Managing Enterprise Systems with the Windows Script Host

STEIN BORGE

Apress™

Managing Enterprise Systems with the Windows Script Host

ISBN (pbk): 1-893115-67-4
Printed and bound in the United States of America 12345678910
Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Technical Reviewer: Eric Lippert
Editorial Directors: Dan Appleman, Gary Cornell, Jason Gilmore, Karen Watterson
Project Managers: Alexa Stuart, Erin Mulligan
Developmental Editor: Kenyon Brown
Copy Editor: Nicole LeClerc
Production Editor: Kari Brooks
Compositor: Impressions Book and Journal Services, Inc.
Indexer: Valerie Robbins
Cover Designer: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc.,175 Fifth Avenue, New York, NY, 10010
and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany
In the United States, phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com.
Outside the United States, fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 901 Grayson Street, Suite 204, Berkeley, CA 94710.
Phone 510-549-5938, fax: 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section. You will need to answer questions pertaining to this book in order to successfully download the code.

CHAPTER 6

Input/Output Streams

Even with the graphical user interface, command-line console applications are still important, especially when mass repetitive changes are required that would take a great amount of time to perform through the graphical user interface or when scripts are designed to perform operations that do not require user interaction.

Windows Script Host (WSH) 1.0 allowed the processing of command-line parameters. WSH version 2.0 introduces the ability to process standard input and output streams (`StdIn` and `StdOut`). This new feature is of great importance for the creation of flexible console applications.

The command-line environment has always supported the capability of "piping" streams from one console application to another, but it has never provided a great number of built-in commands to use this ability, apart from the `MORE` and `SORT` commands.

This capability has always been an important feature in the UNIX environment, and most native UNIX shell commands allow (or require) input to be provided via standard input, allowing complex sequences of operations to be executed within a single command line.

> **NOTE** *You can download* `StdIn`, `StdOut`, *and* `StdErr` *property documentation and WSH documentation from* `http://msdn.microsoft.com/scripting`.

6.1 Using Regular Expressions to Filter the Contents of an Input Stream

Problem

You require a routine that filters supplied information and then outputs any results that meet the criteria to the standard output (`StdOut`). The resulting output can be used by other console applications.

Solution

The following script reads input from standard input using the `WScript.StdIn` object, and then it filters the line against a regular expression. Only lines that match the expression are written to standard output:

```
<?xml version="1.0" ?>
<job>
<!--comment
Script:wshgrep.wsf
performs regular expression filtering against standard input
-->
 <script language="VBScript" src="fsolib.vbs">
<![CDATA[

 Option Explicit

 Dim nF, objFSO, strLine
 Dim objRegExp, strFilter
 On Error Resume Next

If Not IsCscript Then ExitScript _
    "This script must be run from command line using cscript.exe", True


 If WScript.Arguments.Count <> 1 Then
  ShowUsage
  WScript.Quit
End If
  strFilter= WScript.Arguments(0)
  Set objRegExp = New RegExp
  objRegExp.Pattern = strFilter   objRegExp.IgnoreCase = True


  Do While Not WScript.StdIn.AtEndOfStream
    strLine = WScript.StdIn.ReadLine

    If objRegExp.Test(strLine) Then
WScript.StdOut.WriteLine(strLine)
   End If
 Loop
```

```
Sub ShowUsage()
 WScript.Echo _
     WScript.ScriptName & " filters standard input against a regular
expression." _
    & vbCrLf & "Syntax:" &  vbCrLf & _
    WScript.ScriptName & "  regexp" &  vbCrLf & _
    "regexp  regular expression"
 End Sub
  ]]>
  </script>
</job>
```

Discussion

WSH version 2.0 provides access to the standard input and outputs (`StdIn` and `StdOut`) from the Windows console. This allows scripts to "pipe" information between console applications.

Piping allows for information to be passed from one console application to another. Using the vertical bar (|) pipes information between one or more console applications. For example:

```
dir /b | sort | more
```

To access the `StdIn` or `StdOut` stream, use the `StdIn` and `StdOut` property of the `WScript` object. These properties return `TextStream` objects that can be read and written to as if they were text files. For example, the following WSH script reads a line from the standard input and writes its uppercase equivalent to the standard output by using the `StdIn` and `StdOut` properties:

```
'ucasein.vbs
'converts standard Input stream to uppercase
'and redirects to stdout
Dim strText
strText = WScript.StdIn.ReadAll
WScript.StdOut.Write Ucase(strText)
```

To use the `ucasein` script, pipe output to it from other applications. The following command-line snippet pipes the file `users.txt` to the script:

```
 cscript ucasein.vbs < users.txt > ucusers.txt
```

The contents of the input stream (in this case, `users.txt`) is converted to uppercase and written to standard output. The standard output in this case is redirected to a new file: `ucusers.txt`. Information can be piped to a WSH script from any existing console application or other scripts that write information to the `StdOut` stream.

