Windows PowerShell: Build a Better Function

The Advanced Functions in Windows PowerShell 2.0 let you emulate native cmdlets with a relatively simple script.

Don Jones

In Windows PowerShell 2.0, Microsoft introduced a new type of function called an “advanced function.” A lot of folks call this a “script cmdlet.” The idea with these functions is that you can now use the simplified scripting language of Windows PowerShell to create something that looks, works, smells and feels almost exactly like a real, native Windows PowerShell cmdlet.

Obviously, not all of you are going to be in a position to create reusable tools for yourself and your coworkers. However, if you are creating reusable tools, these advanced functions are the only way to go. In fact, I’m currently modifying my classroom courseware to only teach this kind of function.

Because advanced functions work so much like real cmdlets, a properly made function can be easier for others to use because it “fits” with the way the rest of the shell works. That being the case, here’s a template that you can use to more quickly create these advanced functions:

```
function Get-Something {
  <#
  .SYNOPSIS
  Describe the function here
  .DESCRIPTION
  Describe the function in more detail
  .EXAMPLE
  Give an example of how to use it
  .EXAMPLE
  Give another example of how to use it
  .PARAMETER computername
  The computer name to query. Just one.
  .PARAMETER logname
  The name of a file to write failed computer names to. Defaults to errors.txt.
  #>
  [CmdletBinding()]
  param
  (
    [Parameter(Mandatory=$True,
    ValueFromPipeline=$True,
    ValueFromPipelineByPropertyName=$True,
      HelpMessage='What computer name would you like to target?')]
    [Alias('host')]
    [ValidateLength(3,30)]
    [string[]]$computername,

    [string]$logname = 'errors.txt'
  )

  begin {
  write-verbose "Deleting $logname"
    del $logname -ErrorActionSilentlyContinue
  }

  process {

    write-verbose "Beginning process loop"

    foreach ($computer in $computername) {
      Write-Verbose "Processing $computer"
      # use $computer to target a single computer
      # create a hashtable with your output info
      $info = @{
        'info1'=$value1;
        'info2'=$value2;
        'info3'=$value3;
        'info4'=$value4
      }
      Write-Output (New-Object –TypenamePSObject –Prop $info)
    }
  }
}
```

There are a few important things to notice about this function—many of which you'll need to tweak when using this template:

- The function name should look like a cmdlet name, starting with one of the commonly used Windows PowerShell verbs like "Get" and ending with a singular noun.
- Fill in the comment-based help. Provide a description of each parameter, a description of the overall function, and examples of how it's used. Run help about_comment_based_help for more information on writing help in this fashion.
- The first parameter, $computername, is fairly complex. It can accept one or more values on the parameter or from the pipeline. It only accepts values that are three to 30 characters long. You can use –host instead of –computername when running the function. This parameter is mandatory. If someone runs the function without providing a computer name, the shell will prompt them to enter one or more.
- The second parameter, $logname, is a simpler parameter. It just accepts a string value. It also has a default.
- The BEGIN script block tries to delete any existing log file, so each run of the function has a fresh log file.
- Within the PROCESS script block, I've indicated where you would use the $computer variable to have your function work with a single computer. You don't work with $computername directly. That parameter can contain one or more values, so the ForEach loop enumerates them and puts just one at a time into $computer for you.
- The $info hashtable should contain your desired output format. Think of this as tabular output. I've created columns called info1, info2, info3 and info4. The values for those columns are pulled from the $valuex variables. This is just an example. I haven't actually put anything into those variables in this template, so you can replace $value1, $value2 and so forth with your own information.

This template is good for any advanced function that needs to get information, but not actually make any changes to the state of the system. For advanced functions that will change the state of the system, you need to implement a few extra items. With that in mind, here's a second template:

```
function Do-Something {
  <#
  .SYNOPSIS
  Describe the function here
  .DESCRIPTION
  Describe the function in more detail
  .EXAMPLE
  Give an example of how to use it
  .EXAMPLE
  Give another example of how to use it
  .PARAMETER computername
  The computer name to query. Just one.
  .PARAMETER logname
  The name of a file to write failed computer names to. Defaults to errors.txt.
  #>
  [CmdletBinding(SupportsShouldProcess=$True,ConfirmImpact='Low')]
  param
  (
    [Parameter(Mandatory=$True,
    ValueFromPipeline=$True,
    ValueFromPipelineByPropertyName=$True,
      HelpMessage='What computer name would you like to target?')]
    [Alias('host')]
    [ValidateLength(3,30)]
    [string[]]$computername,

    [string]$logname = 'errors.txt'
  )

  begin {
  write-verbose "Deleting $logname"
    del $logname –ErrorActionSilentlyContinue
  }

  process {

    write-verbose "Beginning process loop"

    foreach ($computer in $computername) {
      Write-Verbose "Processing $computer"
      if ($pscmdlet.ShouldProcess($computer)) {
        # use $computer here
      }
    }
  }
}
```

In this template, you'll need to modify the "ConfirmImpact" setting. You need to give a relative indication of the potential danger of your advanced function. For example, doing something minor like changing a file's ReadOnly attribute might rate a ConfirmImpact of "Low." Restarting the computer might be a ConfirmImpact of "High." Windows PowerShell uses this setting, along with the shell's built-in $ConfirmPreference variable, to determine whether to automatically provide "Are you sure?" prompts when you run the function.

It's noted where you should actually perform your action, using $computer to target a single computer. That's wrapped in an "If" construct, which uses the built-in $pscmdlet object. If this function is run with either the –whatif or –confirm parameters (which it will support), the $pscmdlet object will take care of producing the "what if" output or generating the "Are you sure?" prompts.

Both of these templates will also support the –verbose parameter, and use Write-Verbose to generate status messages as the function executes. Obviously, not every function will need to target computers. Sometimes, you'll want parameters to accept file names, user names or other information. Just modify the parameter declarations appropriately.

You can find more information on the parameter attributes—such as validation, mandatory, pipeline input and so on—by running help about_functions_advanced_parameter in the shell. Enjoy.

Don Jones *is a popular Windows PowerShell author, trainer and speaker. His most recent book is "Learn Windows PowerShell in a Month of Lunches" (Manning, 2011); visit MoreLunches.com for info and free companion content. He's also available for on-site training classes (visit ConcentratedTech.com/training for more information).*

Related Content

- Windows PowerShell: Package and Distribute Custom Windows PowerShell Tools
- Windows PowerShell: Make a Command into a Reusable Tool
- Windows PowerShell: Think Commands, Not Scripts