A Test Your PowerShell Prowess

I'm often asked whether or not Microsoft has, or ever will have, a PowerShell certification exam. The answer is "no, and probably not." Microsoft certification exams don't typically focus on a technology or tool; they focus on completing job tasks. You'll find PowerShell on many certification exams, but having a standalone exam isn't something the company is likely to pursue.

I think I understand why people would want such a thing, though. After spending a long time learning to use PowerShell, you want some external verification that you have, in fact, learned it! But certification exams aren't the only way to do that. This article will give you a self-guided way of testing your PowerShell capabilities.

The assumption is that you've finished reading *Learn Windows PowerShell in a Month of Lunches,* that you've completed all of the hands-on exercises, and that you've watched at least the free online videos (which are indexed at MoreLunches.com). In this article, you'll be given a task to perform, and given very specific criteria that you have to meet. At the end of this article is a sample solution that meets all of those criteria. This article doesn't explain the example; just as in a certification exam, you've either gotten it right or wrong. This is meant to verify what you know, not to teach you. Everything you need to know is already covered in the book.

If you complete this assignment and your answer looks substantially different from the sample solution, consider visiting http://connect.concentratedtech.com. You'll have to register for an account (when asked where you took a class, enter "Lunches"), but there you can ask me (Don) directly about your solution. Post your code in a message, and I'll get back to you as soon as I can with some tips.

Ready?

1.1 The Assignment

Your goal is to make a standalone tool that looks, works, and feels as much like a "real" PowerShell cmdlet as possible. *Learn Windows PowerShell in a Month of Lunches* covers all of the skills you need to complete this task, although the book doesn't specifically cover every single feature that you'll need to use here. The book definitely shows you how to discover all of the necessary features on your own, using PowerShell's help files, and part of this assignment is to test your ability to self-teach.

You can obviously use search engines to look for example, but for this assignment you should regard that as cheating. The real goal here is to *test yourself,* including your ability to use PowerShell's built-in help to learn what you need to know. The only time it's okay for you to use a search engine is to look up the online documentation for the Win32_OperatingSystem class and its Win32Shutdown() method, since that information isn't contained within PowerShell's help file. In fact, the Web page you need is at http://msdn.microsoft.com/en-us/library/aa394058(v=VS.85).aspx.

There are a few other topics that can be difficult to figure out, even using the built-in help. Where appropriate, I'll provide URLs to online tutorials that should get you through.

Your basic goal is to create an advanced function that can be used to restart, shutdown, power down, or log off one or more remote computers. Now, the book doesn't cover advanced functions – but they're really just like the "pipeline functions" that the book does cover, with a bit of extra typing to make PowerShell do some work – like parameter validation – for you. They're well-documented in PowerShell's help, with plenty of examples. You can also read a blog post I wrote, http://www.windowsitpro.com/blog/powershell-with-a-purpose-blog-36/scripting-languages/advanced-functions-part-1-cmdlets-from-scripts-137369, for more information and even a template. Although I'd rather you try to use PowerShell's built-in help ☺.

This advanced function must have the following characteristics:

- Name your function Restart-Host. This will avoid conflicting with the built-in `Restart-Computer` cmdlet.
- It must have a mandatory –computername parameter that accepts one or more computer names or IP addresses. It must also accept pipeline input ByValue and ByPropertyName.
- It must have a mandatory –action parameter that accepts one of these four strings: LogOff, Shutdown, Restart, PowerOff. If any other string is specified for this mandatory parameter, an error should be displayed to the user.
- It must have a –force parameter. This doesn't accept a value, and should be set up as a *switch,* meaning the user either specifies this parameter or omits it. When specified, your function must execute a forced LogOff, Shutdown, Restart, or PowerOff; when

omitted, your function must execute a non-forced action. The Win32_OperatingSystem documentation page referenced above contains the values needed for both forced and non-forced operations.

- Note that the Win32Shutdown() method accepts numeric values; you will have to translate the user-provided action (like "Restart") into the appropriate value, based on the documentation Web page.
- If a WMI connection fails for any reason, the failed computer name should be logged to a file. A user can specify a log file name by using a –logfile parameter, which should default to "errors.txt." The log file should be deleted each time the function is used, and any failed computer names should be listed one name per line in the file.
- The function must support the -? parameter, and must also support the use of Help, Man, and Get-Help.
- Be sure that your function's help includes usage examples.
- If a computer name and/or action aren't specified by the user, they should be prompted for those items.
- The function should display no output if everything works properly. However, it must support a –verbose parameter that causes the function to display what computer it is currently attempting to connect to.
- You should assume that the person running the function has administrative permissions on all computers they target. Don't worry about permissions or credentials as part of your function.
- The function should properly support both the –whatif and –confirm parameters. This is probably one of the trickiest bits, and it isn't covered in the book – nor is it especially well-covered in PowerShell's help files. Try reading a blog post I wrote at http://www.windowsitpro.com/article/powershell-faqs/q-how-do-i-make-a-powershell-function-support-shouldprocess.