If you execute a console script that reads `StdIn` but does not have information piped from another application or from a file, the application will take input from the user's standard input device, the keyboard. In this case, the script will pause to accept keyboard input. The following command line starts the `ucasein` script and accepts input from the keyboard because no other source is redirected/piped to the script:

```
cscript ucasein.vbs
```

Press Ctrl-Z to end the processing of keyboard input from `StdIn`. This keystroke combination sends an end of file (EOF) sequence to the stream. Pressing Ctrl-Break when reading keyboard input from `StdIn` will force an EOF.

To pipe information to a WSH script, execute a console application and pipe the output to the WSH script. The following script pipes the contents of a `dir` command to the `ucasein` script:

```
dir | cscript ucasein.vbs | more
```

The output of the `dir` command is converted to uppercase. This result is then piped to the `more` command, which displays one screen of text at a time.

When you chain commands together on non–Windows 2000/XP machines, the script must be prefixed by `Cscript` or `Wscript`. The following command line is the equivalent of the previous sample:

```
dir | ucasein.vbs | more
```

This generates an error when run on Windows NT 4.0/9*x*/ME computers. On Windows 2000/XP, it will use the default script host, either `Cscript` or `Wscript`. If a script writes to `StdOut`, you should use `Cscript` because if the result of the output is not piped to another process, an error will occur. The following example will not work:

```
dir | wscript ucasein.vbs
```

The preceding example doesn't work because the results are not piped to another process and the `Wscript` script host does use the results. Replacing `Cscript` with `Wscript` in the example would result in the output being displayed

in the console. The earlier example in which the results were piped to the more command would work using Wscript.

Even though you can use Wscript to execute scripts that use StdIn, you should avoid using it to write to StdOut.

The Solution script evaluates each line on the StdIn against a regular expression. Any resulting matches are written to StdOut. For example, say you want to output the routing tables to a file without any of the additional headings:

```
route print | cscript //NoLogo wshgrep.wsf (\d+\.\d+\.\d+\.\d.+){4} > rt.txt
```

The Route Print command pipes the routing information to the wshgrep.wsf script. WshGrep filters out all lines that meet the criteria and outputs them to StdOut. This output is redirected to the file rt.txt.

The //Nologo switch ensures that no "logo" information from the execution of the script appears with the output. This includes the Microsoft WSH version and copyright information.

If you want to prevent the display of the Microsoft logo and copyright information by default, use the //S switch to save the command-line settings as the default:

```
wscript //NoLogo //S
```

This saves the //NoLogo switch as a default switch.

The wshgrep.wsf script and other scripts in this section include an fsolib.vbs script library to implement repetitive functions. The fsolib.vbs script library is shown here:

```
'fsolib.vbs
'Description: Contains routines used by FSO scripts

'check if script is being run interactively
'Returns:True if run from command line, otherwise false
Function IsCscript()
  IsCScript = (StrComp(Right(WScript.Fullname,11),"cscript.exe",1) = 0)
End Function

'display an error message and exist script
'Parameters:
'strMsg         Message to display
'strUseWscript Use Wscript.Echo to display message.
'   By default StdErr is used, but this cannot be used in
'   interactive (wscript) mode unless redirected to somewhere else.
Sub ExitScript(strMsg, bUseWscript)
```

```
 If bUseWscript Then
  WScript.Echo strMsg
 Else
  'get the standard error stream
   WScript.StdErr.WriteLine strMsg
 End If
 WScript.Quit -1
End Sub

'returns contents of specified file. If file doesn't exist
'terminates script and displays error message
'Parameters:
'strFile Path to file to return
'Returns
'contents of specified file
Function GetFile(strFile)
    On Error Resume Next
    Dim objFSO, objFile
    Set objFSO = CreateObject("Scripting.FileSystemObject")
    Set objFile = objFSO.OpenTextFile(strFile)
    If Err Then ExitScript _
        "Error " & Err.Description & " opening file " & _
        strFile, False
    GetFile = objFile.ReadAll
    objFile.Close
End Function

'terminates script with message if script not run using cscript.ext
'Parameters:None
Sub CheckCScript()
    If Not IsCscript Then ExitScript _
        "This script must be run from command line using cscript.exe", True
End Sub

'checks if specified number of arguments have been passed and exits script
'displaying usage information if not
'Parameters:
'nCount  Number of arguments expected
Sub CheckArguments(nCount)
    If WScript.Arguments.Count <> nCount Then
        WScript.Arguments.ShowUsage
        WScript.Quit
    End If
End Sub
```

See Also

Solution 3.1 and Solution 8.1.

