns a string.

Example

The following code sets the **Arguments** property.

[VBScript]

```
set WshShell = WScript.CreateObject("WScript.Shell")
strDesktop = WshShell.SpecialFolders("Desktop")
set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
oShellLink.TargetPath = WScript.ScriptFullName
oShellLink.WindowStyle = 1
oShellLink.Hotkey = "Ctrl+Alt+f"
oShellLink.IconLocation = "notepad.exe, 0"
oShellLink.Description = "Shortcut Script"
oShellLink.WorkingDirectory = strDesktop
oShellLink.Arguments = "C:\myFile.txt"
oShellLink.Save
```

[JScript]

```
var WshShell = WScript.CreateObject("WScript.Shell");
var strDesktop = WshShell.SpecialFolders("Desktop");
var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
oShellLink.TargetPath = WScript.ScriptFullName;
oShellLink.WindowStyle= 1;
oShellLink.Hotkey= "Ctrl+Alt+f";
oShellLink.IconLocation= "notepad.exe, 0";
oShellLink.Description= "Shortcut Script";
oShellLink.WorkingDirectory= strDesktop;
oShellLink.Arguments = "C:\\myFile.txt"; oShellLink.Save();
```

See Also

WshShortcut Object | WshShell Object | CreateShortcut Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Arguments Property (WScript Object)

Returns the **WshArguments** object (a collection of arguments).

object.**Arguments**

Arguments

object
WScript object.

Remarks

The **Arguments** property contains the **WshArguments** object (a collection of arguments). Use a zero-based index to retrieve individual arguments from this collection.

Example

The following code samples display the entire set of command-line parameters associated with a script.

[VBScript]

```
Set objArgs = WScript.Arguments
For I = 0 to objArgs.Count - 1
   WScript.Echo objArgs(I)
Next
```

[JScript]

```
objArgs = WScript.Arguments;
for (i = 0; i < objArgs.length; i++)
{
   WScript.Echo(objArgs(i));
}
```

See Also

WScript Object | WshArguments Object | WshNamed Object | WshUnnamed Object

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

AtEndOfLine Property

Returns a Boolean value indicating whether the end of a line in an input stream has been reached.

object.**AtEndOfLine**

Arguments

object
StdIn text stream object.

Remarks

The **AtEndOfLine** property contains a Boolean value indicating whether the end of a line in an input stream has been reached. The **AtEndOfLine** property returns **True** if the stream pointer immediately precedes the end-of-line marker in an input stream, **False** if it does not. The **StdIn**, **StdOut**, and **StdErr** properties and methods work only when the script is run with CScript.exe. If the script is run with WScript.exe, an error occurs.

Example

The following samples will read a line of text from the keyboard and display whatever was typed when the end of the line is seen.

[VBScript]

```
Dim Input
```

```
Input = ""

Do While Not WScript.StdIn.AtEndOfLine
   Input = Input & WScript.StdIn.Read(1)
Loop
WScript.Echo Input
```

[JScript]

```
var input = "";
while (!WScript.StdIn.AtEndOfLine)
{
   input += WScript.StdIn.Read(1);
}
WScript.Echo(input);
```

See Also

StdIn Property | Error Messages

Build: Topic Version 5.6.9309.1546

Windows Script Host

AtEndOfStream Property

Returns a Boolean value indicating whether the end of an input stream has been reached.

object.**AtEndOfStream**

Arguments

object
StdIn text stream object.

Remarks

The **AtEndOfStream** property contains a Boolean value indicating whether the end of an input stream has been reached. The **AtEndOfStream** property returns **True** if the stream pointer is at the end of an input stream, **False** if not. The **StdIn**, **StdOut**, and **StdErr** properties and methods work only when the script is run with CScript.exe. If the script is run with WScript.exe, an error occurs.

Example

The following code samples demonstrate the **AtEndOfStream** property by reading a standard directory listing from "dir", stripping the top and bottom lines that aren't actual entries, and double spacing the directory entries.

[VBScript]

```
Dim StdIn, StdOut, Str1, Str2

Set StdIn = WScript.StdIn
Set StdOut = WScript.StdOut

Str1 = ""
Str2 = ""For i = 0 to 4
   StdIn.SkipLine
Next

i = 0
Do While Not StdIn.AtEndOfStream
     If i >= 2 Then
          StdOut.WriteLine Str1
     End If
     i = i + 1
     Str1 = Str2
     Str2 = StdIn.ReadLine
Loop
```

[JScript]

```
var stdin = WScript.StdIn;
var stdout = WScript.StdOut;
```

```
var str1, str2 = "";
var i;
for (i = 0; i < 5; i++)
     stdin.SkipLine();
i = 0;
while (!stdin.AtEndOfStream)
{
     if (i++ >= 2)
     {
          stdout.WriteLine(str1);
     }
     str1 = str2;
     str2 = stdin.ReadLine();
}
```

See Also

StdIn Property | Error Messages

Build: Topic Version 5.6.9309.1546

Windows Script Host

BuildVersion Property

Returns the Windows Script Host build version number.

Object.**BuildVersion**

Parameters

Object
WScript object.

Remarks

The **BuildVersion** property returns an integer value.

The full version number of Windows Script Host consists of the product release version number followed by the build version number. For example, if the Windows Script Host product release version number is 5.6, and the build version number is 6626, the full version number is 5.6.6626.

Example

The following code returns the full version number of the Windows Script Host. This consists of the product release version number plus the build version number.

```
WScript.Echo WScript.Version & "." & WScript.BuildVersion;
```

See Also

Version Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

Character Property

Reports the specific character in a line of code that contains an error.

```
Object.Character
```

Arguments

Object
WshRemoteError object.

Remarks

The **Character** property returns a signed long integer.

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).

Example

The following JScript code demonstrates how the **WshRemoteError** object exposes the character in the line of code that contained the error.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
WScript.ConnectObject RemoteScript, "remote_"
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop

Sub remote_Error
    Dim theError
    Set theError = RemoteScript.Error
    WScript.Echo "Error - Line: " & theError.Line & ", Char: " & theError.Character & vbCrLf & "Description: " & t
    WScript.Quit -1
End Sub
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
WScript.ConnectObject(RemoteScript, "remote_");
```

```
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}

function remote_Error()
{
    var theError = RemoteScript.Error;
    WScript.Echo("Error - Line: " + theError.Line + ", Char: " + theError.Character + "\nDescription: " + theError
    WScript.Quit(-1);
}
```

See Also

WshRemote Object | WshRemoteError Object | Line Property| Description Property | SourceText Property | Number Property | Source Property

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Column Property

Returns the column number of the current character position in an input stream.

object.**Column**

Arguments

object
StdIn text stream object.

Remarks

The **Column** property contains a read-only integer value indicating the column number of the current character position in an input stream. The **Column** property is equal to 1 after a newline character is written (even before any other character is written). The **StdIn**, **StdOut**, and **StdErr** properties and methods work only when the script is run with CScript.exe. If the script is run with WScript.exe, an error occurs.

Example

The following code demonstrates the use of the **Column** property by reading input from the keyboard and breaking it into lines of 20 characters.

[VBScript]

```
Dim Input
Input = ""

Do While Not WScript.StdIn.AtEndOfLine
   Input = Input & WScript.StdIn.Read(1)
   If (WScript.StdIn.Column - 1) Mod 20 = 0 Then
      Input = Input & vbCrLf
   End If
Loop
WScript.Echo Input
```

[JScript]

```
var input = "";
while (!WScript.StdIn.AtEndOfLine)
{
   input += WScript.StdIn.Read(1);
   if ((WScript.StdIn.Column - 1) % 20 == 0)
   input += "\n";
}
WScript.Echo(input);
```

See Also

StdIn Property | Error Messages

Build: Topic Version 5.6.9309.1546

Windows Script Host

ComputerName Property

Returns the name of the computer system.

object.**ComputerName**

Arguments

object
WshNetwork object.

Remarks

The **ComputerName** property contains a string value indicating the name of the computer system.

Example

The following example demonstrates the use of the ComputerName property.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         Set WshNetwork = WScript.CreateObject("WScript.Network")
         WScript.Echo "Domain = " & WshNetwork.UserDomain
         WScript.Echo "Computer Name = " & WshNetwork.ComputerName
         WScript.Echo "User Name = " & WshNetwork.UserName
      </script>
   </job>
```

```
   <job id="js">
      <script language="JScript">
         var WshNetwork = WScript.CreateObject("WScript.Network");
         WScript.Echo("Domain = " + WshNetwork.UserDomain);
         WScript.Echo("Computer Name = " + WshNetwork.ComputerName);
         WScript.Echo("User Name = " + WshNetwork.UserName);
         }
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | WshNetwork Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

CurrentDirectory Property

Retrieves or changes the current active directory.

object.**CurrentDirectory**

Arguments

object
WshShell object.

Remarks

The **CurrentDirectory** returns a string that contains the fully qualified path of the current working directory of the active process.

Example

The following code displays the current active directory.

[VBScript]

```
Dim WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
WScript.Echo WshShell.CurrentDirectory
```

[JScript]

```
var WshShell = WScript.CreateObject ("WScript.Shell");
WScript.Echo (WshShell.CurrentDirectory);
```

See Also

WshShell Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Description Property

Returns a shortcut's description.

object.**Description**

Arguments

object
WshShortcut object.

Remarks

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

Example

The following example demonstrates the use of the Description Property.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         strDesktop = WshShell.SpecialFolders("Desktop")
         set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
         oShellLink.TargetPath = WScript.ScriptFullName
         oShellLink.WindowStyle = 1
         oShellLink.Hotkey = "Ctrl+Alt+e"
         oShellLink.IconLocation = "notepad.exe, 0"
         oShellLink.Description = "Shortcut Script"
         oShellLink.WorkingDirectory = strDesktop
         oShellLink.Save
         set oUrlLink = WshShell.CreateShortcut(strDesktop & "\Microsoft Web Site.url")
         oUrlLink.TargetPath = "http://www.microsoft.com"
         oUrlLink.Save
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         strDesktop = WshShell.SpecialFolders("Desktop");
         var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
         oShellLink.TargetPath = WScript.ScriptFullName;
         oShellLink.WindowStyle = 1;
         oShellLink.Hotkey = "Ctrl+Alt+e";
         oShellLink.IconLocation = "notepad.exe, 0";
```

```
        oShellLink.Description = "Shortcut Script";
        oShellLink.WorkingDirectory = strDesktop;
        oShellLink.Save();
        var oUrlLink = WshShell.CreateShortcut(strDesktop + "\\Microsoft Web Site.url");
        oUrlLink.TargetPath = "http://www.microsoft.com";
        oUrlLink.Save();
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | WshShortcut Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Description Property (WshRemoteError)

Contains a brief description of the error that caused the remote script to terminate.

Object.**Description**

Arguments

Object
WshRemoteError object.

Remarks

The **Description** property returns a string.

If an error description is not available, the **Description** property returns an empty string.

Example

The following JScript code demonstrates how the **WshRemoteError** object exposes the description of the error.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
WScript.ConnectObject RemoteScript, "remote_"
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop

Sub remote_Error
    Dim theError
    Set theError = RemoteScript.Error
    WScript.Echo "Error - Line: " & theError.Line & ", Char: " & theError.Character & vbCrLf & "Description: " & t
    WScript.Quit -1
End Sub
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
WScript.ConnectObject(RemoteScript, "remote_");
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}

function remote_Error()
{
    var theError = RemoteScript.Error;
    WScript.Echo("Error - Line: " + theError.Line + ", Char: " + theError.Character + "\nDescription: " + theError
    WScript.Quit(-1);
}
```

See Also

WshRemote Object | WshRemoteError Object | Line Property | Character Property | SourceText Property | Number Property | Source Property

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Environment Property

Returns the **WshEnvironment** object (a collection of environment variables).

object.**Environment** ([*strType*])

Arguments

object
WshShell object.
strType
Optional. Specifies the location of the environment variable.

Remarks

The **Environment** property contains the **WshEnvironment** object (a collection of environment variables). If *strType* is supplied, it specifies where the environment variable resides with possible values of *System*, *User*, *Volatile*, or *Process*. If *strType* is not supplied, the **Environment** property returns different environment variable types depending on the operating system.

Type of Environment Variable	Operating System
System	Microsoft Windows NT/2000
Process	Windows 95/98/Me

Note For Windows 95/98/Me, only one *strType* is permitted — *Process*.

The following table lists some of the variables that are provided with the Windows operating system. Scripts can access environment variables that have been set by other applications.

Note None of the following variables are available from the *Volatile* type.

Name	Description	System	User	Process (NT/2000)	Process (98/ME)
NUMBER_OF_PROCESSORS	Number of processors running on the machine.	X	-	X	-
PROCESSOR_ARCHITECTURE	Processor type of the user's workstation.	X	-	X	-
PROCESSOR_IDENTIFIER	Processor ID of the user's workstation.	X	-	X	-
PROCESSOR_LEVEL	Processor level of the user's workstation.	X	-	X	-
PROCESSOR_REVISION	Processor version of the user's workstation.	X	-	X	-
OS	Operating system on the user's workstation.	X	-	X	-
COMSPEC	Executable file for the command prompt (typically cmd.exe).	X	-	X	X
HOMEDRIVE	Primary local drive (typically the C drive).	-	-	X	-
HOMEPATH	Default directory for users (typically \users\default in Windows 2000).	-	-	X	-
PATH	PATH environment variable.	X	X	X	X
PATHEXT	Extensions for executable files (typically .com, .exe, .bat, or .cmd).	X	-	X	-

PROMPT	Command prompt (typically PG).	-	-	X	X
SYSTEMDRIVE	Local drive on which the system directory resides (typically c:\).	-	-	X	-
SYSTEMROOT	System directory (for example, c:\winnt). This is the same as WINDIR.	-	-	X	-
WINDIR	System directory (for example, c:\winnt). This is the same as SYSTEMROOT.	X	-	X	X
TEMP	Directory for storing temporary files (for example, c:\temp).	-	X	X	X
TMP	Directory for storing temporary files (for example, c:\temp).	-	X	X	X

Example

The following code retrieves the system environment variable NUMBER_OF_PROCESSORS.

[VBScript]

```
Set WshShell = WScript.CreateObject("WScript.Shell")
Set WshSysEnv = WshShell.Environment("SYSTEM")
WScript.Echo WshSysEnv("NUMBER_OF_PROCESSORS")
```

[JScript]

```
var WshShell = WScript.CreateObject("WScript.Shell");
var WshSysEnv = WshShell.Environment("SYSTEM");
WScript.Echo(WshSysEnv("NUMBER_OF_PROCESSORS"));
```

See Also

WshEnvironment Object | WshShell Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Error Property (WshRemote)

Exposes the **WshRemoteError** object, which holds information about the error that caused the remote script to terminate prematurely.

Object.**Error**

Arguments

Object
WshRemote object.

Remarks

The **Error** property returns a **WshRemoteError** object.

Example

The following code demonstrates how the **Error** property of the **WshRemote** object exposes a **WshRemoteError** object, which exposes the line, character, and description of the error.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
```

```
WScript.ConnectObject RemoteScript, "remote_"
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop

Sub remote_Error
    Dim theError
    Set theError = RemoteScript.Error
    WScript.Echo "Error - Line: " & theError.Line & ", Char: " & theError.Character & vbCrLf & "Description: " & t
    WScript.Quit -1
End Sub
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
WScript.ConnectObject(RemoteScript, "remote_");
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}

function remote_Error()
{
    var theError = RemoteScript.Error;
    WScript.Echo("Error - Line: " + theError.Line + ", Char: " + theError.Character + "\nDescription: " + theError
}
```

See Also

WshController Object | WshRemote Object | WshRemoteError Object | Status Property | Execute Method | Terminate Method | Start Event | End Event | Error Event

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

ExitCode Property

Returns the exit code set by a script or program run using the **Exec()** method.

Object.**ExitCode**

Parameters

Object
WshScriptExec Object

Remarks

Executables set an exit code when they finish running. This conveys the status information when a process ends. Often, it is used to send an error code (or some other piece of information) back to the caller. If the process has not finished, the **ExitCode** property returns 0. The values returned from **ExitCode** depend on the application that was called.

Example

The following code shows an example of the **ExitCode** property.

[VBScript]

```
Dim WshShell, oExec
Set WshShell = CreateObject("WScript.Shell")
Set oExec    = WshShell.Exec("%comspec% /c dire")

Function ReadAllFromAny(oExec)

     If Not oExec.StdOut.AtEndOfStream Then
          ReadAllFromAny = oExec.StdOut.ReadAll
          Exit Function
     End If
```

```
     If Not oExec.StdErr.AtEndOfStream Then
          ReadAllFromAny = oExec.StdErr.ReadAll
          Exit Function
     End If

     ReadAllFromAny = -1
End Function

Dim allInput, tryCount

allInput = ""
tryCount = 0

Do While True

     Dim input
     input = ReadAllFromAny(oExec)

     If -1 = input Then
          If tryCount > 10 And oExec.Status = 1 Then
               Exit Do
          End If
          tryCount = tryCount + 1
          WScript.Sleep 100
     Else
          allInput = allInput & input
          tryCount = 0
     End If
Loop

If oExec.ExitCode <> 0 Then
     WScript.Echo "Warning: Non-zero exit code"
End If

WScript.Echo allInput
```

[JScript]

```
var WshShell = new ActiveXObject("WScript.Shell");
var oExec    = WshShell.Exec("%comspec% /c dire");

function ReadAllFromAny(oExec)
```

```
{
     if (!oExec.StdOut.AtEndOfStream)
          return oExec.StdOut.ReadAll();

     if (!oExec.StdErr.AtEndOfStream)
          return oExec.StdErr.ReadAll();

     return -1;
}

var allInput = "";
var tryCount = 0;

while (true)
{
     var input = ReadAllFromAny(oExec);
     if (-1 == input)
     {
          if (tryCount++ > 10 && oExec.Status == 1)
               break;
          WScript.Sleep(100);
     }
     else
     {
          allInput += input;
          tryCount = 0;
     }
}

if (oExec.ExitCode != 0)
{
     WScript.Echo("Warning: Non-zero exit code");
}

WScript.Echo(allInput);
```

See Also

Exec Method | WshScriptExec Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

FullName Property (WScript Object)

Returns the fully qualified path of the host executable (CScript.exe or WScript.exe).

object.**FullName**

Arguments

object
WScript object.

Remarks

The **FullName** property is a read-only string representing the fully qualified path of the host executable.

Example

The following JScript code uses the FullName property:

```
WScript.Echo (WScript.FullName);
```

produces the following output.

