WorkshopPLUS Windows PowerShell Desired State Configuration

Module 6: Custom Resources

Student Lab Manual

Version 1.2

Contents

Lab 6: Custom Resources

Introduction

There are a small number of built-in DSC resources and a large number available in the public domain, mostly open source on GitHub and the PowerShell Gallery. These do not, however, cover every possible scenario in every environment. During this lab we will create our own **custom resource**.

We will utilize a tool known as the **Resource Designer** that will create the framework of the custom DSC resource module, and then we will write the custom code to fill the three functions **Get-TargetResource**, **Set-TargetResource** and **Test-TargetResource**.

Objectives

After completing this lab, you will be able to:

- Utilize the Resource Designer to create a DSC custom resource.
- Identify the key attributes that are used within a DSC resource
- Execute manual testing against a DSC resource
- Deploy a configuration that utilizes your custom resource

Prerequisites

If you have not completed the Module 1 lab, then you will need to install WMF 5.0 by running the following two scripts in the order listed:

- C:\PShell\Labs\Lab00\Lab_00_01_01_Stage_Resources.ps1
- C:\PShell\Labs\Lab00\Lab_00_01_02_Install_WMF_5.ps1

Start all VMs provided for the workshop labs.

Logon to the **PULL** server as **Administrator** with the password **PowerShell5!**

Estimated time to complete this lab

75 minutes

For more information

The information on DSC custom resources is available on TechNet:

Build Custom Windows PowerShell Desired State Configuration Resources
https://msdn.microsoft.com/en-us/powershell/dsc/authoringResource

Resource Designer Walkthrough
http://blogs.msdn.com/b/powershell/archive/2013/11/19/resource-designer-tool-a-walkthrough-writing-a-dsc-resource.aspx

Invoking DSC Resources Directly
http://blogs.msdn.com/b/powershell/archive/2015/02/27/invoking-dsc-resource-methods-directly-from-powershell-cim-apis.aspx

Scenario

You have a custom application in your environment that creates a large amount of log files. The log files roll over daily, so old log files collect on servers taking up valuable disk space. You must write a custom DSC resource that will remove old log files.

You should specify via the configuration document the age of the log files to be deleted and the path to the log file directory. **Get-DscConfiguration** should report which log files are old and will be deleted, and which files are newer and will not be deleted.

NOTE: Use the Windows PowerShell Integrated Scripting Environment (ISE) for typing and editing the PowerShell scripts in this course. The automatic Intellisense will be very helpful in the learning process. If you need to reactivate Intellisense, use the **CTRL SPACE** keyboard shortcut.

Each lab has its own files. For example, the lab exercises for module 6 have a folder at **C:\PShell\Labs\Lab06** on the **PULL** virtual machine. Use these provided files to avoid typing long sections of code.

We recommend that you connect to the virtual machines for these labs through Remote Desktop rather than connecting through the Hyper-V console. When you connect with Remote Desktop, you can copy and paste from one virtual machine to another.

Exercise 6.1: Create a Custom DSC Resource

Objectives

In this exercise, you will:

- Install and use the DSC Resource Designer module
- Identity key attributes on a custom DSC resource
- Implement the three functions within a DSC resource

Scenario

As an IT professional who leverages PowerShell regularly, it is important to understand PowerShell cmdlets and syntax. When dealing with technologies such as Desired State Configuration (DSC), we come across other file formats, such as Managed Object Format (MOF). The MOF format is used in several ways within DSC, including to define the schema of DSC MOF-based resources. In this exercise we will leverage a tool known as the Resource Designer that can create the MOF file for us. We do not have to learn the specific syntax of the MOF language, allowing us to focus on the core PowerShell DSC cmdlets.

Task 6.1.1: Install and discover the DSC Resource Designer

1. Confirm that the DSC Resource Designer is not installed:

```
Get-Module -Name xDSCResourceDesigner -ListAvailable
```

Question A: Was there any output from the previous command?

____

2. Check the PSGallery for the module:

```
Find-Module -Name xDSCResourceDesigner
```

3. If you are prompted to install **NuGet-anycpu.exe**, select **Yes** to confirm.

Question B: What is the module description?