6.2 Reading Keyboard Input

Problem

You want to create a simple text-based menu.

Solution

You can read a character from standard input using the `Read` method:

```
<?xml version="1.0" ?>
<job>
<!--comment
Script:menu.wsf
demonstrate a simple text-based menu
-->
 <script language="VBScript" src="fsolib.vbs">
 <![CDATA[
'menu.wsf
Dim strOption

CheckCScript

WScript.Echo "-------Menu Options-------------"
WScript.Echo "1 - Copy Information"
WScript.Echo "2 - Move Information"
WScript.Echo "3 - Quit"
WScript.Echo "Select option and press the Enter key to continue"

'read the standard input
strOption = WScript.StdIn.Read(1)

Select Case strOption
 Case "1"
    WScript.Echo "option 1 selected"
Case "2"
    WScript.Echo "option 2 selected"
```

```
Case "3"
    WScript.Quit -1
Case Else
    WScript.Echo "Invalid option selected"
End Select

WScript.StdIn.Close
 ]]>
  </script>
</job>
```

Discussion

Even though Windows provides an advanced graphical user interface, it can still be useful to provide text-based menus for console applications. `StdIn` provides a method of reading input from the console.

If no stream is redirected to `StdIn`, the keyboard is used to read `StdIn`. `StdIn` returns a `TextStream` object and supports the methods provided through this object to read input (`Read`, `ReadLine`, and `ReadAll` methods).

Using the `Read` method, you can specify the number of characters you want to read. The method does not terminate once the number of characters specified has been entered; you must press the Enter key or the EOF key combination (Ctrl-Z). Only the number of characters specified by the `Read` method is actually returned.

See Also

Solution 3.2.

6.3 Generating Template-Based Data

Problem

You want to be able to search and replace values from standard input.

Solution

You can read the standard input stream using `WScript.StdIn` and then use the results to populate templates that are provided through a command-line parameter or external file:

```
<?xml version="1.0" ?>
<job>
<runtime>
    <description>
<![CDATA[
This script demonstrates use of WScript.StdIn/Out/Err by
doing some template processing.  A comma-separated list
of replacement strings is read in from stdin, merged into
a template file and the result is dumped out to stdout.
The process is repeated for each line of replacement strings.
]]>
    </description>
    <unnamed name="TemplateFile" many="false" required="true"
    helpstring="File containing template text." />
    <example>
<![CDATA[
CScript sar.wsf Template.txt < Replacements.txt > Out.txt

Suppose Replacements.txt contained

Bob,*.doc
Sue,*.txt

and Template.txt contained

net use \\odin\</1/> /user:admin /password:bigsecret
copy \\odin\</1/>\backmeup\</2/> \\loki\backups\</1/>\
net use /d \\odin\</1/>

then Out.txt would contain

net use \\odin\bob /user:admin /password:bigsecret
copy \\odin\bob\backmeup\*.doc \\loki\backups\bob\
net use /d \\odin\bob
net use \\odin\sue /user:admin /password:bigsecret
copy \\odin\sue\backmeup\*.txt \\loki\backups\sue\
net use /d \\odin\sue
]]>
    </example>
</runtime>
<script language="VBScript" src="fsolib.vbs">
<![CDATA[
   Dim strTemplate
```

```
    Sub ReplaceText
     Dim strRepls, aRepls, strOut, objRegExp
     Set objRegExp = New RegExp
     objRegExp.Pattern = "<\/\d+\/>"

      'loop through each line of standard input
     Do While Not WScript.StdIn.AtEndOfStream
      strRepls = WScript.StdIn.ReadLine
      aRepls = Split(strRepls, ",")
      strOut = strTemplate
       'replace each element in template
       For nF = 0 To Ubound(aRepls)
          strOut = Replace(strOut , "</" & nF+1 & "/>" , aRepls(nF))
       Next
      'check if all elements were replaced
      If objRegExp.Test(strOut) Then _
         ExitScript "Replacement file has too few values.", False
       WScript.StdOut.Write strOut
      Loop
  End Sub

  CheckCScript
  CheckArguments 1
  strTemplate = GetFile(WScript.Arguments(0))
  ReplaceText
  ]]>
 </script>
</job>
```

Discussion

The search and replace script creates output in which tags in a template string are replaced by elements from standard input.