```
C:\WINNT\System32\cscript.exe
```

See Also

Path Property | WScript Object | WshShortcut Object | WshUrlShortcut Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

FullName Property (WshShortcut Object)

Returns the fully qualified path of the shortcut object's target.

object.**FullName**

Arguments

object
WshShortcut object.

Remarks

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

Example

The following code retrieves the fully qualified path of a shortcut.

[VBScript]

```
set WshShell = WScript.CreateObject("WScript.Shell")
strDesktop = WshShell.SpecialFolders("Desktop")
set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
oShellLink.TargetPath = WScript.ScriptFullName
oShellLink.WindowStyle = 1
oShellLink.Hotkey = "CTRL+SHIFT+F"
oShellLink.IconLocation = "notepad.exe, 0"
oShellLink.Description = "Shortcut Script"
```

```
oShellLink.WorkingDirectory = strDesktop
oShellLink.Save
WScript.Echo oShellLink.FullName
```

[JScript]

```
var WshShell = new ActiveXObject("WScript.Shell");
strDesktop = WshShell.SpecialFolders("Desktop");
var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
oShellLink.TargetPath = WScript.ScriptFullName;
oShellLink.WindowStyle = 1;
oShellLink.Hotkey = "CTRL+SHIFT+F";
oShellLink.IconLocation = "notepad.exe, 0";
oShellLink.Description = "Shortcut Script";
oShellLink.WorkingDirectory = strDesktop;
oShellLink.Save();
WScript.Echo(oShellLink.FullName);
```

See Also

Path Property | WScript Object | WshShortcut Object | WshUrlShortcut Object

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

FullName Property (WshUrlShortcut Object)

Returns the fully qualified path of the shortcut object's target.

object.**FullName**

Arguments

object
WshUrlShortcut object.

Remarks

String. Read-only.

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

Example

The following code retrieves the fully qualified path of a URL shortcut.

[VBScript]

```
set oUrlLink = WshShell.CreateShortcut(strDesktop & "\Microsoft Web Site.url")
oUrlLink.TargetPath = "http://www.microsoft.com"
oUrlLink.Save
WScript.Echo oUrlLink.FullName
```

[JScript]

```
var oUrlLink = WshShell.CreateShortcut(strDesktop + "\\Microsoft Web Site.url");
oUrlLink.TargetPath = "http://www.microsoft.com";
oUrlLink.Save();
WScript.Echo (oUrlLink.FullName);
```

See Also

Path Property | WScript Object | WshShortcut Object | WshUrlShortcut Object

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Hotkey Property

Assigns a key-combination to a shortcut, or identifies the key-combination assigned to a shortcut.

```
object.Hotkey = strHotkey
```

Arguments

object
WshShortcut object.
strHotkey
A string representing the key-combination to assign to the shortcut.

Syntax

The syntax of *strHotkey* is:

```
[KeyModifier]KeyName
```

KeyModifier
Can be any one of the following: ***ALT+***, ***CTRL+***, ***SHIFT+***, ***EXT+***.

Note ***EXT+*** means "Extended key." — it appears here in case a new type of SHIFT-key is added to the character set in the future.

KeyName
a ... z, 0 ... 9, F1 … F12, ...

The KeyName is not case-sensitive.

Remarks

A hotkey is a combination of keys that starts a shortcut when all associated keys are held down at the same time.

- Hotkeys can be used to start shortcuts located on your system's desktop and in the Windows Start menu.

Note Another name for a hotkey is a *Keyboard Shortcut*.

In Windows 2000, valid Hotkeys always begin with *CTRL + ALT*.

Example

The following example demonstrates the use of the HotKey property.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         strDesktop = WshShell.SpecialFolders("Desktop")
         set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
         oShellLink.TargetPath = WScript.ScriptFullName
         oShellLink.WindowStyle = 1
         oShellLink.Hotkey = "Ctrl+Alt+e"
         oShellLink.IconLocation = "notepad.exe, 0"
         oShellLink.Description = "Shortcut Script"
         oShellLink.WorkingDirectory = strDesktop
         oShellLink.Save
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         strDesktop = WshShell.SpecialFolders("Desktop");
         var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
         oShellLink.TargetPath = WScript.ScriptFullName;
         oShellLink.WindowStyle = 1;
         oShellLink.Hotkey = "Ctrl+Alt+e";
         oShellLink.IconLocation = "notepad.exe, 0";
         oShellLink.Description = "Shortcut Script";
         oShellLink.WorkingDirectory = strDesktop;
         oShellLink.Save();
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | Running Your Scripts | WshSpecialFolders Object | WshShortcut Object

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

IconLocation Property

Assigns an icon to a shortcut, or identifies the icon assigned to a shortcut.

object.**IconLocation** = *strIconLocation*

Arguments

object
WshShortcut object.
strIconLocation
A string that locates the icon. The string should contain a fully qualified path and an index associated with the icon. See example for more details.

Remarks

String.

Example

The following example demonstrates the use of the IconLocation property.

```
<package>
   <job id="vbs">
      <script language="VBScript">
```

```
        set WshShell = WScript.CreateObject("WScript.Shell")
        strDesktop = WshShell.SpecialFolders("Desktop")
        set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
        oShellLink.TargetPath = WScript.ScriptFullName
        oShellLink.WindowStyle = 1
        oShellLink.Hotkey = "Ctrl+Alt+e"
        oShellLink.IconLocation = "notepad.exe, 0"  'Zero is the index
        oShellLink.Description = "Shortcut Script"
        oShellLink.WorkingDirectory = strDesktop
        oShellLink.Save
     </script>
  </job>

  <job id="js">
     <script language="JScript">
        var WshShell = WScript.CreateObject("WScript.Shell");
        strDesktop = WshShell.SpecialFolders("Desktop");
        var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
        oShellLink.TargetPath = WScript.ScriptFullName;
        oShellLink.WindowStyle = 1;
        oShellLink.Hotkey = "Ctrl+Alt+e";
        oShellLink.IconLocation = "notepad.exe, 0";  //Zero is the index
        oShellLink.Description = "Shortcut Script";
        oShellLink.WorkingDirectory = strDesktop;
        oShellLink.Save();
     </script>
  </job>
</package>
```

See Also

Running Your Scripts | WshShortcut Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Interactive Property

Sets the script mode, or identifies the script mode.

object.**Interactive**

Arguments

object
WScript object.

Remarks

The Interactive Property returns a Boolean value.

There are two modes, *batch* and *interactive*. In interactive mode (the default), the script provides user interaction. Input to and output from the Windows Script Host is enabled. The script can echo information to dialog boxes and can wait for the user to provide feedback. In batch mode, this type of user interaction is not supported — all input and output to WSH is disabled. You can also set the script mode using the Windows Script Host command line switches //I (*for Interactive*), and //B (*for Batch*).

Example

The following JScript code displays the script mode.

```
WScript.Echo WScript.Interactive;
```

The following JScript code sets the script mode to batch.

```
WScript.Interactive = false;
```

See Also

WScript Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Item Property

Exposes a specified item from a collection.

Object.**Item**(*natIndex*)

Arguments

Object
The result of the **EnumNetworkDrive** or **EnumPrinterConnections** methods, or the object returned by the **Environment** or **SpecialFolders** properties.

natIndex
Item to retrieve.

Remarks

Item is the default property for each collection. For **EnumNetworkDrive** and **EnumPrinterConnections** collections, *index* is an integer; for the **Environment** and **SpecialFolders** collections, *index* is a string.

WshShell.SpecialFolders.Item (*strFolderName*) returns "Empty" in VBScript and "undefined" in JScript if the requested folder (*strFolderName*) is not available.

The following table lists special folders along with the version of Windows that supports them.

Folder	Windows version
AllUsersDesktop	Windows 2000
AllUsersStartMenu	Windows 2000
AllUsersPrograms	Windows 2000
AllUsersStartup	Windows 2000

Desktop	Windows 98/ME, Windows 2000
Favorites	Windows 98/ME, Windows 2000
Fonts	Windows 98/ME, Windows 2000
My Documents	Windows 98/ME, Windows 2000
NetHood	Windows 98/ME, Windows 2000
PrintHood	Windows 98/ME, Windows 2000
Programs	Windows 98/ME, Windows 2000
Recent	Windows 98/ME, Windows 2000
SendTo	Windows 98/ME, Windows 2000
Start Menu	Windows 98/ME, Windows 2000
StartupB	Windows 2000
Templates	Windows 2000

Example

The following code displays network mapping information for the drives and printers.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         Set WshNetwork = WScript.CreateObject("WScript.Network")
         Set oDrives = WshNetwork.EnumNetworkDrives
         Set oPrinters = WshNetwork.EnumPrinterConnections
         WScript.Echo "Network drive mappings:"
         For i = 0 to oDrives.Count - 1 Step 2
            WScript.Echo "Drive " & oDrives.Item(i) & " = " & oDrives.Item(i+1)
         Next
         WScript.Echo
         WScript.Echo "Network printer mappings:"
         For i = 0 to oPrinters.Count - 1 Step 2
            WScript.Echo "Port " & oPrinters.Item(i) & " = " & oPrinters.Item(i+1)
         Next
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshNetwork = WScript.CreateObject("WScript.Network");
         var oDrives = WshNetwork.EnumNetworkDrives();
```

```
         var oPrinters = WshNetwork.EnumPrinterConnections();
         WScript.Echo();
         WScript.Echo("Network drive mappings:");
         for(i = 0; i < oDrives.length; i += 2){
            WScript.Echo("Drive " + oDrives.Item(i) + " = " + oDrives.Item(i + 1));
         }
         WScript.Echo();
         WScript.Echo("Network printer mappings:");
         for(i = 0; i < oPrinters.length; i += 2){
            WScript.Echo("Port " + oPrinters.Item(i) + " = " + oPrinters.Item(i + 1));
         }
      </script>
   </job>
</package>
```

See Also

EnumNetworkDrive Method | EnumPrinterConnections Method | Environment Property | SpecialFolders Property

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Item Property (WshNamed)

Provides access to the items in the **WshNamed** object.

Object.**Item**(*key*)

Parameters

Object
WshNamed object

key
The name of the item you want to retrieve.

Remarks

The **Item** property returns a string. For collections, it returns an item based on the specified key. When entering the arguments at the command line, you can use blanks in string arguments if you enclose the string in quotes. Consider the following example:

```
myscript.vbs /stringarg:"This string has spaces"
```

The quotes will be removed in the **WshNamed** collection. For an argument to be in the **WshNamed** collection, it must have been used on the command line. If the argument has no value (such as a simple argument or an empty string), the **Item** property returns an empty string. Requesting a non-existent named argument from the **Item** property causes an error. To check if an argument exists, use the **Exists** method.

Example 1

In the following example, two named arguments are supplied to run a script. Inside the script, code causes the named arguments to be output. The **Item** property is used to index into the named arguments collection.

The following line is typed at the command prompt to run the script.

```
myScript.vbs /c:arg1 /d:arg2
```

If the following code is executed inside the script:

```
WScript.Echo WScript.Arguments.Named.Item("c")
WScript.Echo WScript.Arguments.Named.Item("d")
```

then the following output is produced:

```
arg1
arg2
```

See Also

WshNamed Object | WshUnnamed Object | Count Method | Item Property | Exists Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Item Property (WshUnnamed)

Returns an item using a zero-based index.

Object.**Item**(*key*)

Parameters

Object
WshUnnamed object.
key
Associated with either a collection or a **WshUnnamed** object.

Remarks

The **Item** property returns a string. For collections, it returns an item based on the specified key.

For the unnamed object, items are not named, so you cannot pass the name of the **WshUnnamed** argument.

The **WshUnnamed** argument items are stored in the order that they were included on the command line.

When entering the **WshUnnamed** argument at the command line, you can use blanks in string arguments if you enclose them in quotes.

The quotes are removed in the **WshUnnamed** collection.

Example

The following example demonstrates how the **Item** property can be used to return items in the **WshUnnamed** collection. In this example, two unnamed arguments are supplied to run a script. Inside the script, code causes the unnamed arguments to be output. The **Item** property is used to index into the unnamed arguments collection.

The following line is typed at the command prompt to run the script.

```
myScript.vbs arg1 arg2
```

When the following code is executed inside the script:

```
WScript.Echo WScript.Arguments.Unnamed.Item(0)
WScript.Echo WScript.Arguments.Unnamed.Item(1)
```

the following output is produced:

```
arg1
arg2
```

See Also

Arguments Property | WshUnnamed Object | WshUnnamed Object | Count Method | Item Property (WshNamed) | Exists Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

length Property (WshArguments object)

Returns the number of command-line parameters belonging to a script (the number of items in an argument's collection).

object.**length**

Arguments

object
WshArguments object.

Remarks

The **length** property is a read-only integer that you use in scripts when you write in JScript. It is similar to VBScript's **Count** method.

See Also

WshArguments Object | Arguments Property | Count Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

length Property (WshEnvironment object)

Returns the number of Windows environment variables on the local computer system (the number of items in an Environment collection).

object.**length**

Arguments

object
WshEnvironment object.

Remarks

The **length** property is a read-only integer that you use in scripts when you write in JScript. It is similar to VBScript's **Count** method.

See Also

WshEnvironment Object | Count Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

length Property (WshSpecialFolders object)

Returns the number of Windows special folders on the local computer system (the number of items in a SpecialFolders collection).

object.**length**

Arguments

object
WshSpecialFolders object.

Remarks

The **length** property is a read-only integer that you use in scripts when you write in JScript. It is similar to VBScript's **Count** method.

See Also

WshSpecialFolders Object | Count Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Line Property (WScript)

Returns the current line number in an input stream.

object.*strStream*.**Line**

Arguments

object
WScript object.
strStream
StdIn property.

Remarks

The **Line** property is a read-only integer.

After a stream is first opened, **Line** will initially be 1.

The **StdIn**, **StdOut**, and **StdErr** properties and methods work only when the script is run with CScript.exe. If the script is run with WScript.exe, an error occurs.

Example

The following code demonstrates the use of the Line property.

[VBScript]

```
Dim StdIn, StdOut
Set StdIn = WScript.StdIn
Set StdOut = WScript.StdOut

Do While Not StdIn.AtEndOfStream
     str = StdIn.ReadLine
     StdOut.WriteLine "Line " & (StdIn.Line - 1) & ": " & str
Loop
```

[JScript]

```
var stdin = WScript.StdIn;
var stdout = WScript.StdOut;

while (!stdin.AtEndOfStream)
{
     var str = stdin.ReadLine();
     stdout.WriteLine("Line " + (stdin.Line - 1) + ": " + str);
}
```

See Also

Error Messages | StdIn Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

Line Property (WshRemoteError)

Identifies the line in a script that contains an error.

Object.**Line**

Arguments

Object
WshRemoteError object.

Remarks

The **Line** property returns an unsigned long integer.

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).

Example

The following JScript code demonstrates how the **WshRemoteError** object exposes the line in which the error occurred.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
WScript.ConnectObject RemoteScript, "remote_"
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop

Sub remote_Error
    Dim theError
    Set theError = RemoteScript.Error
    WScript.Echo "Error - Line: " & theError.Line & ", Char: " & theError.Character & vbCrLf & "Description: " & t
    WScript.Quit -1
End Sub
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
WScript.ConnectObject(RemoteScript, "remote_");
```

```
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}

function remote_Error()
{
    var theError = RemoteScript.Error;
    WScript.Echo("Error - Line: " + theError.Line + ", Char: " + theError.Character + "\nDescription: " + theError
    WScript.Quit(-1);
}
```

See Also

WshRemote Object | WshRemoteError Object | Character Property | Description Property | Line Property | SourceText Property | Number Property | Source Property

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Name Property (WScript Object)

Returns the name of the **WScript** object (the host executable file).

object.**Name**

Arguments

object
WScript object.

Remarks

The **Name** property is a read-only string.

Example

The following JScript code shows how to use the **Name** property.

```
WScript.Echo (WScript.Name);
```

See Also

WScript Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Named Property

Returns the **WshNamed** object (a collection of named arguments).

```
Object.Named
```

Parameters

Object
WshArguments object.

Remarks

The Named property returns the WshNamed collection, the collection of named arguments. Use the name of the argument to find out if the argument was included and to access its value.

Example

The following code displays the number of named and unnamed command-line arguments.

```
<package>
<job id="JS">
<script language="JScript">

var argsNamed = WScript.Arguments.Named;
var argsUnnamed = WScript.Arguments.Unnamed;

WScript.Echo("There are " + argsNamed.length + " named arguments.");
WScript.Echo("There are " + argsUnnamed.length + " unnamed arguments.");

</script>
</job>

<job id="VBS">
<script language="VBScript">

Dim argsNamed, argsUnnamed
Set argsNamed = WScript.Arguments.Named
Set argsUnnamed = WScript.Arguments.Unnamed

WScript.Echo "There are " & argsNamed.Count & " named arguments."
WScript.Echo "There are " & argsUnnamed.Count & " unnamed arguments."

</script>
</job>
</package>
```

See Also

WshArguments Object | WshNamed Object | Unnamed Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

Number Property

Reports the error number representing a script error.

Object.**Number**

Arguments

Object
WshRemoteError object.

Remarks

That **Number** property returns an unsigned long integer.

Example

The following JScript code demonstrates how the **WshRemoteError** object exposes the error number.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
WScript.ConnectObject RemoteScript, "remote_"
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop
```

```
Sub remote_Error
    Dim theError
    Set theError = RemoteScript.Error
    WScript.Echo "Error " & theError.Number & " - Line: " & theError.Line & ", Char: " & theError.Character & vbCr
    WScript.Quit -1
End Sub
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
WScript.ConnectObject(RemoteScript, "remote_");
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}

function remote_Error()
{
    var theError = RemoteScript.Error;
    WScript.Echo("Error " + theError.Number + " - Line: " + theError.Line + ", Char: " + theError.Character + "\nD
    WScript.Quit(-1);
}
```

See Also

WshRemote Object | WshRemoteError Object | Description Property | Line Property | Character Property| SourceText Property| Source Property

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Path Property

Returns the name of the directory containing the host executable (CScript.exe or WScript.exe).

object.**Path**

Arguments

object
WScript object.

Remarks

The **Path** property is a read-only string.

Example

The following VBScript code echoes the directory where the executable file resides.

```
WScript.Echo (WScript.Path);
```

See Also

FullName Property | WScript Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

ProcessID Property

The process ID (PID) for a process started with the **WshScriptExec** object.

Object.**ProcessID**

Parameters

Object
WshScriptExec object.

Remarks

You can use the **ProcessID** property to activate an application (as an argument to the **AppActivate** method).

Example

The following code uses the **ProcessID** property to activate the calculator and notepad applications.