As an example, you should be able to run this:

```
'localhost','server1' |
Restart-Host -action Restart -force -whatif -verbose
```

And get this output:

```
VERBOSE: Checking action and force settings
VERBOSE: Deleting error log file errors.txt
VERBOSE: Action set to Restart with a value of 6
VERBOSE: Attempting to connect to localhost
What if: Performing operation "Restart-Host" on Target "localhost".
VERBOSE: Attempting to connect to server1
What if: Performing operation "Restart-Host" on Target "server1".
```

If you need a few hints to get you started, refer back to *Learn Windows PowerShell in a Month of Lunches.* Also, here are a few pointers:

- You'll use both `Get-WmiObject` and `Invoke-WmiMethod` cmdlets.
- Running `Help *advanced*` will reveal PowerShell's advanced function help files.
- Running `Help *comment*` will reveal information about how to properly include help for your function.

That's all you get. Jump in and start solving the problem.

1.2 Sample Solution

With any problem and solution this complex, you're bound to see differences between what you came up with and what I've listed here. Some differences are unimportant – like the exact variable names you and I picked. Some differences may even be due to the way I had to format my code to make it fit within this article's layout – and those differences aren't important, either. Other differences may be important, and you may want to ask yourself why you chose a different approach than I did. If you don't understand why I chose the approach I did, visit http://connect.concentratedtech.com and ask.

My solution embodies a number of best practices that aren't specifically discussed in the book, so some differences between your code and mine may stem from those practices. Again, if you're unsure, visit the Connect Web site and ask. You might even learn something new!

```
function Restart-Host {
<#
.SYNOPSIS
Restarts one or more computers using the WMI
Win32_OperatingSystem method.
.DESCRIPTION
Restarts, shuts down, logs off, or powers down
one or more computers. This relies on WMI's
Win32_OperatingSystem class. Supports common parameters
-verbose, -whatif, and -confirm.
.PARAM computername
One or more computer names to operate against.
Accepts pipeline input ByValue and ByPropertyName.
.PARAM action
Can be Restart, LogOff, Shutdown, or PowerOff
.PARAM force
Specify this to force the action
.PARAM logfile
Defaults to errors.txt; will contain the names of any computers
that cannot be connected to via WMI.
#>
    [CmdletBinding(
        SupportsShouldProcess=$true,
        ConfirmImpact="High"
    )]
    param (
        [parameter(Mandatory=$true,
                   ValueFromPipeline=$true,
                   ValueFromPipelineByPropertyName=$true)]
        [string[]]$computerName,

        [parameter(Mandatory=$true)]
        [ValidateSet("Restart","LogOff","Shutdown","PowerOff")]
        [string]$action,

        [switch]$force = $false,

        [string]$logfile = 'errors.txt'
```

```
    )
    BEGIN {
        Write-Verbose 'Checking action and force settings'
        switch ($action) {
            "Restart" {
                $_action = 2
                break
            }
            "LogOff" {
                $_action = 0
                break
            }
            "Shutdown" {
                $_action = 2
                break
            }
            "PowerOff" {
                $_action = 8
                break
            }
        }

        # to force, add 4 to the value
        if ($force) {
            $_action += 4
        }

        Write-Verbose "Deleting error log file $logfile"
        Remove-Item $logfile -ErrorAction SilentlyContinue

        write-verbose "Action set to $action with a value of $_action"
    }

    PROCESS {
        foreach ($computer in $computername) {
            write-verbose "Attempting to connect to $computer"
            if ($pscmdlet.ShouldProcess($computername)) {
                try {
                    Write-Verbose "Attempting $computer"
                    get-wmiobject win32_operatingsystem `
                        -computername $computer |
                        invoke-wmimethod -name Win32Shutdown `
                        -argumentlist $_action |
                        Out-Null
                } catch {
                    Write-Verbose "$computer could not be reached"
                    $computer | Out-File $logfile -append
                }
            }
         }
    }

}
```