____

4. Install the module:

```
Install-Module -Name xDSCResourceDesigner
```

5. If you are prompted to install from an untrusted repository, select **Yes** to confirm.
6. Confirm the installation ran successfully:

```
Get-Module -Name xDSCResourceDesigner –ListAvailable
```

7. In the ISE use the **Show Command** add-on to view the commands within the **xDSCResourceDesigner** module.
    a. Next to the **Modules** box drop-down click **Refresh**.
    b. In the **Modules** drop-down select the **xDSCResourceDesigner**.

> **NOTE:** If you cannot see the **Commands Pane** on the right side of the ISE, go to the **View** menu and ensure that the "Show Command Add-on" is selected.

Question C: How many commands are there in the module?

____

8. Open the following lab file:
C:\PShell\Labs\Lab06\Lab_06_01_01_CustomDSCResource.ps1
9. This script utilizes the **xDSCResourceDesigner** module cmdlets to create a new DSC resource. Review the script.
10. In order to delete old log files, we need to know the following:
    a. The directory (**Path**) to search
    b. The age of the files in days (**FileAgeDays**)
    c. The list of old files (**OldFiles**) that will be deleted
    d. The list of new files (**NewFiles**) that will not be deleted. This can be useful for reporting.
11. Observe the **New-xDscResourceProperty** cmdlets at the top of the script.
    a. Path will be the unique and required **Key** attribute.
    b. FileAgeDays is a **Required** attribute.
    c. OldFiles and NewFiles are **Read-**only attributes that will be reported from the get function.
12. When we create the DSC resource, these are the parameters or resource properties that will be defined in the **schema.mof**.

> **NOTE:** View a full walkthrough and all guidelines for creating a custom resource at the following URL:
>
> http://blogs.msdn.com/b/powershell/archive/2013/11/19/resource-designer-tool-a-walkthrough-writing-a-dsc-resource.aspx

13. Execute the script by pressing **F5**. It will perform the following tasks:

   a. Create the template files for a custom DSC resource module called **xAppMaintenance**
   b. Open the module resource file **Contoso_xClearLogFiles.psm1** for editing
   c. View the **schema.mof** file
   d. Print the tree output of the file structure for the DSC resource module

14. Review the **schema.mof** file now open in the ISE. Compare it to the script you ran, then close the file.

> **NOTE:** The **psd1** file is a module manifest, and the **psm1** is a script module. These are the standard file formats for PowerShell modules in general. DSC resources also contain the unique **schema.mof** file. You could write this file yourself. However, the resource designer module creates this file for you along with the other file structure.

15. All DSC resources have three core functions: **Get-TargetResource**, **Set-TargetResource**, and **Test-TargetResource**. Your task is to complete the code within these three functions shown in the file **Contoso_xClearLogFiles.psm1** opened for you in the ISE.

16. Review the above file and the green comments within each function.

Question D: What type of object should the variable **$returnValue** be set to in the **Get-TargetResource** function?

Question E: What type of object should the variable **$result** be set to in the **Test-TargetResource** function?

Question F: If the resource that you create requires a reboot of the system, it is important that you do not directly perform the reboot. Instead, use the special global variable to notify the LocalConfigurationManager (LCM). What is the name of this variable in the **Set-TargetResource** function?

---

17. Open the following lab file:
    C:\PShell\Labs\Lab06\Lab_06_01_01_CustomDSCResourceSampleCode.ps1

18. This file contains three regions. Copy and paste the sample code into the open file **Contoso_xClearLogFiles.psm1** in the following steps. Copy the code **within** the first region for the **Get-TargetResource**.

```
#region Get-TargetResource code sample

        <Copy this code inside of the region comments>

#endregion Get
```

19. Select the open file in the ISE named **Contoso_xClearLogFiles.psm1**
20. Locate the first function **Get-TargetResource**.
21. Select **all** of the green comments within the function, and then paste in the sample code inside using **CTRL+V**.
22. Select the open file in the ISE named **Lab_06_01_01_CustomDSCResourceSampleCode.ps1**.
23. Copy the code within the second region for the **Set-TargetResource**.

```
#region Set-TargetResource code sample

        <Copy this code inside of the region comments>

#endregion Set
```