[VBScript]

```
Sub delayedSendKeys(str)
     WScript.Sleep 100
     WshShell.SendKeys str
End Sub

Dim WshShell, oCalc, oNotepad
Set WshShell = CreateObject("WScript.Shell")
Set oCalc = WshShell.Exec("calc")
Set oNotepad = WshShell.Exec("notepad")
WScript.Sleep 500

WshShell.AppActivate oCalc.ProcessID
delayedSendKeys "1{+}1~"
delayedSendKeys "^C"
delayedSendKeys "%{F4}"

WshShell.AppActivate oNotepad.ProcessID
delayedSendKeys "1 {+} 1 = ^V"
```

[JScript]

```
function delayedSendKeys(str)
{
     WScript.Sleep(100);
     WshShell.SendKeys(str);
}

var WshShell = new ActiveXObject("WScript.Shell");
var oCalc = WshShell.Exec("calc");
var oNotepad = WshShell.Exec("notepad");
WScript.Sleep(500);

WshShell.AppActivate(oCalc.ProcessID);
delayedSendKeys("1{+}1~");
delayedSendKeys("^C");
delayedSendKeys("%{F4}");

WshShell.AppActivate(oNotepad.ProcessID);
delayedSendKeys("1 {+} 1 = ^V");
```

See Also

WshScriptExec Object | AppActivate Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

RelativePath Property

Assigns a relative path to a shortcut, or identifies the relative path of a shortcut.

object.**RelativePath**

Arguments

object
WshShortcut object.

Remarks

String.

Example

The following code sets the relative path of a shortcut.

[VBScript]

```
Dim WshShell, WshShortcut
Set WshShell = WScript.CreateObject ("WScript.Shell")
Set WshShortcut = WshShell.CreateShortcut("MyScript.lnk")
WshShortcut.RelativePath = "C:\Scripts\"
```

[JScript]

```
var WshShell = WScript.CreateObject ("WScript.Shell");
var WshShortcut = WshShell.CreateShortcut("MyScript.lnk");
WshShortcut.RelativePath = "C:\\Scripts\\";
```

See Also

WshShortcut Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

ScriptFullName Property

Returns the full path of the currently running script.

object.**ScriptFullName**

Arguments

object
WScript object.

Remarks

The **ScriptFullName** property is a read-only string.

Example

The following example demonstrates the use of the ScriptFullName property.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         strDesktop = WshShell.SpecialFolders("Desktop")
         set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
         oShellLink.TargetPath = WScript.ScriptFullName
         oShellLink.WindowStyle = 1
         oShellLink.Hotkey = "Ctrl+Alt+e"
         oShellLink.IconLocation = "notepad.exe, 0"
         oShellLink.Description = "Shortcut Script"
         oShellLink.WorkingDirectory = strDesktop
         oShellLink.Save
         set oUrlLink = WshShell.CreateShortcut(strDesktop & "\Microsoft Web Site.url")
         oUrlLink.TargetPath = "http://www.microsoft.com"
         oUrlLink.Save
      </script>
```

```
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         strDesktop = WshShell.SpecialFolders("Desktop");
         var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
         oShellLink.TargetPath = WScript.ScriptFullName;
         oShellLink.WindowStyle = 1;
         oShellLink.Hotkey = "Ctrl+Alt+e";
         oShellLink.IconLocation = "notepad.exe, 0";
         oShellLink.Description = "Shortcut Script";
         oShellLink.WorkingDirectory = strDesktop;
         oShellLink.Save();
         var oUrlLink = WshShell.CreateShortcut(strDesktop + "\\Microsoft Web Site.url");
         oUrlLink.TargetPath = "http://www.microsoft.com";
         oUrlLink.Save();
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | ScriptName Property | WScript Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

ScriptName Property

Returns the file name of the currently running script.

object.**ScriptName**

Arguments

object
WScript object.

Remarks

The ScriptName property is a read-only string.

Example

The following VBScript code echoes the name of the script being run.

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

See Also

ScriptFullName Property | WScript Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Source Property

Identifies the COM object responsible for causing the script error.

```
Object.Source
```

Arguments

Object
WshRemoteError object.

Remarks

The **Source** property returns a string.

Example

The following JScript code demonstrates how the **WshRemoteError** object exposes the COM object responsible for causing the script error.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
WScript.ConnectObject RemoteScript, "remote_"
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop

Sub remote_Error
    Dim theError
    Set theError = RemoteScript.Error
    WScript.Echo "Error - Line: " & theError.Line & ", Char: " & theError.Character & vbCrLf & "Description: " & t
    WScript.Quit -1
End Sub
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
WScript.ConnectObject(RemoteScript, "remote_");
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}
```

```
function remote_Error()
{
    var theError = RemoteScript.Error;
    WScript.Echo("Error - Line: " + theError.Line + ", Char: " + theError.Character + "\nDescription: " + theError
    WScript.Quit(-1);
}
```

See Also

WshRemote Object | WshRemoteError Object | Description Property | Line Property | Character Property| SourceText Property | Number Property

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

SourceText Property

Contains the line of source code that caused an error.

Object.**SourceText**

Arguments

Object
WshRemoteError object.

Remarks

The **SourceText** property returns a string.

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

Example

The following JScript code demonstrates how the **WshRemoteError** object exposes the line of source code that caused an error.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
WScript.ConnectObject RemoteScript, "remote_"
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop

Sub remote_Error
    Dim theError
    Set theError = RemoteScript.Error
    WScript.Echo "Error - Line: " & theError.Line & ", Char: " & theError.Character & vbCrLf & "Description: " & t
    WScript.Quit -1
End Sub
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
WScript.ConnectObject(RemoteScript, "remote_");
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}

function remote_Error()
{
    var theError = RemoteScript.Error;
    WScript.Echo("Error - Line: " + theError.Line + ", Char: " + theError.Character + "\nDescription: " + theError
    WScript.Quit(-1);
}
```

See Also

WshRemote Object | WshRemoteError Object | Description Property | Line Property | Character Property| Number Property | Source Property

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

SpecialFolders Property

Returns a **SpecialFolders** object (a collection of special folders).

```
object.SpecialFolders(objWshSpecialFolders)
```

Arguments

object
WshShell object.
objWshSpecialFolders
The name of the special folder.

Remarks

The **WshSpecialFolders** object is a collection. It contains the entire set of Windows special folders, such as the Desktop folder, the Start Menu folder, and the Personal Documents folder. The special folder name is used to index into the collection to retrieve the special folder you want. The **SpecialFolders** property returns an empty string if the requested folder (*strFolderName*) is not available. For example, Windows 95 does not have an AllUsersDesktop folder and returns an empty string if *strFolderName* is AllUsersDesktop.

The following special folders are available:

- AllUsersDesktop

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

Example

The following example demonstrates the use of the SpecialFolders property:

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         strDesktop = WshShell.SpecialFolders("Desktop")
         set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
         oShellLink.TargetPath = WScript.ScriptFullName
         oShellLink.WindowStyle = 1
         oShellLink.Hotkey = "Ctrl+Alt+e"
         oShellLink.IconLocation = "notepad.exe, 0"
         oShellLink.Description = "Shortcut Script"
         oShellLink.WorkingDirectory = strDesktop
         oShellLink.Save
         set oUrlLink = WshShell.CreateShortcut(strDesktop & "\Microsoft Web Site.url")
         oUrlLink.TargetPath = "http://www.microsoft.com"
         oUrlLink.Save
      </script>
   </job>

   <job id="js">
```

```
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         strDesktop = WshShell.SpecialFolders("Desktop");
         var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
         oShellLink.TargetPath = WScript.ScriptFullName;
         oShellLink.WindowStyle = 1;
         oShellLink.Hotkey = "Ctrl+Alt+e";
         oShellLink.IconLocation = "notepad.exe, 0";
         oShellLink.Description = "Shortcut Script";
         oShellLink.WorkingDirectory = strDesktop;
         oShellLink.Save();
         var oUrlLink = WshShell.CreateShortcut(strDesktop + "\\Microsoft Web Site.url");
         oUrlLink.TargetPath = "http://www.microsoft.com";
         oUrlLink.Save();
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | WshSpecialFolders Object | WshShell Object

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Status Property (WshRemote)

Reports the current status of the remote script.

Object.**Status**

Arguments

Object
WshRemote object.

Remarks

The **Status** property returns a read-only enumerated data type.

Values

The **Status** property returns one of three possible values.

Return Value	Numeric value	Description
NoTask	0	The remote script object has been created but has not yet executed.
Running	1	The remote script object is currently running.
Finished	2	The remote script object has finished running.

Example

The following JScript code demonstrates how to use the **Status** property in a test block that checks to see if a remote script terminated normally.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
RemoteScript.Execute();
```

```
while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}
```

See Also

WshController Object | WshRemote Object | Error Property | Execute Method | Terminate Method | Start Event | End Event | Error Event

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Status Property (WshScriptExec)

Provides status information about a script run with the **Exec()** method.

Object.**Status**

Arguments

Object
WshScriptExec object.

Remarks

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

Return Values

The **Status** property returns a value from an enumerated type.

WshRunning (= 0)
The job is still running.
WshFinished (= 1)
The job has completed.

Example

The following code runs calc.exe and echoes the final status to the screen.

[VBScript]

```
Dim WshShell, oExec
Set WshShell = CreateObject("WScript.Shell")

Set oExec = WshShell.Exec("calc")

Do While oExec.Status = 0
     WScript.Sleep 100
Loop

WScript.Echo oExec.Status
```

[JScript]

```
var WshShell = new ActiveXObject("WScript.Shell");
var oExec = WshShell.Exec("calc");

while (oExec.Status == 0)
{
     WScript.Sleep(100);
}

WScript.Echo(oExec.Status);
```

See Also

Exec Method | WshScriptExec Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

StdErr Property (WScript)

Exposes the write-only error output stream for the current script.

object.**StdErr**

Arguments

object
WScript object.

Remarks

The **StdErr** property returns an object representing the standard error stream. The **StdIn**, **StdOut**, and **StdErr** streams can be accessed while using CScript.exe only. Attempting to access these streams while using WScript.exe produces an error.

See Also

Error Messages | WScript Object

© 2001 Microsoft Corporation. All rights reserved.

Build: Topic Version 5.6.9309.1546

Windows Script Host

StdErr Property (WshScriptExec)

Provides access to the **stderr** output stream of the **Exec** object.

Object.**StdErr**

Arguments

Object
WshScriptExec object.

Remarks

Use the StdErr property to retrieve data sent to the stderr stream from a process started with Exec.

Example

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

[VBScript]

```
Dim WshShell, oExec
Set WshShell = CreateObject("WScript.Shell")
Set oExec    = WshShell.Exec("%comspec% /c dire")

Function ReadAllFromAny(oExec)

     If Not oExec.StdOut.AtEndOfStream Then
          ReadAllFromAny = oExec.StdOut.ReadAll
          Exit Function
     End If

     If Not oExec.StdErr.AtEndOfStream Then
          ReadAllFromAny = "STDERR: " + oExec.StdErr.ReadAll
          Exit Function
     End If

     ReadAllFromAny = -1
End Function
```

```
Dim allInput, tryCount

allInput = ""
tryCount = 0

Do While True

     Dim input
     input = ReadAllFromAny(oExec)

     If -1 = input Then
          If tryCount > 10 And oExec.Status = 1 Then
               Exit Do
          End If
          tryCount = tryCount + 1
          WScript.Sleep 100
     Else
          allInput = allInput & input
          tryCount = 0
     End If
Loop

WScript.Echo allInput
```

[JScript]

```
var WshShell = new ActiveXObject("WScript.Shell");
var oExec    = WshShell.Exec("%comspec% /c dire");

function ReadAllFromAny(oExec)
{
     if (!oExec.StdOut.AtEndOfStream)
          return oExec.StdOut.ReadAll();

     if (!oExec.StdErr.AtEndOfStream)
          return "STDERR: " + oExec.StdErr.ReadAll();

     return -1;
}

var allInput = "";
var tryCount = 0;
```

```
while (true)
{
     var input = ReadAllFromAny(oExec);
     if (-1 == input)
     {
          if (tryCount++ > 10 && oExec.Status == 1)
               break;
          WScript.Sleep(100);
     }
     else
     {
          allInput += input;
          tryCount = 0;
     }
}

WScript.Echo(allInput);
```

See Also

WshScriptExec Object | StdIn Property | StdOut Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

StdIn Property (WScript)

Exposes the read-only input stream for the current script.

object.**StdIn**

Arguments

object
WScript object.

Remarks

The **StdIn** property returns an object representing the standard input stream. The **StdIn**, **StdOut**, and **StdErr** streams can be accessed while using CScript.exe only. Attempting to access these streams while using WScript.exe produces an error.

See Also

Error Messages | WScript Object | Read Method | ReadAll Method | ReadLine Method | Skip Method | SkipLine Method | AtEndOfLine Property | Close Method | Column Property | Line Property (WScript)

Build: Topic Version 5.6.9309.1546

Windows Script Host

StdIn Property (WshScriptExec)

Exposes the **stdin** input stream of the **Exec** object.

Object.**StdIn**

Arguments

Object
WshScriptExec object.

Remarks

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

Example

The following code starts a batch file and waits for the user input prompt. After entering the needed data through the StdIn stream, the batch file will be able to complete.

[VBScript]

```
Dim WshShell, oExec, input
Set WshShell = CreateObject("WScript.Shell")
Set oExec    = WshShell.Exec("test.bat")
input = ""

Do While True

     If Not oExec.StdOut.AtEndOfStream Then
          input = input & oExec.StdOut.Read(1)
          If InStr(input, "Press any key") <> 0 Then Exit Do
     End If
     WScript.Sleep 100
Loop

oExec.StdIn.Write VbCrLf

Do While oExec.Status <> 1
     WScript.Sleep 100
Loop
```

[JScript]

```
var WshShell = new ActiveXObject("WScript.Shell");
var oExec    = WshShell.Exec("test.bat");
var input = "";

while (true)
{
     if (!oExec.StdOut.AtEndOfStream)
     {
```

```
          input += oExec.StdOut.Read(1);
          if (input.indexOf("Press any key") != -1)
               break;
     }
     WScript.Sleep(100);
}

oExec.StdIn.Write("\n");

while (oExec.Status != 1)
     WScript.Sleep(100);
```

See Also

WshScriptExec Object | StdIn Property | StdErr Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

StdOut Property (WScript)

Exposes the write-only output stream for the current script.

object.**StdOut**

Arguments

object
WScript object.

Remarks

The **StdOut** property returns an object representing the standard output stream. The **StdIn**, **StdOut**, and **StdErr** streams can be accessed while using CScript.exe only. Attempting to access these streams while using WScript.exe produces an error.

See Also

Error Messages | WScript Object | Write Method | WriteBlankLines Method | WriteLine Method | Close Method | Column Property | Line Property (WScript)

Build: Topic Version 5.6.9309.1546

Windows Script Host

StdOut Property (WshScriptExec)

Exposes the write-only **stdout** output stream of the **Exec** object.

Object.**StdOut**

Arguments

Object
WshScriptExec object.

Remarks

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

Example

The following code starts a batch file and waits for the user input prompt. After entering the needed data through the StdIn stream, the batch

file will be able to complete.

[VBScript]

```
Dim WshShell, oExec, input
Set WshShell = CreateObject("WScript.Shell")
Set oExec    = WshShell.Exec("test.bat")
input = ""

Do While True

     If Not oExec.StdOut.AtEndOfStream Then
          input = input & oExec.StdOut.Read(1)
          If InStr(input, "Press any key") <> 0 Then Exit Do
     End If
     WScript.Sleep 100
Loop

oExec.StdIn.Write VbCrLf

Do While oExec.Status <> 1
     WScript.Sleep 100
Loop
```

[JScript]

```
var WshShell = new ActiveXObject("WScript.Shell");
var oExec    = WshShell.Exec("test.bat");
var input = "";

while (true)
{
     if (!oExec.StdOut.AtEndOfStream)
     {
          input += oExec.StdOut.Read(1);
          if (input.indexOf("Press any key") != -1)
               break;
     }
     WScript.Sleep(100);
}

oExec.StdIn.Write("\n");
```

```
while (oExec.Status != 1)
     WScript.Sleep(100);
```

See Also

WshScriptExec Object | StdIn Property | StdErr Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

TargetPath Property

The path to the shortcut's executable.

object.**TargetPath**

Arguments

object
WshShortcut or **WshUrlShortcut** object.

Remarks

String.

This property is for the shortcut's target path only. Any arguments to the shortcut must be placed in the Argument's property.

Example

The following example demonstrates the use of the TargetPath property:

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         strDesktop = WshShell.SpecialFolders("Desktop")
         set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
         oShellLink.TargetPath = WScript.ScriptFullName
         oShellLink.WindowStyle = 1
         oShellLink.Hotkey = "Ctrl+Alt+e"
         oShellLink.IconLocation = "notepad.exe, 0"
         oShellLink.Description = "Shortcut Script"
         oShellLink.WorkingDirectory = strDesktop
         oShellLink.Save
         set oUrlLink = WshShell.CreateShortcut(strDesktop & "\Microsoft Web Site.url")
         oUrlLink.TargetPath = "http://www.microsoft.com"
         oUrlLink.Save
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         strDesktop = WshShell.SpecialFolders("Desktop");
         var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
         oShellLink.TargetPath = WScript.ScriptFullName;
         oShellLink.WindowStyle = 1;
         oShellLink.Hotkey = "Ctrl+Alt+e";
         oShellLink.IconLocation = "notepad.exe, 0";
         oShellLink.Description = "Shortcut Script";
         oShellLink.WorkingDirectory = strDesktop;
         oShellLink.Save();
         var oUrlLink = WshShell.CreateShortcut(strDesktop + "\\Microsoft Web Site.url");
         oUrlLink.TargetPath = "http://www.microsoft.com";
         oUrlLink.Save();
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | WshShortcut Object | WshUrlShortcut Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unnamed Property

Returns the **WshUnnamed** object (a collection of unnamed arguments).

Object.**Unnamed**

Parameters

Object
WshArguments object.

Remarks

The Unnamed property returns the WshUnnamed collection, the collection of unnamed arguments. The arguments are presented in the order that they were entered on the command line. The first unnamed argument at index 0.

Example

The following code displays the number of named and unnamed command-line arguments.

```
<package>
<job id="JS">
<script language="JScript">

var argsNamed = WScript.Arguments.Named;
var argsUnnamed = WScript.Arguments.Unnamed;

WScript.Echo("There are " + argsNamed.length + " named arguments.");
```

```
WScript.Echo("There are " + argsUnnamed.length + " unnamed arguments.");

</script>
</job>

<job id="VBS">
<script language="VBScript">

Dim argsNamed, argsUnnamed
Set argsNamed = WScript.Arguments.Named
Set argsUnnamed = WScript.Arguments.Unnamed

WScript.Echo "There are " & argsNamed.Count & " named arguments."
WScript.Echo "There are " & argsUnnamed.Count & " unnamed arguments."