Each line of the standard input must consist of data elements delimited by a comma. These elements are identified by their ordinal position in the line, so the first element is 1, the second element is 2, and so on.

The following `users.txt` text file contains information that can be piped to the script. In this example, there are three elements for each line:

```
Freds,Fred Smith,Accounting Manager
Joeb,Joe Blow,Computer Operator
```

The template string can either be a text file or a command-line parameter. In the template, any instance of an element number surrounded by **</** and **/>** is replaced with the corresponding element from standard input.

In the following example, </1/> is replaced by the first element from standard input, </2/> with the second, and so on:

```
net user password </1/> /ADD /FULLNAME:"</2/>" /COMMENT:"</3/>"
```

Using first line of `users.txt` as input, the following output is generated:

```
net user password Freds /ADD /FULLNAME:"Fred Smith" /COMMENT:"Accounting Manager"
net user password Joeb /ADD /FULLNAME:"Joe Blow" /COMMENT:"Computer Operator"
```

To run the `users.txt` file against a layout string and redirect the output to a batch file called `newusers.bat`, use the following:

```
cscript sar.wsf template.txt  < users.txt  > newusers.bat
```

The `sar` script processes each line of the standard input for data. You can use this ability to use the search and replace script to fill in a template with data as a very flexible tool for creating formatted output. For example, suppose that you want to take the list of users from a text file and generate an HTML file containing the user list in a table. The following layout file, `details.txt`, contains the template table details for each user:

```
<tr><td></1/></td><td></2/></td><td></3/></td></tr>
```

With this template, you can generate the HTML table details using the `users.txt` file:

```
cscript sar.wsf details.txt < users.txt
```

However, to create a complete HTML document, you need to include the appropriate HTML <html>, <body>, and <table> elements to surround the detail lines. You can't use the `sar.wsf` script to insert the details into the body because it processes line by line and would generate an unusable HTML document. You require results of the table generation to be inserted into the body of an HTML document.

To do this, create a modified version of the `sar.wsf` script called `sarw.wsf` to treat the standard input as one element to be replaced in a template:

```
<?xml version="1.0" ?>
<job>
```

```
<runtime>
    <description>
<![CDATA[
This script demonstrates use of WScript.StdIn/Out/Err by
doing some template processing.  The whole StdIn is read and
merged into a template file and the result is dumped out to stdout.
]]>
    </description>
    <unnamed name="TemplateFile" many="false" required="true"
    helpstring="File containing template text." />
    <example>
<![CDATA[
CScript sarw.wsf Template.txt < Replacement.txt > Out.txt

Suppose Replacements.txt contained
Fred Smith  555-1234
Joe Blow    555-2432

and Template.txt contained
     Phone List
Name         Phone
</1/>

then Out.txt would contain:
     Phone List
Name         Phone
Fred Smith  555-1234
Joe Blow    555-2432
]]>
    </example>
</runtime>
<script language="VBScript" src="fsolib.vbs">
<![CDATA[
   Dim strTemplate
   Sub ReplaceText
    Dim strRepls, strOut

    'check if replacement element exists
     If Instr(strTemplate,"</1/>") = 0 Then _
        ExitScript "Template file missing replacement element ", False

     'read the body from standard input and replace template layout
     strRepls = WScript.StdIn.ReadAll
```

```
      strOut = Replace(strTemplate , "</1/>" , strRepls)

     WScript.StdOut.Write strOut
  End Sub

  CheckCScript
  CheckArguments 1
  strTemplate = GetFile(WScript.Arguments(0))
  ReplaceText
  ]]>
 </script>
</job>
```

The `sarw.wsf` script replaces the element `</1/>` in a template file with the `StdIn` contents and writes the results to `StdOut`. The following template file, `body.txt`, is used to generate the body of the HTML file:

```
<html>
<head></head>
<body>
<table border="1" width="100%">
  </1/>
</table>
</body>
</html>
```

The following command sequence generates the HTML file `usrs.htm` using the `details.txt` and `body.txt` templates:

```
cscript sar.wsf details.txt < users.txt | cscript sarw.wsf body.txt > usrs.htm
```

The resulting output is similar to this:

```
<html>
<head></head>
<body>
<table border="1" width="100%">
  <tr><td>Freds</td><td>Fred Smith</td><td> Accounting Manager</td></tr>
  <tr><td>Joeb</td><td>Joe Blow</td><td>Computer Operator</td></tr>
</table>
</body>
</html>
```

The first step redirects the `users.txt` file to `sar.wsf`, which generates the HTML table details. The result of this operation is piped to `sarw.wsf`, which inserts it into the `body.txt` template. The result of this operation is redirected to the `usrs.htm` file.

