Argue Over what your Script Should Do

Use arguments with your scripts to dictate variables when a script is called

by Bob Kelly
May 2007

It is obvious that establishing variables for certain things just makes good sense. If there is a path to a file or directory, that is a perfect candidate for a variable so the code will be easy to change in the future or reuse in another script. Sometimes you may even set a variable to easily change the behavior of your script, such as if your script should display information or not, if it should take actions or just display what it would do for testing purposes, and I'm sure you can think of many other such "yes or no" type options you may want to control when you run the script. If you want to change these values frequently or if you will be sharing this script for others to execute, constantly editing these values in the code may not be desirable. This is where arguments become a valuable feature.

By handling arguments, your script can function much as a command line tool. Whatever options your script provides may be handled by these parameters as you dictate in your code. The Windows Script host supports arguments as property values where you may check how many arguments have been passed to your script and what text each contains. A space is the delimiter which separates parameters and not the common dash or slash characters used to identify command line parameters (you can require or simply ignore such identifiers if you wish). How you want to handle arguments will depend very much on your script and what it does. If you make use of arguments today you know the value, but if you have not worked with them before, below is a skeleton which is aimed to show how you may easily take advantage of this functionality.

```
If WScript.arguments.count = 0 Then
        Exit Do
End If

For i = 0 To (WScript.arguments.count - 1)
sCurrentArgument = WScript.Arguments.Item(i)
Select Case (UCase(sCurrentArgument)
        Case "/A", "-A"
                optionA = True
        Case "/B", "-B"
                optionB = True
        Case "/C", "-C"
                optionC = True
        Case Else
                WScript.Echo "Invalid Argument Passed: " & sCurrentArgument & ".
Aborting."
                WScript.Quit
        End Select
```

```
Next

If optionA Then WScript.Echo "Do actions for option A." End If
If optionB Then WScript.Echo "Do actions for option B." End If
If optionC Then WScript.Echo "Do actions for option C." End If
```

The above script will let you know if an A, B or C switch is passed upon execution. Using this same code and logic, you should be able to easily implement command line arguments in your scripts. Try it out!

Bob Kelly is president and co-founder of AdminScriptEditor.com, home to an integrated suite of scripting tools and a shared library of scripts and language help. He has authored books on scripting and desktop administration as well as several white papers. Bob also owns and operates AppDeploy.com, where he writes and produces videos on topics related to software deployment. You can contact Bob about "Argue Over what your Script Should Do" at bkelly@adminscripteditor.com.