</script>
</job>
</package>
```

See Also

WshArguments Object | WshUnnamed Object | Named Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

UserDomain Property

Returns a user's domain name.

object.**UserDomain**

Arguments

object
WshNetwork object.

Remarks

String.

The UserDomain property does not work on Windows98 and Windows ME unless the USERDOMAIN environment variable is set. The variable is not set by default.

Example

The following example demonstrates the use of the UserDomain property:

```
<package>
   <job id="vbs">
      <script language="VBScript">
         Set WshNetwork = WScript.CreateObject("WScript.Network")
         WScript.Echo "Domain = " & WshNetwork.UserDomain
         WScript.Echo "Computer Name = " & WshNetwork.ComputerName
         WScript.Echo "User Name = " & WshNetwork.UserName
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshNetwork = WScript.CreateObject("WScript.Network");
         WScript.Echo("Domain = " + WshNetwork.UserDomain);
         WScript.Echo("Computer Name = " + WshNetwork.ComputerName);
         WScript.Echo("User Name = " + WshNetwork.UserName);
         }
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | WshNetwork Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

UserName Property

Returns the name of a user.

object.**UserName**

Arguments

object
WshNetwork object.

Remarks

String.

If you are using this property in a login script, see Creating an Automated Login Script.

Example

The following example demonstrates the use of the UserName property:

```
<package>
   <job id="vbs">
      <script language="VBScript">
         Set WshNetwork = WScript.CreateObject("WScript.Network")
         WScript.Echo "Domain = " & WshNetwork.UserDomain
         WScript.Echo "Computer Name = " & WshNetwork.ComputerName
         WScript.Echo "User Name = " & WshNetwork.UserName
```

```
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshNetwork = WScript.CreateObject("WScript.Network");
         WScript.Echo("Domain = " + WshNetwork.UserDomain);
         WScript.Echo("Computer Name = " + WshNetwork.ComputerName);
         WScript.Echo("User Name = " + WshNetwork.UserName);
         }
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | WshNetwork Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Version Property

Returns the version of Windows Script Host.

object.**Version**

Arguments

object
WScript object.

Remarks

String.

Example

The following VBScript code echoes the current version of Windows Script Host.

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

See Also

WScript Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

WindowStyle Property

Assigns a window style to a shortcut, or identifies the type of window style used by a shortcut.

```
object.WindowStyle = intWindowStyle
```

Arguments

object
WshShortcut object.
intWindowStyle
Sets the window style for the program being run.

Remarks

The WindowStyle property returns an integer.

The following table lists the available settings for *intWindowStyle.*

intWindowStyle	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.

Example

The following example demonstrates the use of the WindowStyle property:

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         strDesktop = WshShell.SpecialFolders("Desktop")
         set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
         oShellLink.TargetPath = WScript.ScriptFullName
         oShellLink.WindowStyle = 1
         oShellLink.Hotkey = "Ctrl+Alt+e"
         oShellLink.IconLocation = "notepad.exe, 0"
         oShellLink.Description = "Shortcut Script"
         oShellLink.WorkingDirectory = strDesktop
         oShellLink.Save
         set oUrlLink = WshShell.CreateShortcut(strDesktop & "\Microsoft Web Site.url")
         oUrlLink.TargetPath = "http://www.microsoft.com"
         oUrlLink.Save
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         strDesktop = WshShell.SpecialFolders("Desktop");
         var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
```

```
        oShellLink.TargetPath = WScript.ScriptFullName;
        oShellLink.WindowStyle = 1;
        oShellLink.Hotkey = "Ctrl+Alt+e";
        oShellLink.IconLocation = "notepad.exe, 0";
        oShellLink.Description = "Shortcut Script";
        oShellLink.WorkingDirectory = strDesktop;
        oShellLink.Save();
        var oUrlLink = WshShell.CreateShortcut(strDesktop + "\\Microsoft Web Site.url");
        oUrlLink.TargetPath = "http://www.microsoft.com";
        oUrlLink.Save();
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | WshShortcut Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

WorkingDirectory Property

Assign a working directory to a shortcut, or identifies the working directory used by a shortcut.

object.**WorkingDirectory** = *strWorkingDirectory*

Arguments

object
WshShortcut object.
strWorkingDirectory

String. Directory in which the shortcut starts.

Remarks

String.

Example

The following example demonstrates the use of the WorkingDirectory property:

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         strDesktop = WshShell.SpecialFolders("Desktop")
         set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
         oShellLink.TargetPath = WScript.ScriptFullName
         oShellLink.WindowStyle = 1
         oShellLink.Hotkey = "Ctrl+Alt+e"
         oShellLink.IconLocation = "notepad.exe, 0"
         oShellLink.Description = "Shortcut Script"
         oShellLink.WorkingDirectory = strDesktop
         oShellLink.Save
         set oUrlLink = WshShell.CreateShortcut(strDesktop & "\Microsoft Web Site.url")
         oUrlLink.TargetPath = "http://www.microsoft.com"
         oUrlLink.Save
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         strDesktop = WshShell.SpecialFolders("Desktop");
         var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
         oShellLink.TargetPath = WScript.ScriptFullName;
         oShellLink.WindowStyle = 1;
         oShellLink.Hotkey = "Ctrl+Alt+e";
         oShellLink.IconLocation = "notepad.exe, 0";
         oShellLink.Description = "Shortcut Script";
         oShellLink.WorkingDirectory = strDesktop;
         oShellLink.Save();
         var oUrlLink = WshShell.CreateShortcut(strDesktop + "\\Microsoft Web Site.url");
```

```
        oUrlLink.TargetPath = "http://www.microsoft.com";
        oUrlLink.Save();
      </script>
   </job>
</package>
```

See Also

Running Your Scripts

Build: Topic Version 5.6.9309.1546

Windows Script Host

Methods

In this Section

AddPrinterConnection Method
: Adds a DOS-style printer connection to your computer.

AddWindowsPrinterConnection Method
: Adds a Windows-style printer connection to your computer.

AppActivate Method
: Activates an application window.

Close Method
: Closes an open stream.

ConnectObject Method
: Connects an object's event sources to functions with a given prefix.

Count Method
: Returns the number of switches in the **WshNamed** or **WshUnnamed** objects.

CreateObject Method

Creates an object specified by the *strProgID* parameter.

CreateScript Method

Creates a **WshRemote** object (an object that represents an instance of a script running in a remote process).

CreateShortcut Method

Creates an object reference to a shortcut or URLshortcut.

DisconnectObject Method

Disconnects a previously connected object from Windows Script Host.

Echo Method

Sends output to a dialog box or the console.

EnumNetworkDrives Method

Returns the current network drive mappings.

EnumPrinterConnections Method

Returns the current network printer mappings.

Exec Method

Runs an application in a child command-shell, providing access to the stdin/stdout/stderr channels, and the sharing of environment variables.

Execute Method

Starts execution of a remote script object.

Exists Method

Indicates whether a specific key value exists in the **WshNamed** object.

ExpandEnvironmentStrings Method

Expands the requested environment variable from the running process and returns the result string.

GetObject Method

Retrieves an **Automation** object from a file or an object specified by the *strProgID* parameter.

getResource Method

Returns the value of a resource defined with the resource element.

LogEvent Method

Logs an event in the Windows NT event log or WSH.log file.

MapNetworkDrive Method

Maps the share point specified by *strRemoteName* to the local resource name *strLocalName*.

Popup Method

Displays a pop-up message box window that contains the message contained in *strText*.

Quit Method

Quits execution with a specified error code.

Read Method

Reads a specified number of characters from an input stream and returns the resulting string.

ReadAll Method

Reads an entire input stream and returns the resulting string.

ReadLine Method

Reads an entire line (up to, but not including, the newline character) from an input stream and returns the resulting string.

RegDelete Method

Deletes from the registry the key or value named by *strName.*

RegRead Method

Returns the registry key or value named by *strName.*

RegWrite Method

Sets the registry key or value named by *strName.*

Remove Method

Deletes the environment variable specified by *strName.*

RemoveNetworkDrive Method

Removes the current resource connection denoted by *strName.*

RemovePrinterConnection Method

Removes the current resource connection denoted by *strName.*

Run Method

Creates a new process that executes *strCommand*.

Save Method

Saves a shortcut to the specified location.

SendKeys Method

Sends one or more keystrokes to the active window (as if typed on the keyboard).

SetDefaultPrinter Method

Sets the default printer to the remote printer specified.

ShowUsage Method

Displays information about how a script should be used.

Skip Method

Skips a specified number of characters when reading an input stream.

SkipLine Method

Skips the next line when reading an input stream.

Sleep Method

Places the script process into an inactive state for the number of milliseconds specified and then continues execution.

Terminate Method (WshScriptExec)

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

Write Method

Writes a specified string to an output stream.

WriteBlankLines Method

Writes a specified number of newline characters to an output stream.

WriteLine Method
Writes a specified string and newline character to an output stream.

Related Sections

WSH Language
List of elements that make up WSH Reference.
WSH Basics
Learn the basics of WSH.

Build: Topic Version 5.6.9309.1546

Windows Script Host

AddPrinterConnection Method

Adds a remote MS-DOS-based printer connection to your computer system.

```
object.AddPrinterConnection(strLocalName, strRemoteName[,bUpdateProfile][,strUser][,strPassword])
```

Arguments

object
WshNetwork object.
strLocalName
Sting value indicating the local name to assign to the connected printer.
strRemoteName
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**.

strUser
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 *strUser* and *strPassword.*

strPassword
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 *strUser* and *strPassword.*

Remarks

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.

Example

The following code uses the **AddPrinterConnection** method to connect a network printer to LPT1.

[VBScript]

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
WshNetwork.AddPrinterConnection "LPT1", "\\Server\Print1"
```

[JScript]

```
var WshNetwork = WScript.CreateObject("WScript.Network");
WshNetwork.AddPrinterConnection ("LPT1", "\\\\Server\\Print1");
```

See Also

WshNetwork Object | AddWindowsPrinterConnection Method | EnumPrinterConnections Method | RemovePrinterConnection Method | SetDefaultPrinter Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

AddWindowsPrinterConnection Method

Adds a Windows-based printer connection to your computer system.

```
Windows NT/2000:
object.AddWindowsPrinterConnection(
   strPrinterPath
)

Windows 9x/Me:
object.AddWindowsPrinterConnection(
   strPrinterPath,
   strDriverName[,strPort]
)
```

Arguments

object
: **WshNetwork** object.

strPrinterPath
: String value indicating the path to the printer connection.

strDriverName
: String value indicating the name of the driver (ignored if used on Windows NT/Windows 2000).

strPort
: Optional. String value specifying a printer port for the printer connection (ignored on Windows NT/Windows 2000).

Remarks

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. In Windows 9x/Me, the printer driver must already be installed on the machine for the **AddWindowsPrinterConnection** method to work. If the driver is not installed, Windows returns an error message.

Example 1

The following code uses the **AddWindowsPrinterConnection** method to connect a network printer to a Windows NT/2000 computer system.

[VBScript]

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
PrinterPath = "\\printserv\DefaultPrinter"
WshNetwork.AddWindowsPrinterConnection PrinterPath
```

[JScript]

```
var WshNetwork = WScript.CreateObject("WScript.Network");
var PrinterPath = "\\\\printserv\\DefaultPrinter";
WshNetwork.AddWindowsPrinterConnection(PrinterPath);
```

Example 2

The following code uses the **AddWindowsPrinterConnection** method to connect a network printer to a Windows 9x/Me computer system.

[VBScript]

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
PrinterPath = "\\printserv\DefaultPrinter"
PrinterDriver = "Lexmark Optra S 1650"
WshNetwork.AddWindowsPrinterConnection PrinterPath, PrinterDriver
```

[JScript]

```
var WshNetwork = WScript.CreateObject("WScript.Network");
var PrinterPath = "\\\\printserv\\DefaultPrinter";
var PrinterDriver = "Lexmark Optra S 1650";
WshNetwork.AddWindowsPrinterConnection(PrinterPath, PrinterDriver);
```

See Also

WshNetwork Object | AddPrinterConnection Method | EnumPrinterConnections Method | RemovePrinterConnection Method | SetDefaultPrinter Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

AppActivate Method

Activates an application window.

object.**AppActivate** *title*

Arguments

object
WshShell object.
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.

Remarks

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 example demonstrates the use of a single .wsf file for two jobs in different script languages (VBScript and JScript). The

functionality of both jobs is the same — each runs the Windows calculator and sends it keystrokes to execute a simple calculation.

The following example starts the Windows calculator and uses **AppActivate** to ensure that the calculator is at the top.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         WshShell.Run "calc"
         WScript.Sleep 100
         WshShell.AppActivate "Calculator"
         WScript.Sleep 100
         WshShell.SendKeys "1{+}"
         WScript.Sleep 500
         WshShell.SendKeys "2"
         WScript.Sleep 500
         WshShell.SendKeys "~"
         WScript.Sleep 500
         WshShell.SendKeys "*3"
         WScript.Sleep 500
         WshShell.SendKeys "~"
         WScript.Sleep 2500
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         WshShell.Run("calc");
         WScript.Sleep(100);
         WshShell.AppActivate("Calculator");
         WScript.Sleep(100);
         WshShell.SendKeys("1{+}");
         WScript.Sleep(500);
         WshShell.SendKeys("2");
         WScript.Sleep(500);
         WshShell.SendKeys("~");
         WScript.Sleep(500);
         WshShell.SendKeys("*3");
         WScript.Sleep(500);
         WshShell.SendKeys("~");
         WScript.Sleep(2500);
      </script>
```

```
   </job>
</package>
```

See Also

Running Your Scripts | WshShell Object | SendKeys Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Close Method

Closes a text stream.

object.**Close**

Arguments

object
StdIn, **StdOut**, or **StdErr** text stream objects.

Remarks

The **StdIn**, **StdOut,** and **StdErr** properties and methods work when running the script with the CScript.exe host executable file only. An error is returned when run with WScript.exe. It is not necessary to close standard streams; they close automatically when the process ends. If you close a standard stream, be aware that any other pointers to that standard stream become invalid. This method is provided for compatibility with the **TextStream** object.

See Also

StdErr Property | StdIn Property | StdOut Property | Error Messages

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

ConnectObject Method

Connects the object's event sources to functions with a given prefix.

```
object.ConnectObject(strObject, strPrefix)
```

Arguments

object
WScript object.
strObject
Required. String value indicating the name of the object you want to connect.
strPrefix
Required. String value indicating the function prefix.

Remarks

Connected objects are useful when you want to sync an object's events. The **ConnectObject** method connects the object's outgoing interface to the script file after creating the object. Event functions are a combination of this prefix and the event name.

Example

The following example demonstrates using the **ConnectObject** method to connect to the WshRemote object's Error event.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
WScript.ConnectObject RemoteScript, "remote_"
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop

Sub remote_Error
    Dim theError
    Set theError = RemoteScript.Error
    WScript.Echo "Error " & theError.Number & " - Line: " & theError.Line & ", Char: " & theError.Character & vbCr
    WScript.Quit -1
End Sub
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
WScript.ConnectObject(RemoteScript, "remote_");
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}

function remote_Error()
{
    var theError = RemoteScript.Error;
    WScript.Echo("Error " + theError.Number + " - Line: " + theError.Line + ", Char: " + theError.Character + "\nD
    WScript.Quit(-1);
}
```

See Also

WScript Object | DisconnectObject Method | CreateObject Method | GetObject Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Count Method

Returns the number of switches in the **WshNamed** or **WshUnnamed** objects.

object.**Count**

Arguments

object
Arguments object.

Remarks

The **Count** method returns an integer value. The Count method is primarily intended for VBScript users. JScript users should generally use the length property instead.

Example (WshNamed)

The following example demonstrates the Count method using the WshNamed object. Begin by typing the following text at the Command Prompt.

```
myScript.vbs /c:"WSH is a wonderful thing" /s:"scripts are wonderful"
```

Next, add the following VBScript code.

```
For i = 0 to WScript.Arguments.Count-1
    WScript.Echo WScript.Arguments.Named(i)
next i
```

Following is the result.

```
WSH is a wonderful thing
scripts are wonderful
```

Example (WshUnnamed)

The following example demonstrates the Count method using the WshUnnamed object. Begin by typing the following text at the command line.

```
myscript.vbs "WSH is a wonderful thing" "scripts are wonderful"
```

Next, add the following VBScript code.

```
For i = 0 to WScript.Arguments.Count-1
    WScript.Echo WScript.Arguments.Unnamed(i)
next i
```

Following is the result.

```
WSH is a wonderful thing
scripts are wonderful
```

See Also

Arguments Property | WshNamed Object | WshUnnamed Object | Item Property | Exists Method | Length Property

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

CreateObject Method

Creates a COM object.

```
object.CreateObject(strProgID[,strPrefix])
```

Arguments

object
WScript object.
strProgID
String value indicating the programmatic identifier (ProgID) of the object you want to create.
strPrefix
Optional. String value indicating the function prefix.

Remarks

Objects created with the **CreateObject** method using the *strPrefix* argument are connected objects. These are useful when you want to sync an object's events. The object's outgoing interface is connected to the script file after the object is created. Event functions are a combination of this prefix and the event name. If you create an object and do not supply the *strPrefix* argument, you can still sync events on the object by using the **ConnectObject** method. When the object fires an event, WSH calls a subroutine with *strPrefix* attached to the beginning of the event name. For example, if *strPrefix* is *MYOBJ* and the object fires an event named *OnBegin*, Windows Script Host calls the *MYOBJ_OnBegin* subroutine located in the script. The **CreateObject** method returns a pointer to the object's IDispatch interface.

Example

The following VBScript code uses the **CreateObject** method to create a **WshNetwork** object:

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
```

See Also

WScript Object | GetObject Method | ConnectObject Method | DisconnectObject Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

CreateScript Method

Creates a **WshRemote** object.

object.**CreateScript(***CommandLine*,[*MachineName*])

Parameters

object
WshController Object.
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).

Remarks

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 example demonstrates how the **CreateScript** method of the **WshController** object is used to create a **WshRemote** object (an instance of a remote script).