6.4 Creating Multiple-User Prompts

Problem

Existing data files usually provide the standard input that scripts read. This is useful when processing multiple items, but it can be a bit impractical for single pieces of information. You want to be able to query the user with one or more predefined prompts and then take the results and send them to the standard output.

Solution

You can use the `StdErr` output stream to prompt users for information, which is then piped to standard output for further processing. Using `StdErr` instead of `StdOut` to output information ensures that the user prompts do not get piped with the user input results.

```
<?xml version="1.0" ?>
<job>
<runtime>
    <description>
<![CDATA[
This script demonstrates use of WScript.StdIn/Out/Err by
prompting the user with a set of prompts read from a file
and then dumping the results of those prompts as a comma-
separated list to stdout.
]]>
    </description>
    <unnamed name="PromptFile" many="false" required="true"
    helpstring="File containing prompts." />
    <example>
<![CDATA[
CScript prompt.wsf Prompts.txt

Suppose Prompts.txt contained

What is the user's name?
```

```
What files should be backed up?  (eg, *.doc)

Then this program would ask the user for the values and
output

Bob,*.txt
]]>
    </example>
</runtime>
<script language="VBScript" src="fsolib.vbs">
<![CDATA[

Dim strPromptFile, strPrompts

Sub AskUser
    Dim aPrompts, strPrompt, fComma
    aPrompts = Split(strPrompts, vbCrLf)
        fComma = False
    For Each strPrompt In aPrompts
        ' The file may contain blank lines.
        If Trim(strPrompt) <> "" Then
            If fComma Then WScript.StdOut.Write ","
            WScript.StdErr.Write strPrompt
            WScript.StdOut.Write WScript.StdIn.ReadLine
             fComma = True
        End If
    Next
End Sub

 CheckCScript
 CheckArguments 1
 strPrompts = GetFile(WScript.Arguments(0))
 AskUser
]]>
  </script>
</job>
```

Discussion

The `prompt.wsf` script queries the user for input with prompts that are defined by a template file. This allows the script to prompt the user for information that is

piped or redirected to another process, and it provides an alternative to building data files to redirect to scripts.

You use the script to create a solution that builds a batch file to create a new NT user by prompting for user details. The following `nusr.txt` file contains the prompts to create a new user:

```
Enter user id:
Enter user full name
Enter comment:
```

Each prompt appears on its own line in the file. You now need a template file to fill in the user details. Use the `sar.wsf` script from Solution 6.3 to insert the prompts into a template. The following text file contains the layout for the `nuser.txt` template:

```
rem nuser.txt
Rem create user
net user </1/> /ADD

Rem create a user directoryMd d:\users\</1/>

rem Create the share
net share </1/>$=d:\users\</1/>

rem Grant </1/> and Domain Admins full access to the share
rem shrperm is part of Backoffice resource kit
shrperm \\Odin\</1/>$ </1/>:F "Domain Admins":F

rem Grant user </1/> full access to his or her directory
cacls d:\users\</1/> /T /E /G </1/>:F

rem Remove Everyone access from directory
cacls d:\users\</1/> /T /E /R Everyone

remPermit Domain Admins to have full access in directory.
cacls d:\users\</1/> /T /E /P "Domain admins":F

rem set the home directory setting for user </1/>

net user </1/> /HOMEDIR:\\Odin\</1/>$
net user </1/> /FULLNAME:"</2/>"
net user </1/> /COMMENT:"</3/>"
```

The following command line uses `prompt.wsf` to prompt for a user ID, description, and comment:

```
cscript prompt.wsf inp.txt | cscript sar.wsf nuser.txt > nuser.bat
```

Next, this information is piped to the `sar.wsf` script, which builds the `nuser.bat` batch file using the `nuser.txt` template file.

The `prompt.wsf` script generates user prompts from a file. These prompts are displayed using the `StdErr` stream. The standard error (`StdErr`) output stream is used to display the prompts. Functionally, `StdErr` appears similar to `StdOut`. It returns a `TextStream` object and any output written to it appears on the console.

The difference is that anything written to the `StdErr` stream is not available to be read by the `StdIn` stream. The purpose of the `StdErr` stream is to display error messages in console scripts that perform `StdIn`/`StdOut` operations. This behavior is used by `prompt.wsf` to display the prompts. If `StdOut` or `WScript.Echo` had been used, the prompts would be piped with the results of user prompts.

See Also

Solution 3.8 and Solution 3.9.