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
WScript.ConnectObject RemoteScript, "remote_"
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop

WScript.DisconnectObject RemoteScript

Sub remote_Error
    Dim theError
    Set theError = RemoteScript.Error
    WScript.Echo "Error " & theError.Number & " - Line: " & theError.Line & ", Char: " & theError.Character & vbCr
    WScript.Quit -1
End Sub
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
WScript.ConnectObject(RemoteScript, "remote_");
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}

WScript.DisconnectObject(RemoteScript);

function remote_Error()
{
    var theError = RemoteScript.Error;
    WScript.Echo("Error " + theError.Number + " - Line: " + theError.Line + ", Char: " + theError.Character + "\nD
    WScript.Quit(-1);
}
```

See Also

CreateObject Method | CreateScript Method | WshRemote Object | Execute Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

CreateShortcut Method

Creates a new shortcut, or opens an existing shortcut.

object.**CreateShortcut**(*strPathname*)

Arguments

object
WshShell object.
strPathname
String value indicating the pathname of the shortcut to create.

Remarks

The **CreateShortcut** method returns either a **WshShortcut** object or a **WshURLShortcut** object. 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:

1. Create an instance of a WshShortcut object.
2. Initialize its properties.
3. Save it to disk with the **Save** method.

> **Note** A common problem is putting arguments in the TargetPath property of the shortcut object, which doesn't work. All arguments to the shortcut must be put in the Arguments property.

Example

The following example creates a **WshShell** object and uses the **CreateShortcut** method to create two shortcuts.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         strDesktop = WshShell.SpecialFolders("Desktop")
         set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
         oShellLink.TargetPath = WScript.ScriptFullName
         oShellLink.WindowStyle = 1
         oShellLink.Hotkey = "CTRL+SHIFT+F"
         oShellLink.IconLocation = "notepad.exe, 0"
         oShellLink.Description = "Shortcut Script"
         oShellLink.WorkingDirectory = strDesktop
         oShellLink.Save
         set oUrlLink = WshShell.CreateShortcut(strDesktop & "\Microsoft Web Site.url")
         oUrlLink.TargetPath = "http://www.microsoft.com"
         oUrlLink.Save
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         strDesktop = WshShell.SpecialFolders("Desktop");
         var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
         oShellLink.TargetPath = WScript.ScriptFullName;
         oShellLink.WindowStyle = 1;
         oShellLink.Hotkey = "CTRL+SHIFT+F";
         oShellLink.IconLocation = "notepad.exe, 0";
         oShellLink.Description = "Shortcut Script";
         oShellLink.WorkingDirectory = strDesktop;
         oShellLink.Save();
         var oUrlLink = WshShell.CreateShortcut(strDesktop + "\\Microsoft Web Site.url");
         oUrlLink.TargetPath = "http://www.microsoft.com";
         oUrlLink.Save();
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | WshShortcut Object | WshUrlShortcut Object | WshShell Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

DisconnectObject Method

Disconnects a connected object's event sources.

object.**DisconnectObject**(*obj*)

Arguments

object
WScript object.
obj
String value indicating the name of the object to disconnect.

Remarks

Once an object has been "disconnected," WSH will not respond to its events. The object is still capable of firing events, though. Note that the **DisconnectObject** method does nothing if the specified object is not already connected.

Example

The following example demonstrates using the **DisconnectObject** method to disconnect to the WshRemote object's Error event after a remote script has completed.

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
```

```
Set RemoteScript = Controller.CreateScript("test.js", "remoteserver")
WScript.ConnectObject RemoteScript, "remote_"
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop

WScript.DisconnectObject RemoteScript

Sub remote_Error
    Dim theError
    Set theError = RemoteScript.Error
    WScript.Echo "Error " & theError.Number & " - Line: " & theError.Line & ", Char: " & theError.Character & vbCr
    WScript.Quit -1
End Sub
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("test.js", "remoteserver");
WScript.ConnectObject(RemoteScript, "remote_");
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}

WScript.DisconnectObject(RemoteScript)

function remote_Error()
{
    var theError = RemoteScript.Error;
    WScript.Echo("Error " + theError.Number + " - Line: " + theError.Line + ", Char: " + theError.Character + "\nD
    WScript.Quit(-1);
}
```

See Also

WScript Object | ConnectObject Method | CreateObject Method | GetObject Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Echo Method

Outputs text to either a message box or the command console window.

object.**Echo** [*Arg1*] [,*Arg2*] [,*Arg3*] ...

Arguments

object
WScript object.
Arg1, *Arg2*, *Arg3* ...
Optional. String value indicating the list of items to be displayed.

Remarks

The **Echo** method behaves differently depending on which WSH engine you are using.

WSH engine	Text Output
Wscript.exe	graphical message box
Cscript.exe	command console window

Each displayed item is separated with a space character. When using CScript.exe, each item is displayed with a newline character. If no items are provided as arguments to the **Echo** method, a blank line is output.

Example

The following example uses the **Echo** Method to display the domain name, computer name, and user name for the current machine, and to

display network mapping information for the drives and printers.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         Set WshNetwork = WScript.CreateObject("WScript.Network")
         Set oDrives = WshNetwork.EnumNetworkDrives
         Set oPrinters = WshNetwork.EnumPrinterConnections
         WScript.Echo "Domain = " & WshNetwork.UserDomain
         WScript.Echo "Computer Name = " & WshNetwork.ComputerName
         WScript.Echo "User Name = " & WshNetwork.UserName
         WScript.Echo
         WScript.Echo "Network drive mappings:"
         For i = 0 to oDrives.Count - 1 Step 2
            WScript.Echo "Drive " & oDrives.Item(i) & " = " & oDrives.Item(i+1)
         Next
         WScript.Echo
         WScript.Echo "Network printer mappings:"
         For i = 0 to oPrinters.Count - 1 Step 2
            WScript.Echo "Port " & oPrinters.Item(i) & " = " & oPrinters.Item(i+1)
         Next
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshNetwork = WScript.CreateObject("WScript.Network");
         var oDrives = WshNetwork.EnumNetworkDrives();
         var oPrinters = WshNetwork.EnumPrinterConnections();
         WScript.Echo("Domain = " + WshNetwork.UserDomain);
         WScript.Echo("Computer Name = " + WshNetwork.ComputerName);
         WScript.Echo("User Name = " + WshNetwork.UserName);
         WScript.Echo();
         WScript.Echo("Network drive mappings:");
         for(i=0; i<oDrives.Count(); i+=2){
            WScript.Echo("Drive " + oDrives.Item(i) + " = " + oDrives.Item(i+1));
         }
         WScript.Echo();
         WScript.Echo("Network printer mappings:");
         for(i=0; i<oPrinters.Count(); i+=2){
            WScript.Echo("Port " + oPrinters.Item(i) + " = " + oPrinters.Item(i+1));
         }
      </script>
   </job>
```

```
</package>
```

See Also

Running Your Scripts | WScript Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

EnumNetworkDrives Method

Returns the current network drive mapping information.

objDrives = *object*.**EnumNetworkDrives**

Arguments

object
WshNetwork object.
objDrives
Variable that holds the network drive mapping information.

Remarks

The **EnumNetworkDrives** method returns a collection. This collection is an array that associates pairs of items — network drive local names and their associated UNC names. Even-numbered items in the collection represent local names of logical drives. Odd-numbered items represent the associated UNC share names. The first item in the collection is at index zero (0).

Example

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

```
<package>
   <job id="vbs">
      <script language="VBScript">
         Set WshNetwork = WScript.CreateObject("WScript.Network")
         Set oDrives = WshNetwork.EnumNetworkDrives
         Set oPrinters = WshNetwork.EnumPrinterConnections
         WScript.Echo "Network drive mappings:"
         For i = 0 to oDrives.Count - 1 Step 2
            WScript.Echo "Drive " & oDrives.Item(i) & " = " & oDrives.Item(i+1)
         Next
         WScript.Echo
         WScript.Echo "Network printer mappings:"
         For i = 0 to oPrinters.Count - 1 Step 2
            WScript.Echo "Port " & oPrinters.Item(i) & " = " & oPrinters.Item(i+1)
         Next
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshNetwork = WScript.CreateObject("WScript.Network");
         var oDrives = WshNetwork.EnumNetworkDrives();
         var oPrinters = WshNetwork.EnumPrinterConnections();
         WScript.Echo("Network drive mappings:");
         for(i = 0; i < oDrives.length; i += 2) {
            WScript.Echo("Drive " + oDrives.Item(i) + " = " + oDrives.Item(i + 1));
         }
         WScript.Echo();
         WScript.Echo("Network printer mappings:");
         for(i = 0; i < oPrinters.length; i += 2) {
            WScript.Echo("Port " + oPrinters.Item(i) + " = " + oPrinters.Item(i + 1));
         }
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | WshNetwork Object | MapNetworkDrive Method | RemoveNetworkDrive Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

EnumPrinterConnections Method

Returns the current network printer mapping information.

objPrinters = *object*.**EnumPrinterConnections**

Arguments

object
WshNetwork object.
objPrinters
Variable that holds the network printer mapping information.

Remarks

The **EnumPrinterConnections** method returns a collection. This collection is an array that associates pairs of items — network printer local names and their associated UNC names. Even-numbered items in the collection represent printer ports. Odd-numbered items represent the networked printer UNC names. The first item in the collection is at index zero (0).

Example

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

```
<package>
   <job id="vbs">
      <script language="VBScript">
```

```
         Set WshNetwork = WScript.CreateObject("WScript.Network")
         Set oDrives = WshNetwork.EnumNetworkDrives
         Set oPrinters = WshNetwork.EnumPrinterConnections
         WScript.Echo "Network drive mappings:"
         For i = 0 to oDrives.Count - 1 Step 2
            WScript.Echo "Drive " & oDrives.Item(i) & " = " & oDrives.Item(i+1)
         Next
         WScript.Echo
         WScript.Echo "Network printer mappings:"
         For i = 0 to oPrinters.Count - 1 Step 2
            WScript.Echo "Port " & oPrinters.Item(i) & " = " & oPrinters.Item(i+1)
         Next
   </script>
</job>

   <job id="js">
      <script language="JScript">
         var WshNetwork = WScript.CreateObject("WScript.Network");
         var oDrives = WshNetwork.EnumNetworkDrives();
         var oPrinters = WshNetwork.EnumPrinterConnections();
         WScript.Echo("Network drive mappings:");
         for(i = 0; i < oDrives.length; i += 2) {
            WScript.Echo("Drive " + oDrives.Item(i) + " = " + oDrives.Item(i + 1));
         }
         WScript.Echo();
         WScript.Echo("Network printer mappings:");
         for(i = 0; i < oPrinters.length; i += 2) {
            WScript.Echo("Port " + oPrinters.Item(i) + " = " + oPrinters.Item(i + 1));
         }
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | WshNetwork Object | AddPrinterConnection Method | AddWindowsPrinterConnection Method | RemovePrinterConnection Method | SetDefaultPrinter Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Exec Method

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

object.**Exec**(*strCommand*)

Arguments

object
WshShell object.
strCommand
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.

Remarks

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).

Example

The following example demonstrates the basics of the **Exec** method.

[VBScript]

```
Dim WshShell, oExec
Set WshShell = CreateObject("WScript.Shell")

Set oExec = WshShell.Exec("calc")

Do While oExec.Status = 0
     WScript.Sleep 100
```

```
Loop

WScript.Echo oExec.Status
```

[JScript]

```
var WshShell = new ActiveXObject("WScript.Shell");
var oExec = WshShell.Exec("calc");

while (oExec.Status == 0)
{
     WScript.Sleep(100);
}

WScript.Echo(oExec.Status);
```

See Also

WshScriptExec Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Execute Method

Starts execution of a remote script object.

object.**Execute**

Parameters

object
WshRemote Object

Remarks

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).

Example

The following example demonstrates how the **Execute** method is used to create a **WshRemote** object (start an instance of a remote script).

[VBScript]

```
Dim Controller, RemoteScript
Set Controller = WScript.CreateObject("WSHController")
Set RemoteScript = Controller.CreateScript("remote1.js")
RemoteScript.Execute

Do While RemoteScript.Status <> 2
    WScript.Sleep 100
Loop
```

[JScript]

```
var Controller = WScript.CreateObject("WSHController");
var RemoteScript = Controller.CreateScript("remote1.js");
RemoteScript.Execute();

while (RemoteScript.Status != 2) {
    WScript.Sleep(100);
}
```

See Also

WshController Object | WshRemote Object | Status Property | Error Property | Terminate Method | Start Event | End Event

Build: Topic Version 5.6.9309.1546

Windows Script Host

Exists Method

Indicates whether a specific key value exists in the **WshNamed** object.

object.**Exists**(*key*)

Parameters

object
WshNamed object.
Key
String value indicating an argument of the **WshNamed** object.

Remarks

The **Exists** method returns a Boolean value. It returns **true** if the requested argument was specified on the command line (otherwise, it returns **false**).

Example

Consider the following information typed at the command line.

```
myScript.vbs /c:"WSH is a wonderful thing"
```

You could use these two lines of JScript code to find out whether the following command line switches were used to start the script.

```
WScript.Echo(WScript.Arguments.Named.Exists("C"));
WScript.Echo(WScript.Arguments.Named.Exists("D"));
```

See Also

Arguments Property | WshNamed Object | WshUnnamed Object | Count Method | Item Property

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

ExpandEnvironmentStrings Method

Returns an environment variable's expanded value.

object.**ExpandEnvironmentStrings**(*strString*)

Arguments

object
WshShell object.
strString
String value indicating the name of the environment variable you want to expand.

Remarks

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 expands the Windows Directory environment variable and displays it:

[VBScript]

```
set WshShell = WScript.CreateObject("WScript.Shell")
```

```
WScript.Echo "WinDir is " & WshShell.ExpandEnvironmentStrings("%WinDir%")
```

[JScript]

```
var WshShell = WScript.CreateObject("WScript.Shell");
WScript.Echo("WinDir is " + WshShell.ExpandEnvironmentStrings("%WinDir%"));
```

See Also

WshShell Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

GetObject Method

Retrieves an existing object with the specified ProgID, or creates a new one from a file.

object.**GetObject**(*strPathname* [,*strProgID*], [*strPrefix*])

Arguments

object
WScript object.
strPathname
The fully qualified path name of the file that contains the object persisted to disk.
strProgID
Optional. The object's program identifier (ProgID).
strPrefix
Optional. Used when you want to sync the object's events. If you supply the *strPrefix* argument, WSH connects the object's outgoing

interface to the script file after creating the object.

Remarks

Use the **GetObject** method when an instance of the object exists in memory, or when you want to create the object from a file. If no current instance exists and you do not want the object created from a file, use the **CreateObject** method. The **GetObject** method can be used with all COM classes, independent of the language used to create the object. If you supply the *strPrefix* argument, WSH connects the object's outgoing interface to the script file after creating the object. When the object fires an event, WSH calls a subroutine with *strPrefix* attached to the beginning of the event name. For example, if *strPrefix* is `MYOBJ_` and the object fires an event named `OnBegin`, WSH calls the `MYOBJ_OnBegin` subroutine located in the script.

If an object is registered as a single-instance object, only one instance of the object is created (regardless of how many times **GetObject** is executed). The **GetObject** method always returns the same instance when called with the zero-length string syntax (""), and it causes an error if you do not supply the path parameter. You cannot use the **GetObject** method to obtain a reference to a Microsoft Visual Basic class created with Visual Basic 4.0 or earlier.

Example

The following VBScript code starts the application associated with the specified file (*strPathname*):

```
Dim MyObject As Object
Set MyObject = GetObject("C:\CAD\SCHEMA.CAD")
MyApp = MyObject.Application
```

Some applications allow you to activate part of a file. To do this, add an exclamation mark (!) to the end of the file name, and follow it with a string that identifies the part of the file you want to activate. For example, in a drawing application, a drawing stored in a file might have multiple layers. The following code activates a layer within a drawing file called `SCHEMA.CAD`:

```
Set LayerObject = GetObject("C:\CAD\SCHEMA.CAD!Layer3")
```

If you do not specify the object's class (*strProgID*), COM determines the application to start from the file name. Some files can support more than one class of object. For example, a drawing might support three different types of objects: an application object, a drawing object, and a toolbar object. All may be part of the same file.

In the following VBScript code, the drawing application `FIGMENT` starts and opens the object `DRAWING` from within the file `SAMPLE.DRW`.

```
Dim MyObject As Object
```

```
Set MyObject = GetObject("C:\DRAWINGS\SAMPLE.DRW", "FIGMENT.DRAWING")
```

See Also

WScript Object | CreateObject Method | DisconnectObject Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

getResource Method

Returns the value of a resource defined with the **<resource>** element.

getResource(*resourceID*)

Arguments

resourceID
A string that uniquely identifies the resource information contained within a set of resource tags in a *.WSF script file.

Remarks

The **getResource** method returns a string. Use the **<resource>** element to isolate strings or numbers that are within the .wsf file and that you want to reference. This feature makes it easy to maintain a set of strings that are localized into several languages. A WSH script file (*.wsf) can contain several different pieces of resource information — each one with a unique resource identifier.

Example

The following WSH script defines a resource called `errNonNumeric`. The value of `errNonNumeric` is displayed if the parameter `upperBound` is not a number.

```
<package>
<job id="JS">
<resource id="errNonNumeric">Error: A non-numeric value was entered where a number was expected.</resource>
<script language="JScript">

function getRandomNumber(upperBound)
{
   var realUpperBound = parseInt(upperBound);
   if (!isNaN(realUpperBound))
      return (realUpperBound * Math.random) + 1
   else
   {
      WScript.Echo(getResource("errNonNumeric"));
      WScript.Quit(-1);
   }
}

NewValue = getRandomNumber("Bad Value");

</script>
</job>

<job id="VBS">
<resource id="errNonNumeric">Error: A non-numeric value was entered where a number was expected.</resource>
<script language="VBScript">

Function getRandomNumber(upperBound)
   If IsNumeric(upperBound) Then
      getRandomNumber = CInt(upperBound * Rnd + 1)
   Else
      WScript.Echo getResource("errNonNumeric")
      WScript.Quit -1
   End If
End Function

NewValue = getRandomNumber("Bad Value")

</script>
</job>
</package>
```

See Also

<resource> Element

Build: Topic Version 5.6.9309.1546

Windows Script Host

LogEvent Method

Adds an event entry to a log file.

```
object.LogEvent(intType, strMessage [,strTarget])
```

Arguments

object
: **WshShell** object.

intType
: Integer value representing the event type.

strMessage
: String value containing the log entry text.

strTarget
: 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.

Remarks

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

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

Example

The following code logs SUCCESS or ERROR depending on the outcome of the function runLoginScript().

[VBScript]

```
Set WshShell = WScript.CreateObject("WScript.Shell")
rc = runLoginScript()      'Returns true if logon succeeds.

if rc then
   WshShell.LogEvent 0, "Logon Script Completed Successfully"
else
   WshShell.LogEvent 1, "Logon Script failed"
end if
```

[JScript]

```
var WshShell = WScript.CreateObject("WScript.Shell");
var rc = runLoginScript();

if (rc)
   WshShell.LogEvent(0, "Logon Script Completed Successfully");
else
   WshShell.LogEvent(1, "Logon Script failed");
```

See Also

WshShell Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

MapNetworkDrive Method

Adds a shared network drive to your computer system.

```
object.MapNetworkDrive(strLocalName, strRemoteName, [bUpdateProfile], [strUser], [strPassword])
```

Arguments

object
: **WshNetwork** object.

strLocalName
: String value indicating the name by which the mapped drive will be known locally.

strRemoteName
: 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)**.

strUser
: 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.

strPassword
: 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.

Remarks

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

Example

The following code maps the logical drive "E" to a network share with the UNC name "\\Server\Public."

[VBScript]

```
Dim WshNetwork
Set WshNetwork = WScript.CreateObject("WScript.Network")
WshNetwork.MapNetworkDrive "E:", "\\Server\Public"
```

[JScript]

```
var WshNetwork = WScript.CreateObject("WScript.Network");
WshNetwork.MapNetworkDrive ("E:", "\\\\Server\\Public");
```

See Also

WshNetwork Object | EnumNetworkDrives Method | RemoveNetworkDrive Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Popup Method

Displays text in a pop-up message box.

intButton = *object*.**Popup**(*strText*,[*nSecondsToWait*],[*strTitle*],[*nType*])

Arguments

object
WshShell object.

strText
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.
strTitle
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.
IntButton
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.

Remarks

The **Popup** method displays a message box regardless of which host executable file is running (WScript.exe or CScript.exe). If *nSecondsToWaitis* 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 **MessageBox** function. The following tables show the values and their meanings. You can combine values in these tables.

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

Button Types

Value	Description
0	Show **OK** button.
1	Show **OK** and **Cancel** buttons.
2	Show **Abort**, **Retry**, and **Ignore** buttons.
3	Show **Yes**, **No**, and **Cancel** buttons.
4	Show **Yes** and **No** buttons.
5	Show **Retry** and **Cancel** buttons.

Icon Types

Value	Description
16	Show "Stop Mark" icon.
32	Show "Question Mark" icon.
48	Show "Exclamation Mark" icon.
64	Show "Information Mark" icon.

The previous two tables do not cover all values for *nType*. For a complete list, see the Microsoft Win32 documentation.

The return value *intButton* denotes the number of the button that the user clicked. If the user does not click a button before *nSecondsToWait* seconds, *intButton* is set to -1.

Value	Description
1	**OK** button
2	**Cancel** button
3	**Abort** button
4	**Retry** button
5	**Ignore** button
6	**Yes** button
7	**No** button

Example

The following code generates a simple pop-up window.

[VBScript]

```
Dim WshShell, BtnCode
Set WshShell = WScript.CreateObject("WScript.Shell")

BtnCode = WshShell.Popup("Do you feel alright?", 7, "Answer This Question:", 4 + 32)

Select Case BtnCode
   case 6      WScript.Echo "Glad to hear you feel alright."
   case 7      WScript.Echo "Hope you're feeling better soon."
   case -1     WScript.Echo "Is there anybody out there?"
End Select
```

[JScript]

```
var WshShell = WScript.CreateObject("WScript.Shell");
var BtnCode = WshShell.Popup("Do you feel alright?", 7, "Answer This Question:", 4 + 32);
switch(BtnCode) {
   case 6:
      WScript.Echo("Glad to hear you feel alright.");
      break;
   case 7:
      WScript.Echo("Hope you're feeling better soon.");
      break;
   case -1:
      WScript.Echo("Is there anybody out there?");
      break;
}
```

See Also

WshShell Object | Echo Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Quit Method

Forces script execution to stop at any time.

object.**Quit(**[*intErrorCode*])

Arguments

object

WScript object.

intErrorCode

Optional. Integer value returned as the process's exit code. If you do not include the *intErrorCode* parameter, no value is returned.

Remarks

The **Quit** method can return an optional error code. If the **Quit** method is the final instruction in your script (and you have no need to return a non-zero value), you can leave it out, and your script will end normally.

Example

The following JScript code snippet quits execution and returns an error code of 1:

```
WScript.Quit (1);

// This line of code is never executed.
var i = 0;
```

See Also

WScript Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Read Method

Returns a specified number of characters from an input stream.

```
object.Read(characters)
```

Arguments

object
StdIn text stream object.
characters
Integer value indicating the number of characters you want to read.

Remarks

The **Read** method returns a string. The **StdIn**, **StdOut,** and **StdErr** properties and methods work when running the script with the CScript.exe host executable file only. An error is returned when run with WScript.exe. Reading begins at the current position pointer location and moves forward one character at a time.

The Read method does not return until the enter key is pressed. Only the number of characters requested will be returned. Any additional characters will be returned on subsequent calls to the **Read**, **ReadLine**, or **ReadAll** methods.

Example

The following code uses the **Read** method to get a character from the keyboard and display it on the console.

[VBScript]

```
Dim Input
Input = ""

Do While Not WScript.StdIn.AtEndOfLine
   Input = Input & WScript.StdIn.Read(1)
Loop
WScript.Echo Input
```

[JScript]

```
var input = "";
while (!WScript.StdIn.AtEndOfLine)
{
   input += WScript.StdIn.Read(1);
}
WScript.Echo(input);
```

See Also

StdIn Property (WScript)

Build: Topic Version 5.6.9309.1546

Windows Script Host

ReadAll Method

Returns all characters from an input stream.

object.**ReadAll**

Arguments

object
StdIn text stream object.

Remarks

The **ReadAll** method returns a string. The **StdIn**, **StdOut,** and **StdErr** properties and methods work when running the script with the CScript.exe host executable file only. An error is returned when run with WScript.exe.

Example

The following code demonstrates the use of **ReadAl**l.

[VBScript]

```
Dim Input
```

```
Input = ""

Do While Not WScript.StdIn.AtEndOfStream
   Input = Input & WScript.StdIn.ReadAll
Loop
WScript.Echo Input
```

[JScript]

```
var input = "";
while (!WScript.StdIn.AtEndOfStream)
{
   input += WScript.StdIn.ReadAll();
}
WScript.Echo(input);
```

See Also

StdIn Property | FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

ReadLine Method

Returns an entire line from an input stream.

object.`ReadLine`

Arguments

object
StdIn text stream object.

Remarks

The **ReadLine** method returns a string. The **StdIn**, **StdOut,** and **StdErr** properties and methods work when running the script with the CScript.exe host executable file only. An error is returned when run with WScript.exe. A line is a sequence of characters that ends with a newline character.

Note Although this method extracts the newline character, it does not add it to the string.

Example

The following code demonstrates the use of the **ReadLine** method.

[VBScript]

```
Dim StdIn, StdOut
Set StdIn = WScript.StdIn
Set StdOut = WScript.StdOut

Do While Not StdIn.AtEndOfStream
     str = StdIn.ReadLine
     StdOut.WriteLine "Line " & (StdIn.Line - 1) & ": " & str
Loop
```

[JScript]

```
var stdin = WScript.StdIn;
var stdout = WScript.StdOut;

while (!stdin.AtEndOfStream)
{
     var str = stdin.ReadLine();
     stdout.WriteLine("Line " + (stdin.Line - 1) + ": " + str);
}
```

See Also

StdIn Property | FileSystemObject Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

RegDelete Method

Deletes a key or one of its values from the registry.

```
object.RegDelete(strName)
```

Arguments

object
WshShell object.
strName
String value indicating the name of the registry key or key value you want to delete.

Remarks

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. The five possible root keys you can use are listed in the following table.

Root key Name	Abbreviation
HKEY_CURRENT_USER	HKCU
HKEY_LOCAL_MACHINE	HKLM
HKEY_CLASSES_ROOT	HKCR
HKEY_USERS	HKEY_USERS

HKEY_CURRENT_CONFIG HKEY_CURRENT_CONFIG

Example

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

[VBScript]

```
Dim WshShell, bKey
Set WshShell = WScript.CreateObject("WScript.Shell")

WshShell.RegWrite "HKCU\Software\ACME\FortuneTeller\", 1, "REG_BINARY"
WshShell.RegWrite "HKCU\Software\ACME\FortuneTeller\MindReader", "Goocher!", "REG_SZ"

bKey = WshShell.RegRead("HKCU\Software\ACME\FortuneTeller\")
WScript.Echo WshShell.RegRead("HKCU\Software\ACME\FortuneTeller\MindReader")

WshShell.RegDelete "HKCU\Software\ACME\FortuneTeller\MindReader"
WshShell.RegDelete "HKCU\Software\ACME\FortuneTeller\"
WshShell.RegDelete "HKCU\Software\ACME\"
```

[JScript]

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

WshShell.RegWrite ("HKCU\\Software\\ACME\\FortuneTeller\\", 1, "REG_BINARY");
WshShell.RegWrite ("HKCU\\Software\\ACME\\FortuneTeller\\MindReader", "Goocher!", "REG_SZ");

var bKey =    WshShell.RegRead ("HKCU\\Software\\ACME\\FortuneTeller\\");
WScript.Echo (WshShell.RegRead ("HKCU\\Software\\ACME\\FortuneTeller\\MindReader"));

WshShell.RegDelete ("HKCU\\Software\\ACME\\FortuneTeller\\MindReader");
WshShell.RegDelete ("HKCU\\Software\\ACME\\FortuneTeller\\");
WshShell.RegDelete ("HKCU\\Software\\ACME\\");
```

See Also

WshShell Object | RegRead Method | RegWrite Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

RegRead Method

Returns the value of a key or value-name from the registry.

object.**RegRead**(*strName*)

Arguments

object
WshShell object.
strName
String value indicating the key or value-name whose value you want.

Remarks

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 (e.g., "%windir%\\calc.exe")	A string
REG_MULTI_SZ	An array of strings	A VBArray of strings

You can specify a key-name by ending *strName* 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. Fully qualified key-names and value-names begin with a root key. You may use abbreviated versions of root key names with the **RegRead** method. The five possible root keys are listed in the following table.

Root key Name	Abbreviation
HKEY_CURRENT_USER	HKCU
HKEY_LOCAL_MACHINE	HKLM
HKEY_CLASSES_ROOT	HKCR
HKEY_USERS	HKEY_USERS
HKEY_CURRENT_CONFIG	HKEY_CURRENT_CONFIG

Example

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

[VBScript]

```
Dim WshShell, bKey
Set WshShell = WScript.CreateObject("WScript.Shell")

WshShell.RegWrite "HKCU\Software\ACME\FortuneTeller\", 1, "REG_BINARY"
WshShell.RegWrite "HKCU\Software\ACME\FortuneTeller\MindReader", "Goocher!", "REG_SZ"

bKey = WshShell.RegRead("HKCU\Software\ACME\FortuneTeller\")
WScript.Echo WshShell.RegRead("HKCU\Software\ACME\FortuneTeller\MindReader")

WshShell.RegDelete "HKCU\Software\ACME\FortuneTeller\MindReader"
WshShell.RegDelete "HKCU\Software\ACME\FortuneTeller\"
WshShell.RegDelete "HKCU\Software\ACME\"
```

[JScript]

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

WshShell.RegWrite ("HKCU\\Software\\ACME\\FortuneTeller\\", 1, "REG_BINARY");
WshShell.RegWrite ("HKCU\\Software\\ACME\\FortuneTeller\\MindReader", "Goocher!", "REG_SZ");

var bKey =    WshShell.RegRead ("HKCU\\Software\\ACME\\FortuneTeller\\");
WScript.Echo (WshShell.RegRead ("HKCU\\Software\\ACME\\FortuneTeller\\MindReader"));
```

```
WshShell.RegDelete ("HKCU\\Software\\ACME\\FortuneTeller\\MindReader");
WshShell.RegDelete ("HKCU\\Software\\ACME\\FortuneTeller\\");
WshShell.RegDelete ("HKCU\\Software\\ACME\\");
```

See Also

WshShell Object | RegDelete Method | RegWrite Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

RegWrite Method

Creates a new key, adds another value-name to an existing key (and assigns it a value), or changes the value of an existing value-name.

object.**RegWrite(***strName, anyValue* [,*strType*])

Arguments

object
WshShell object.
strName
String value indicating the key-name, value-name, or value you want to create, add, or change.
anyValue
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.
strType
Optional. String value indicating the value's data type.

Remarks

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

Converted to	***strType***
String	REG_SZ
String	REG_EXPAND_SZ
Integer	REG_DWORD
Integer	REG_BINARY

Note The REG_MULTI_SZ type is not supported for the **RegWrite** method.

Tip **RegWrite** will write at most one DWORD to a REG_BINARY value. Larger values are not supported with this method.

Fully qualified key-names and value-names are prefixed with a root key. You may use abbreviated versions of root key names with the **RegWrite** method. The five root keys are listed in the following table.

Root key Name	**Abbreviation**
HKEY_CURRENT_USER	HKCU
HKEY_LOCAL_MACHINE	HKLM
HKEY_CLASSES_ROOT	HKCR
HKEY_USERS	HKEY_USERS
HKEY_CURRENT_CONFIG	HKEY_CURRENT_CONFIG

The four possible data types you can specify with *strType* are listed in the following table.

Type	**Description**	**In the Form of**
REG_SZ	A string	A string
REG_DWORD	A number	An integer
REG_BINARY	A binary value	An integer
REG_EXPAND_SZ	An expandable string (e.g., "%windir%\\calc.exe")	A string

Example

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

[VBScript]

```
Dim WshShell, bKey
Set WshShell = WScript.CreateObject("WScript.Shell")

WshShell.RegWrite "HKCU\Software\ACME\FortuneTeller\", 1, "REG_BINARY"
WshShell.RegWrite "HKCU\Software\ACME\FortuneTeller\MindReader", "Goocher!", "REG_SZ"

bKey = WshShell.RegRead("HKCU\Software\ACME\FortuneTeller\")
WScript.Echo WshShell.RegRead("HKCU\Software\ACME\FortuneTeller\MindReader")

WshShell.RegDelete "HKCU\Software\ACME\FortuneTeller\MindReader"
WshShell.RegDelete "HKCU\Software\ACME\FortuneTeller\"
WshShell.RegDelete "HKCU\Software\ACME\"
```

[JScript]

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

WshShell.RegWrite ("HKCU\\Software\\ACME\\FortuneTeller\\", 1, "REG_BINARY");
WshShell.RegWrite ("HKCU\\Software\\ACME\\FortuneTeller\\MindReader", "Goocher!", "REG_SZ");

var bKey =    WshShell.RegRead ("HKCU\\Software\\ACME\\FortuneTeller\\");
WScript.Echo (WshShell.RegRead ("HKCU\\Software\\ACME\\FortuneTeller\\MindReader"));

WshShell.RegDelete ("HKCU\\Software\\ACME\\FortuneTeller\\MindReader");
WshShell.RegDelete ("HKCU\\Software\\ACME\\FortuneTeller\\");
WshShell.RegDelete ("HKCU\\Software\\ACME\\");
```

See Also

WshShell Object | RegDelete Method | RegRead Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Remove Method

Removes an existing environment variable.

object.**Remove**(*strName*)

Arguments

object
WshEnvironment object.
strName
String value indicating the name of the environment variable you want to remove.

Remarks

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`.

[VBScript]

```
Dim WshShell, WshEnv
Set WshShell = WScript.CreateObject("WScript.Shell")
Set WshEnv = WshShell.Environment("PROCESS")
WshEnv("TestVar") = "Windows Script Host"
WScript.Echo WshShell.ExpandEnvironmentStrings("The value of the test variable is: '%TestVar%'")
```

```
WshEnv.Remove "TestVar"
WScript.Echo WshShell.ExpandEnvironmentStrings("The value of the test variable is: '%TestVar%'")
```

[JScript]

```
var WshShell = WScript.CreateObject("WScript.Shell");
var WshEnv = WshShell.Environment("PROCESS");
WshEnv("TestVar") = "Windows Script Host";
WScript.Echo(WshShell.ExpandEnvironmentStrings("The value of the test variable is: '%TestVar%'"));
WshEnv.Remove("TestVar");
WScript.Echo(WshShell.ExpandEnvironmentStrings("The value of the test variable is: '%TestVar%'"));
```

See Also

WshEnvironment Object | Environment Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

RemoveNetworkDrive Method

Removes a shared network drive from your computer system.

object.**RemoveNetworkDrive**(*strName*, [*bForce*], [*bUpdateProfile*])

Arguments

object
WshNetwork object.
strName
String value indicating the name of the mapped drive you want to remove. The *strName* 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.

Remarks

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

Example

The following code removes the logical drive "E."

[VBScript]

```
Dim WshNetwork
Set WshNetwork = WScript.CreateObject("WScript.Network")
WshNetwork.RemoveNetworkDrive "E:"
```

[JScript]

```
var WshNetwork = WScript.CreateObject("WScript.Network");
WshNetwork.RemoveNetworkDrive ("E:");
```

See Also

WshNetwork Object | EnumNetworkDrives Method | MapNetworkDrive Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

RemovePrinterConnection Method

Removes a shared network printer connection from your computer system.

```
object.RemovePrinterConnection(strName, [bForce], [bUpdateProfile])
```

Arguments

object
WshNetwork object.
strName
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.

Remarks

The **RemovePrinterConnection** method removes both Windows and MS-DOS based printer connections. If the printer was connected using the method **AddPrinterConnection**, *strName* 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 *strName* must be the printer's UNC name.

Example

The following code disconnects a network printer.

[VBScript]

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
PrinterPath = "\\printserv\DefaultPrinter"
WshNetwork.RemovePrinterConnection PrinterPath, true, true
```

[JScript]

```
var WshNetwork = WScript.CreateObject("WScript.Network");
var PrinterPath = "\\\\PRN-CORP1\\B41-4523-A";
WshNetwork.RemovePrinterConnection(PrinterPath, true, true);
```

See Also

WshNetwork Object | AddPrinterConnection Method | AddWindowsPrinterConnection Method | EnumPrinterConnections Method | SetDefaultPrinter Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Run Method

Runs a program in a new process.

object.**Run**(*strCommand*, [*intWindowStyle*], [*bWaitOnReturn*])

Arguments

object
WshShell object.
strCommand
String value indicating the command line you want to run. You must include any parameters you want to pass to the executable file.

intWindowStyle
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).

Remarks

The **Run** method returns an integer. The **Run** method starts a program running in a new Windows process. 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 *strCommand* 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 *intWindowStyle*.

intWindowStyle	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 1

The following VBScript code opens a copy of the currently running script with Notepad.

```
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Run "%windir%\notepad " & WScript.ScriptFullName
```

The following VBScript code does the same thing, except it 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.

```
Set WshShell = WScript.CreateObject("WScript.Shell")
Return = WshShell.Run("notepad " & WScript.ScriptFullName, 1, true)
```

Example 2

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

```
Dim oShell
Set oShell = WScript.CreateObject ("WSCript.shell")
oShell.run "cmd /K CD C:\ & Dir"
Set oShell = Nothing
```

See Also

WshShell Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Save Method

Saves a shortcut object to disk.

object.**Save**

Arguments

object
WshShortcut or **WshUrlShortcut** object.

Remarks

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. The **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).

Example

The following example demonstrates the use of a single .wsf file for two jobs in different script languages (VBScript and JScript). Each job creates a shortcut to the script being run and a URLshortcut to www.microsoft.com.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         strDesktop = WshShell.SpecialFolders("Desktop")
         set oShellLink = WshShell.CreateShortcut(strDesktop & "\Shortcut Script.lnk")
         oShellLink.TargetPath = WScript.ScriptFullName
         oShellLink.WindowStyle = 1
         oShellLink.Hotkey = "CTRL+SHIFT+F"
         oShellLink.IconLocation = "notepad.exe, 0"
         oShellLink.Description = "Shortcut Script"
         oShellLink.WorkingDirectory = strDesktop
         oShellLink.Save
         set oUrlLink = WshShell.CreateShortcut(strDesktop & "\Microsoft Web Site.url")
         oUrlLink.TargetPath = "http://www.microsoft.com"
         oUrlLink.Save
      </script>
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         strDesktop = WshShell.SpecialFolders("Desktop");
         var oShellLink = WshShell.CreateShortcut(strDesktop + "\\Shortcut Script.lnk");
```

```
        oShellLink.TargetPath = WScript.ScriptFullName;
        oShellLink.WindowStyle = 1;
        oShellLink.Hotkey = "CTRL+SHIFT+F";
        oShellLink.IconLocation = "notepad.exe, 0";
        oShellLink.Description = "Shortcut Script";
        oShellLink.WorkingDirectory = strDesktop;
        oShellLink.Save();
        var oUrlLink = WshShell.CreateShortcut(strDesktop + "\\Microsoft Web Site.url");
        oUrlLink.TargetPath = "http://www.microsoft.com";
        oUrlLink.Save();
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | WshShortcut Object | WshUrlShortcut Object | FullName Property | TargetPath Property | WindowStyle Property | Hotkey Property | IconLocation Property | Description Property | WorkingDirectory Property

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

SendKeys Method

Sends one or more keystrokes to the active window (as if typed on the keyboard).

object.**SendKeys**(*string*)

Arguments

object
WshShell object.

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

Remarks

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. For example, to send the letter x, send the *string* argument "x".

Note 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). This set of special characters consists of parentheses, brackets, braces, and the:

- plus sign "+",
- caret "^",
- percent sign "%",
- and 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
BACKSPACE	{BACKSPACE}, {BS}, or {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL or DELETE	{DELETE} or {DEL}

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

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	%

Note 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.

Note 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".

Note You cannot send the PRINT SCREEN key {PRTSC} to an application.

Example

The following example demonstrates the use of a single .wsf file for two jobs in different script languages (VBScript and JScript). Each job runs the Windows calculator and sends it keystrokes to execute a simple calculation.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         WshShell.Run "calc"
```

```
        WScript.Sleep 100
        WshShell.AppActivate "Calculator"
        WScript.Sleep 100
        WshShell.SendKeys "1{+}"
        WScript.Sleep 500
        WshShell.SendKeys "2"
        WScript.Sleep 500
        WshShell.SendKeys "~"
        WScript.Sleep 500
        WshShell.SendKeys "*3"
        WScript.Sleep 500
        WshShell.SendKeys "~"
        WScript.Sleep 2500
     </script>
  </job>

  <job id="js">
     <script language="JScript">
        var WshShell = WScript.CreateObject("WScript.Shell");
        WshShell.Run("calc");
        WScript.Sleep(100);
        WshShell.AppActivate("Calculator");
        WScript.Sleep(100);
        WshShell.SendKeys ("1{+}");
        WScript.Sleep(500);
        WshShell.SendKeys("2");
        WScript.Sleep(500);
        WshShell.SendKeys("~");
        WScript.Sleep(500);
        WshShell.SendKeys("*3");
        WScript.Sleep(500);
        WshShell.SendKeys("~");
        WScript.Sleep(2500);
     </script>
  </job>
</package>
```

See Also

WshShell Object | Run Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

SetDefaultPrinter Method

Assigns a remote printer the role Default Printer.

object.**SetDefaultPrinter**(*strPrinterName*)

Arguments

object
WshNetwork object.
strPrinterName
String value indicating the remote printer's UNC name.

Remarks

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.

Example

The following code uses the **AddWindowsPrinterConnection** method to connect a network printer and set it as the default printer.

[VBScript]

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
PrinterPath = "\\research\library1"
WshNetwork.AddWindowsPrinterConnection PrinterPath
WshNetwork.SetDefaultPrinter PrinterPath
```

[JScript]

```
var WshNetwork = WScript.CreateObject("WScript.Network");
var PrinterPath = "\\\\research\\library1";
WshNetwork.AddWindowsPrinterConnection(PrinterPath);
WshNetwork.SetDefaultPrinter(PrinterPath);
```

See Also

WshNetwork Object | AddPrinterConnection Method | AddWindowsPrinterConnection Method | EnumPrinterConnections Method | RemovePrinterConnection Method

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

ShowUsage Method

Makes a script self-documenting by displaying information about how it should be used.

object.**ShowUsage**

Parameters

object
WScript Object.

Remarks

When you run the **ShowUsage** method, a help screen (referred to as the usage) appears and displays details about the script's command line options. This information comes from the runtime section of the *.WSF file. Everything written between the **<runtime>** and **</runtime>** tags is pieced together to produce what is called a "usage statement." The usage statement tells the user how to use the script.

Note The usage can also be displayed using the /? switch.

Example

The following example demonstrates how to set up usage information in a *.WSF script file.

```
<job>
    <runtime>
        <description>This script reboots a server</description>
        <named
            name = "Server"
            helpstring = "Server to run the script on"
            type = "string"
            required = "true"
        />
        <example>Example: reboot.wsf /server:scripting</example>
    </runtime>
<script language="VBScript">

If WScript.Arguments.Count <> 1 Then
   WScript.Arguments.ShowUsage
   WScript.Quit
End If

</script>
</job>
```

The JScript code for the equivalent script block would be:

```
if (WScript.Arguments.length != 1)
{
   WScript.Arguments.ShowUsage();
   WScript.Quit();
}
```

Calling the **ShowUsage** method from this script results in the following output:

```
This script reboots a server
Usage: reboot.wsf /server:value

Options:
```

```
server : Server to run the script onExample:
reboot.wsf /server:scripting
```

See Also

WScript Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Sign Method

Signs a script stored in a string.

***Object*.Sign (FileExtension, Text, Certificate, Store)**

Arguments

object
Scripting.Signer
FileExtension
A string designating the script extension type (.vbs, .js, or .wsf). This provides a mechanism by which the operating system can determine the type of script file being verified.
Text
A string containing the script to be signed.
Certificate
A string designating the author's certificate name.
Store
Optional. A string designating the name of the certificate store. Typically certificates that contain private keys — i.e., certificates you

can use for code signing — are in a certificate store called "my". The default value is "my".

Remarks

The Sign method is used to digitally sign a script stored in a string. In order to create a digital signature, the caller must have a valid certificate.

Example

```
Dim Signer, UnsignedText, SignedText
Set Signer = CreateObject("Scripting.Signer")
UnsignedText = _
     "Dim X " & vbCrLf & _
     "X = 123" & vbCrLf & _
     "WScript.Echo X" & vbCrLf
SignedText = Signer.Sign(".VBS", UnsignedText, "Your Certificate Name Here")
```

See Also

Scripting.Signer Object | SignFile Method | Verify Method | VerifyFileMethod | Signing a Script

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

SignFile Method

Signs a script using a digital signature.

***Object*.SignFile (FileName, Certificate, Store)**

Arguments

object
Scripting.Signer
FileName
A string containing the name of the script file.
Certificate
A string designating the author's certificate name.
Store
An optional string designating the name of the certificate store. Typically certificates that contain private keys — i.e., certificates you can use for code signing — are in a certificate store called "my". The default value is "my".

Remarks

In order to sign a digital signature, the author must have a valid certificate.

Example

The following example demonstrates not only signature checking but also the command-line argument.

```
<job>
<runtime>
<named name="file" helpstring="the file to sign" required="true" type="string"/>
<named name="cert" helpstring="the name of the signing certificate" required="true" type="string"/>
<named name="store" helpstring="the name of the certificate store" required="false" type="string"/>
</runtime>
<script language="vbscript">
Dim Signer, File, Cert, Store
If Not (WScript.Arguments.Named.Exists("cert") And WScript.Arguments.Named.Exists("file")) Then
     WScript.Arguments.ShowUsage
     WScript.Quit
End If
Set Signer = CreateObject("Scripting.Signer")
File  = WScript.Arguments.Named("file")
Cert  = WScript.Arguments.Named("cert")
If WScript.Arguments.Named.Exists("store") Then
     Store = WScript.Arguments.Named("store")
Else
     Store = "my"
End If
Signer.SignFile File, Cert, Store
</script>
```

```
</job>
```

See Also

Scripting.Signer Object | Sign Method | Verify Method | VerifyFile Method | Signing a Script

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Skip Method

Skips a specified number of characters when reading from an input text stream.

object.**Skip**(*characters*)

Arguments

object
StdIn text stream object.
characters
Integer value indicating the number of characters you want to skip.

Remarks

The **StdIn**, **StdOut,** and **StdErr** properties and methods work when running the script with the CScript.exe host executable file only. An "Invalid Handle" error is returned when run with WScript.exe. The position pointer moves forward by the number of characters (bytes) specified in the argument *characters*. You cannot use the **Skip** method to skip backwards through a file (negative character values are not supported). The **Skip** method is limited to the open for reading mode only (you cannot skip a specified number of characters when writing to an output stream).

Example

The following code uses the **Skip** method to jump over the first character in a text stream, read a line from the keyboard, and write it to the **StdOut** text stream.

[VBScript]

```
WScript.StdIn.Skip 1
Input = WScript.StdIn.ReadLine
WScript.StdOut.Write Input
```

[JScript]

```
WScript.StdIn.Skip(1);
Input = WScript.StdIn.ReadLine();
WScript.StdOut.Write(Input);
```

See Also

StdIn Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

SkipLine Method

Skips the next line when reading from an input text stream.

object.**SkipLine**

Arguments

object
StdIn text stream object.

Remarks

A line is a sequence of characters that ends with a newline character. The **StdIn**, **StdOut,** and **StdErr** properties and methods work when running the script with the CScript.exe host executable file only. An "Invalid Handle" error is returned when run with WScript.exe. The position pointer moves forward to the point just past the next newline character. You cannot use the **SkipLine** method to skip backwards through a file. The **SkipLine** method is limited to the open for reading mode only (you cannot skip lines when writing to an output stream).

Example

The following code demonstrates the **SkipLine** method.

[VBScript]

```
Dim StdIn, StdOut, Str1, Str2

Set StdIn = WScript.StdIn
Set StdOut = WScript.StdOut

Str1 = ""
Str2 = ""

For i = 0 to 4
   StdIn.SkipLine
Next

i = 0
Do While Not StdIn.AtEndOfStream
     If i >= 2 Then
          StdOut.WriteLine Str1
     End If
     i = i + 1
     Str1 = Str2
     Str2 = StdIn.ReadLine
Loop
```

[JScript]

```
var stdin = WScript.StdIn;
var stdout = WScript.StdOut;
var str1, str2 = "";
var i;
for (i = 0; i < 5; i++)
     stdin.SkipLine();
i = 0;
while (!stdin.AtEndOfStream)
{
     if (i++ >= 2)
     {
          stdout.WriteLine(str1);
     }
     str1 = str2;
     str2 = stdin.ReadLine();
}
```

See Also

StdIn Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

Sleep Method

Suspends script execution for a specified length of time, then continues execution.

object.**Sleep**(*intTime*)

Arguments

object
WScript object.
intTime
Integer value indicating the interval (in milliseconds) you want the script process to be inactive.

Remarks

The thread running the script is suspended, releasing its CPU utilization. Execution resumes as soon as the interval expires. Using the **Sleep** method can be useful when you are running asynchronous operations, multiple processes, or if your script includes code triggered by an event. To be triggered by an event, a script must be continually active (a script that has finished executing will certainly not detect an event). Events handled by the script will still be executed during a sleep.

Note Passing the **Sleep** method a 0 or –1 does *not* cause the script to suspend indefinitely.

Example

The following example demonstrates the use of a single .wsf file for two jobs in different script languages (VBScript and JScript). The functionality of both jobs is the same — each runs the Windows calculator and sends it keystrokes to execute a simple calculation.

```
<package>
   <job id="vbs">
      <script language="VBScript">
         set WshShell = WScript.CreateObject("WScript.Shell")
         WshShell.Run "calc"
         WScript.Sleep 100
         WshShell.AppActivate "Calculator"
         WScript.Sleep 100
         WshShell.SendKeys "1{+}"
         WScript.Sleep 500
         WshShell.SendKeys "2"
         WScript.Sleep 500
         WshShell.SendKeys "~"
         WScript.Sleep 500
         WshShell.SendKeys "*3"
         WScript.Sleep 500
         WshShell.SendKeys "~"
         WScript.Sleep 2500
      </script>
```

```
   </job>

   <job id="js">
      <script language="JScript">
         var WshShell = WScript.CreateObject("WScript.Shell");
         WshShell.Run("calc");
         WScript.Sleep(100);
         WshShell.AppActivate("Calculator");
         WScript.Sleep(100);
         WshShell.SendKeys("1{+}");
         WScript.Sleep(500);
         WshShell.SendKeys("2");
         WScript.Sleep(500);
         WshShell.SendKeys("~");
         WScript.Sleep(500);
         WshShell.SendKeys("*3");
         WScript.Sleep(500);
         WshShell.SendKeys("~");
         WScript.Sleep(2500);
      </script>
   </job>
</package>
```

See Also

Running Your Scripts | WScript Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Terminate Method (WshScriptExec)

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

object.**Terminate**

Arguments

object
WshScriptExec object.

Remarks

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.

Example

The following JScript example demonstrates how to use the **Terminate** method to stop a running script.

```
var aScript = WScript.Exec("%comspec% /c myScript.js");
aScript.Terminate();
```

See Also

WshScriptExec Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Verify Method

Verifies a digital signature retrieved as a string.

object.**Verify (FileExtension, Text, ShowUI)**

Arguments

object
Scripting.Signer
FileExtension
A string designating the script extension type (.vbs, .js, or .wsf). This provides a mechanism by which the operating system can determine the type of script file being verified.
Text
The text to verify.
ShowUI
A Boolean value. If the **ShowUI** argument is false, then the Scripting.Signer object determines whether a trusted source provided the signature without prompting the user. If it is true then the Scripting.Signer object may create dialog boxes to prompt the user if there is not sufficient information to determine trust.

> **Note** On some operating systems, the operating system also creates a dialog box if the flag is on, the file is trusted, and you have not already checked the "Always trust …" option.

Remarks

The Verify method is used to verify a digital signed script stored in a string.

Example

In this example the Verify method determines whether the script in the UnsignedText variable is trusted. (In this example there is no digital signature, so the Scripting.Signer object asks the user whether to extend trust.)

```
Dim Signer, UnsignedText, Trusted
Set Signer = CreateObject("Scripting.Signer")
UnsignedText = _
     "Dim X " & vbCrLf & _
     "X = 123" & vbCrLf & _
     "WScript.Echo X" & vbCrLf
Trusted = Signer.Verify(".VBS", UnsignedText, True)
```

See Also

Scripting.Signer Object | VerifyFile Method | Sign Method | SignFile Method | Verifying a Script

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

VerifyFile Method

Verifies the digital signature encapsulated in a script.

```
Object.VerifyFile (FileName, ShowUI)
```

Arguments

object
Scripting.Signer
FileName
A string containing the name of the script file.
ShowUI
A Boolean value. If the **ShowUI** argument is false, then the Scripting.Signer object determines whether a trusted source provided the signature without prompting the user. If it is true then the Scripting.Signer object may create dialog boxes to prompt the user if there is not sufficient information to determine trust.

> **Note** On some operating systems, the operating system also creates a dialog box if the flag is on, the file is trusted, and you have not already checked the "Always trust …" option.

Example

The following example demonstrates signature checking using the command-line argument processing features.

```
<job>
```

```
<runtime>
<named name="file" helpstring="the file to sign" required="true" type="string"/>
<named name="UI" helpstring="produce user interface for untrusted scripts" required="false"/>
</runtime>
<script language="vbscript">
Dim Signer, File, UI, OK
If Not WScript.Arguments.Named.Exists("file") Then
     WScript.Arguments.ShowUsage
     WScript.Quit
End If
Set Signer = CreateObject("Scripting.Signer")
File = WScript.Arguments.Named("file")
UI   = WScript.Arguments.Named.Exists("ui")
OK = Signer.VerifyFile(File, UI)
If OK Then
     WScript.Echo File & " is trusted."
Else
     WScript.Echo File & " is NOT trusted."
End If
</script>
</job>
```

See Also

Scripting.Signer Object | Verify Method | Sign Method | SignFile Method | Verifying a Script

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Write Method

Sends a string to an output stream.

```
object.Write(strText)
```

Arguments

object
StdOut or **StdErr** text stream objects.
strText
String value indicating the text you want to write to the stream.

Remarks

The **StdIn**, **StdOut,** and **StdErr** properties and methods work when running the script with the CScript.exe host executable file only. An "Invalid Handle" error is returned when run with WScript.exe. The position pointer moves to the point just beyond the last character in *strText.*

Example

The following code demonstrates the use of the **Write** method.

[VBScript]

```
Dim strInput
strInput = WScript.StdIn.ReadAll
WScript.StdOut.Write strInput
```

[JScript]

```
var strInput = WScript.StdIn.ReadAll();
WScript.StdOut.Write(strInput);
```

See Also

StdErr Property | StdOut Property

Build: Topic Version 5.6.9309.1546

Windows Script Host

WriteBlankLines Method

Sends a specified number of blank lines (newline characters) to an output stream.

object.**WriteBlankLines**(*intLines*)

Arguments

object
StdOut or **StdErr** text stream objects.
intLines
Integer value indicating the number of blank lines you want to write to the stream.

Remarks

Calling the **WriteLine** method without supplying the *strText* argument is equivalent to calling **WriteBlankLines(1)**. The **StdIn**, **StdOut,** and **StdErr** properties and methods work when running the script with the CScript.exe host executable file only. An "Invalid Handle" error is returned when run with WScript.exe.

Example

The following code demonstrates the **WriteBlankLines** method.

[VBScript]

```
Dim StdIn, StdOut
Set StdIn = WScript.StdIn
Set StdOut = WScript.StdOut

Do While Not StdIn.AtEndOfStream
```

```
     str = StdIn.ReadLine
     StdOut.Write "Line " & (StdIn.Line - 1) & ": " & str
     StdOut.WriteBlankLines 1
Loop
```

[JScript]

```
var stdin = WScript.StdIn;
var stdout = WScript.StdOut;

while (!stdin.AtEndOfStream)
{
     var str = stdin.ReadLine();
     stdout.Write("Line " + (stdin.Line - 1) + ": " + str);
     stdout.WriteBlankLines(1);
}
```

See Also

StdErr Property | StdOut Property | WriteLine Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

WriteLine Method

Sends a string with a newline character to an output stream.

object.**WriteLine**([*strText*])

Arguments

object
StdOut or **StdErr** text stream objects.
strText
Optional. String value indicating the text you want to write to the stream. If omitted, a newline character is written to the output stream.

Remarks

The **WriteLine** method always appends a newline character to the string. Calling the **WriteLine** method without supplying the argument *strText* is equivalent to calling **WriteBlankLines(1)**. The **StdIn**, **StdOut,** and **StdErr** properties and methods work when running the script with the CScript.exe host executable file only. An "Invalid Handle" error is returned when run with WScript.exe. A line is a sequence of characters that ends with a newline character.

Example

The following code demonstrates the **WriteLine** method.

[VBScript]

```
Dim StdIn, StdOut
Set StdIn = WScript.StdIn
Set StdOut = WScript.StdOut

Do While Not StdIn.AtEndOfStream
     str = StdIn.ReadLine
     StdOut.WriteLine "Line " & (StdIn.Line - 1) & ": " & str
Loop
```

[JScript]

```
var stdin = WScript.StdIn;
var stdout = WScript.StdOut;

while (!stdin.AtEndOfStream)
{
     var str = stdin.ReadLine();
     stdout.WriteLine("Line " + (stdin.Line - 1) + ": " + str);
}
```

See Also

StdErr Property | StdOut Property | WriteBlankLines Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Events

In this Section

End Event
: Signals the script engine to end execution of a remote script object.

Error Event
: Signals the script engine to end execution of a remote script object when the remote script terminates prematurely due to an error.

Start Event
: Signals the script engine to begin execution of a remote script object.

Related Sections

WSH Language
: List of elements that make up WSH Reference.

WSH Basics
: Learn the basics of WSH.

Build: Topic Version 5.6.9309.1546

Windows Script Host

End Event

Event that is fired when the remote script completes.

***Object*_end**

Parameters

object
WshRemote object.

Remarks

The **End** event is fired when the remote script object has finished executing. This can be when the remote script object has terminated normally, timed out, or terminated due to an error.

Example

```
var WshController = new ActiveXObject("WSHController");
var RemoteProc = WshController.CreateScript("foo.wsf", "remotemachine");
WScript.ConnectObject(RemoteProc, "RemoteProc_");
var Done = false;
RemoteProc.Execute();
while (!Done)
   WScript.Sleep(100);

function RemoteProc_End()
{
   WScript.Echo("The process has ended");
   Done = true;
}

function RemoteProc_Error()
{
   WScript.Echo("An error has occurred: " + RemoteProc.Error.Description);
   Done = true;
}
```

```
function RemoteProc_Start()
{
   WScript.Echo("The process has started");
}
```

See Also

WshController Object | WshRemote Object | Status Property | Error Property | Execute Method | Terminate Method | Start Event | Error Event

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Error Event

Event that is fired when an error occurs in the remote script.

*Object*_**Error**

Parameters

object
WshRemote object.

Remarks

The remote script object fires the **Error** event when the remote script terminates prematurely. The **Error** property contains the **WshRemoteError** object (which holds information about the error that caused the remote script to terminate prematurely).

Example

```
var WshController = new ActiveXObject("WSHController");
var RemoteProc = WshController.CreateScript("foo.wsf", "remotemachine");
WScript.ConnectObject(RemoteProc, "RemoteProc_");
var Done = false;
RemoteProc.Execute();
while (!Done)
   WScript.Sleep(100);

function RemoteProc_End()
{
   WScript.Echo("The process has ended");
   Done = true;
}

function RemoteProc_Error()
{
   WScript.Echo("An error has occurred: " + RemoteProc.Error.Description);
   Done = true;
}

function RemoteProc_Start()
{
   WScript.Echo("The process has started");
}
```

See Also

WshController Object | WshRemote Object | WshRemoteError Object | Status Property | Error Property | Execute Method | Terminate Method | Start Event | End Event

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Start Event

Event that is fired when the remote script begins executing.

*Object*_**Start**

Parameters

object
WshRemote object.

Remarks

The **Start** event is fired when the **Execute** method is called.

Example

```
var WshController = new ActiveXObject("WSHController");
var RemoteProc = WshController.CreateScript("foo.wsf", "remotemachine");
WScript.ConnectObject(RemoteProc, "RemoteProc_");
var Done = false;
RemoteProc.Execute();
while (!Done)
   WScript.Sleep(100);

function RemoteProc_End()
{
   WScript.Echo("The process has ended");
   Done = true;
}

function RemoteProc_Error()
{
   WScript.Echo("An error has occurred: " + RemoteProc.Error.Description);
   Done = true;
}

function RemoteProc_Start()
{
```

```
   WScript.Echo("The process has started");
}
```

See Also

WshController Object | WshRemote Object | Status Property | Error Property | Execute Method | Terminate Method | End Event | Error Event

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Error Messages

In this Section

Here is a list of the errors messages you may encounter when running scripts in the Windows Script Host 5.6 environment.

//E option requires name of script engine.

//H option requires host name.

//T option requires timeout value.

A duplicate name for a named or unnamed element was encountered.

An attempt at saving your settings via the //S option failed.

Can't change default script host <host name>.

Can't find script engine <engine name> for script.

Can't find script file <script file name>.

Can't read script from stdin.

Can't save settings.

Command line option mismatch.

Could not connect object <object name>.

Could not locate automation class name <automation class name>.

Execution of the Windows Script Host failed.

Host name for //H option must be "cscript" or "wscript".

Initialization of the Windows Script Host failed.

Invalid attempt to call Exec without a command.

Invalid pathname.

Invalid root in registry key <name> for reading.

Invalid syntax in URL<name>.

Invalid timeout value for //T option.

Loading script <script name> failed.

Loading your settings failed.

Missing job name.

Protocol handler for <name> not found.

Remote script object can only be executed once.

Script execution time was exceeded on script <script name>. <script name> execution was terminated.

Script setting file <settings filename>is invalid.

The shortcut pathname must end with .lnk or .url.

There is no file extension in <file name>.

There is no printer called <name>.

There is no script engine for file extension <file extension>.

There is no script file specified.

Unable to execute - arguments list too long.

Unable to open registry key <name> for reading.

Unable to remove environment variable <name>.

Unable to remove registry key <name>.

Unable to save shortcut <name>.

Unable to set shortcut target to <name>.

Unable to write to wsh.log. Please check with your administrator.

Unable to wait for process.

Unable to execute remote script.

Unable to find job <job identifier>.

Unicode is not supported on this platform.

Unknown option <option designation> specified.

Windows Script Host access is disabled on this machine. Contact your administrator for details.

Related Sections

WSH Reference
List of elements that make up WSH Reference.
WSH Basics
Learn the basics of WSH.

Build: Topic Version 5.6.9309.1546

Windows Script Host

//H option requires host name.

You failed to specify the default host name.

To correct this error

- Specify the default host name when using the //H option.

See Also

Host name for //H option must be "cscript" or "wscript".

Build: Topic Version 5.6.9309.1546

Windows Script Host

//T option requires timeout value.

You failed to input a timeout value for the //T option.

To correct this error

- Input a timeout value using the //T option.

See Also

Invalid timeout value for //T option | Script execution time was exceeded on script <script name>. <script name> execution was terminated.

Build: Topic Version 5.6.9309.1546

Windows Script Host

A duplicate name for a named or unnamed element was encountered.

You are attempting to name a <named> or <unnamed> element with a name that is already in use by another <named> or <unnamed>

element. The "name" attribute of the <named> and <unnamed> elements must be unique throughout the <job>.

To correct this error

- Choose a different name for the <named> or <unnamed> element.

See Also

WshArguments Object | WshNamed Object | WshUnnamed Object | <named> Element

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

An attempt at saving your settings via the //S option failed.

Again this is usually a systems permissions issue.

To correct this error

- Consult your Administrator about potential network or security problems.

See Also

Can't change default script host <host name> | Setting Script Properties

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unable to execute - arguments list too long.

This error generally occurs when using the drag and drop feature of Windows Script Host. This happens when too many files are dropped on a Windows Script File.

To correct this error

- Shorten the argument list by dragging and dropping fewer items.

  Note The maximum command-line length that your system allows determines the number of files you can drag onto a script.

See Also

- Drag and Drop Support | Command line option mismatch

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unable to write to wsh.log. Please check with your administrator.

On Windows 95/98 and Windows Millennium Editions, you are calling the method **LogEvent()**, but it fails because the wsh.log file is locked.

To correct this error

- Unlock the wsh.log file if you the proper permissions or consult your system administrator.

See Also

LogEvent Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Can't change default script host <host name>.

This is usually a systems permission issue and the likely cause is that the system registry has been locked by an Administrator.

To correct this error

- Consult your Administrator about potential network or security problems.

See Also

An attempt at saving your settings via the //s option failed

Build: Topic Version 5.6.9309.1546

Windows Script Host

Can't find script engine <engine name> for script.

Usually, this means that the script engine is not installed.

To correct this error

- Consult your Administrator about potential network or security problems.
- Make sure that the particular script engine is installed or that you are specifying the correct script engine.
- Check your spelling.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Can't find script file <script file name>.

You have given an incorrect path or the script file is not there.

To correct this error

- Check the path given and correct it.
- Make sure that the file is actually there

See Also

Invalid pathname.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Can't read script from stdin.

This is often a permissions issue.

To correct this error

- Consult your Administrator about potentially serious network or security problems.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Can't save settings.

Unable to save the control file (*.wsh) for this script.

To correct this error

- Verify that you are not trying to save over a read-only file, a file that is open in another application, or a file locked by an administrator.
- Consult your Administrator about potential network or security problems.

See Also

Setting Script Properties

Build: Topic Version 5.6.9309.1546

Windows Script Host

Command line option mismatch.

Often this means you have specified conflicting arguments. For example, //B - batch mode is inconsistent with //I - interactive mode.

To correct this error

- Check your arguments for conflicts.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Could not connect object <object name>.

It is likely that you are trying to sync events to an object or are sourcing events to an object that cannot accept those object events.

For example, IE can't be connected to events once it is created. You have to connect any object events to IE at the time it is created as the object as a part of the creation process.

To correct this error

- Create the object and connect its object events to it during the creation process.

See Also

<object> Element | WScript Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Could not locate automation class name <automation class name>.

Either you were unsuccessful in creating an automation class or the automation class was not properly installed.

To correct this error

- Make sure the automation class that you created is installed.
- Consult your Administrator about potential network or security problems.

Build: Topic Version 5.6.9309.1546

Windows Script Host

//E option requires name of script engine.

You have not designated the script engine using the //E option.

To correct this error

- Be sure to designate the script engine when using the //E option.

See Also

There is no script engine for file extension <file extension>.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unable to remove environment variable <name>.

You are calling the method **Environment.Remove()** on an unrecognized environment variable.

This error generally occurs when using the **Remove** method. If the environment variable does not exist or is misspelled, the system is not able to remove it.

To correct this error

- Check the names of the current environment variables and make sure they exist and are spelled correctly.

See Also

Remove Method | WshEnvironment Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Execution of the Windows Script Host failed.

Usually, this is a systems permissions issue.

To correct this error

- Consult your Administrator about potentially serious network or security problems.

See Also

Loading script <script name> failed

Build: Topic Version 5.6.9309.1546

Windows Script Host

Host name for //H option must be "cscript" or "wscript".

You specified something other than "cscript" or "wscript" as your host name when you used the //H option.

To correct this error

- Specify either "cscript" or "wscript" as the host name when using the //H option.

See Also

H option requires host name.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Initialization of the Windows Script Host failed.

Windows Script Host failed to initialize.

To correct this error

- Consult your Administrator about potentially serious network or security problems.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Invalid attempt to call Exec without a command.

You called the **WshShell.Exec()** method but did not supply a command to execute.

To correct this error

- *strCommand* is a required argument. It is a string that represents the command line used to run the script.

See Also

Exec Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Invalid pathname.

You have given an incorrect path.

To correct this error

- Check the path given and correct it.

See Also

Can't find script file <script file name>.

Build: Topic Version 5.6.9309.1546

Windows Script Host

The shortcut pathname must end with .lnk or .url

When you named your shortcut, either you did not give it a file extension or you gave it an extension other than **.lnk*, nor **.url*.

When using the **CreateShortcut** method, the file extension is incorrect or missing.

To correct this error

- Check that the file extension for the shortcut ends with *.lnk* for a Windows shortcut or with *.url* for an Internet shortcut.

See Also

WshShortcut Object | WshUrlShortcut Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Invalid timeout value for //T option.

The timeout value that you entered using the //T option is invalid. You must designate a non-negative integer for //T option.

To correct this error

- Input an appropriate timeout value using the //T option. Specify a non-negative integer for //T.

See Also

//T option requires timeout value | Script execution time was exceeded on script <script name>. <script name> execution was terminated.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Loading script <script name> failed.

Usually, this is a systems permissions issue and you have been denied read access to the file in question.

To correct this error

- Consult your Administrator about potentially serious network or security problems.

See Also

Loading your settings failed

Build: Topic Version 5.6.9309.1546

Windows Script Host

Loading your settings failed.

Usually, this is a systems permissions issue.

To correct this error

- Consult your Administrator about potentially serious network or security problems.

See Also

Loading script <script name> failed

Build: Topic Version 5.6.9309.1546

Windows Script Host

Missing job name.

You have failed to specify the job name.

To correct this error

- Specify the job name.

See Also

Unable to find job <job identifier>

Build: Topic Version 5.6.9309.1546

Windows Script Host

There is no printer called <name>.

The **SetDefaultPrinter** method accepts only the names of currently installed printers. This error generally occurs if you specify the printer port instead of the name of the printer, or if you use the name of a printer that is not currently installed.

To correct this error

- Check the names of the currently installed printers and make sure the specified printer is installed.
- Check the spelling of the printer path for typing errors.
- Make sure the networked printer is online.

See Also

SetDefaultPrinter Method | EnumPrinterConnections Method | AddWindowsPrinterConnection Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Protocol handler for <name> not found.

This error generally occurs when the protocol handler for the URL shortcut target is misspelled. The two most common protocol handlers are HTTP and FTP.

To correct this error

- Check the spelling of the protocol handler.

See Also

WshUrlShortcut Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Invalid root in registry key <name> for reading.

You are calling the methods **RegRead()** or **RegWrite()** on an unrecognized root registry key.

This error generally occurs when using the **RegRead** or **RegWrite** methods with an invalid registry key.

To correct this error

- Check the Windows registry and make sure the registry key exists and is spelled correctly.

See Also

RegRead Method | RegWrite Method | RegDelete Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unable to open registry key <name> for reading.

You are calling the method **RegRead()** on an unrecognized registry key.

This error generally occurs when using the **RegRead** method with an invalid registry key.

To correct this error

- Check the Windows registry and make sure the registry key exists and is spelled correctly.

See Also

RegRead Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unable to remove registry key <name>.

You are calling the method **RegDelete()** on an unrecognized registry key.

This error generally occurs when using the **RegDelete** method on an invalid registry key.

To correct this error

- Check the Windows registry and make sure the registry key exists and is spelled correctly.

See Also

RegDelete Method | RegRead Method | RegWrite Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Remote script object can only be executed once.

You are attempting to rerun a remote script object.

To correct this error

- Create another remote script object and run it instead.

See Also

WshRemote Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Script execution time was exceeded on script <script name>. <script name> execution was terminated.

The script in question could not be executed within the user-defined execution time (//T).

To correct this error

- Troubleshoot your script to determine why it is not executing within the time parameter (//T) that you set. Redesign your script so that it executes faster and/or change the time parameter (//T).

See Also

T option requires timeout value | Invalid timeout value for T option

---

Build: Topic Version 5.6.9309.1546

Windows Script Host

Script setting file <settings filename>is invalid.

You used an invalid script setting file.

To correct this error

- Recreate your script setting file or consult your systems administrator.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Invalid syntax in URL<name>.

This error generally occurs when the text that makes up the URL target contains incorrect syntax.

To correct this error

- Check the spelling of the URL target for typing errors.

See Also

WshUrlShortcut Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unable to save shortcut <name>.

This error generally occurs when the name of the shortcut already exists or is a read-only file.

To correct this error

- Change the read-only attribute to read/write.
- Rename your shortcut to something unique.
- Change the name of the existing shortcut.

See Also

WshShortcut Object | WshUrlShortcut Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unable to set shortcut target to <name>.

This error generally occurs when the protocol handler for the URL shortcut target is invalid or improperly registered. The two most common protocol handlers are HTTP and FTP.

To correct this error

- Use a valid protocol handler.
- Check the spelling of the protocol handler being used.
- Contact your system administrator to make sure the protocol handler is properly installed.
- Consult your Administrator about potential network or security problems.

See Also

WshUrlShortcut Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

There is no file extension in <file name>.

In order to determine what script engine is apt, the program usually needs the appropriate file extension.

To correct this error

- Be sure include a correct file extension that is appropriate to the scripting language you wish to use, along with the file name.

Build: Topic Version 5.6.9309.1546

Windows Script Host

There is no script engine for file extension <file extension>.

This file extension is not mapped to a script engine.

To correct this error

- Check your file extension and also the spelling of your scripting language designation.
- Use //E to designate a script engine.

See Also

E option requires name of script engine.

Build: Topic Version 5.6.9309.1546

Windows Script Host

There is no script file specified.

You have failed to specify a script file.

To correct this error

- Specify the appropriate script file.

See Also

Unknown option <option designation> specified

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unable to execute remote script.

The script failed to initialize.

To correct this error

- Consult your Administrator about potential network or security problems.

See Also

WshRemote Object

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unable to find job <job identifier>.

You made a reference to an unrecognized job ID in your Windows Script file (*.wsf file).

To correct this error

- Check the spelling of the job ID for typing errors.

See Also

Using Windows Script Files | <job> Element

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unable to wait for process.

This error generally occurs when the script is hung up waiting for a process that does not return a value, for example, waiting for the result from running a shortcut link.

To correct this error

- Execute the program directly instead of using a shortcut.
- Check the shortcut link to be sure it is still current.

See Also

Run Method

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unicode is not supported on this platform.

The WIN 95/98 and Windows Millennium Edition platforms do not support the use of Unicode.

To correct this error

- Don't attempt to use Unicode on WIN95/98 or Windows Millennium Edition platforms.

Build: Topic Version 5.6.9309.1546

Windows Script Host

Unknown option <option designation> specified.

You have specified an option that is invalid for Windows script host.

To correct this error

- Use the Windows script host option appropriate to the script language you are using or specify another supported script language.
- Also check your spelling.

See Also

There is no script file specified

Build: Topic Version 5.6.9309.1546

Windows Script Host

Windows Script Host access is disabled on this machine. Contact your administrator for details.

Usually, this is a systems permissions issue.

To correct this error

- Consult your Administrator about potential network or security problems.

See Also

Execution of the Windows Script Host failed

Build: Topic Version 5.6.9309.1546