24. Select the open file in the ISE named **Contoso_xClearLogFiles.psm1**.
25. Locate the second function **Set-TargetResource**.
26. Select **all** of the green comments within the function, and then paste in the sample code inside using **CTRL+V**.
27. Select the open file in the ISE named **Lab_06_01_01_CustomDSCResourceSampleCode.ps1**.
28. Copy the code within the third region for the **Test-TargetResource**.

```
#region Test-TargetResource code sample

        <Copy this code inside of the region comments>

#endregion Test
```

29. Select the open file in the ISE named **Contoso_xClearLogFiles.psm1**.
30. Locate the third function **Test-TargetResource**.
31. Select **all** of the green comments within the function, and then paste in the sample code inside using **CTRL+V**.
32. Save the file named **Contoso_xClearLogFiles.psm1**.
33. Review the three functions in the file.

NOTE: The functions within a DSC resource utilize standard PowerShell code and syntax. These three functions enable DSC's idempotent manner.

34. Review the final code to ensure that you copied and pasted correctly. Compare yours to the solution file **C:\PShell\Labs\Lab06\Lab_06_01_01_CustomDSCResourceSolution.ps1**.

Exercise 6.2: Testing the Custom DSC Resource

Objectives

In this exercise, you will:

- Use the **Invoke-DSCResource** cmdlet
- Test your custom resource to ensure there are no errors

Scenario

There are many occasions when you may want to manually test a DSC resource:

- Discovery of new resources
- Testing your own custom resource
- Testing a resource in a new environment (different operating system, Windows Management Framework version, etc.)

The **Invoke-DSCResource** cmdlet can directly call any of the Get, Test or Set methods and pass any required parameters. You can do this without authoring an entire DSC configuration script.

In this exercise you will use this cmdlet to test your new custom resource.

Task 6.2.1: Testing the DSC Resource

1. In order to test our custom resource we will use a setup script to create sample application log files. Open the script: **C:\PShell\Labs\Lab06\Lab_06_02_01_CreateAppLogFiles.ps1**
2. This script will create a directory and sample log files at **C:\MyAppLogs**. Run the script by pressing **F5**.
3. Select the open file in the ISE named **Contoso_xClearLogFiles.psm1**.
4. Review the function **Get-TargetResource**. Notice that it will return a value that contains the four properties that we described in the attributes for our resource.
   - Path
   - FileAgeDays
   - OldFiles
   - NewFiles
5. Notice that the **Get-TargetResource** function takes two parameters. We will pass these values into the function when we test it.
6. We will call the method **Get** directly from the module and pass in the required parameters. Test the resource by executing the following command:

NOTE: The code below should be typed on a single line.

```
Invoke-DscResource -Name xClearLogFiles -Method Get -ModuleName xAppMaintenance
 -Property @{Path = 'C:\MyAppLogs'; FileAgeDays = 5}
```

Question G: What happens when you execute this command?

---

7. Now execute the **Test** Method:

```
Invoke-DscResource -Name xClearLogFiles -Method test -ModuleName xAppMaintenance
 -Property @{Path = 'C:\MyAppLogs'; FileAgeDays = 5} –Verbose
```

Question H: What is the result?

Question I: Run this command multiple times, changing the value of **FileAgeDays**. What minimum value for **FileAgeDays** returns **InDesiredState** to be **True**?

---

8. Now execute the **Set** method:

```
Invoke-DscResource -Name xClearLogFiles -Method Set -ModuleName xAppMaintenance
 -Property @{Path = 'C:\MyAppLogs'; FileAgeDays = 5} –Verbose
```

9. When calling the **Set** method you invoke the LocalConfigurationManager (LCM) to apply the specific settings and bring the configuration into desired state. In this case any files older than **FileAgeDays** will be deleted.
10. Verify that the system is in compliance using the **Test** method:

```
Invoke-DscResource -Name xClearLogFiles -Method test -ModuleName xAppMaintenance
 -Property @{Path = 'C:\MyAppLogs'; FileAgeDays = 5} –Verbose
```

11. Using **Invoke-DSCResource** is the recommended way to ensure that your code is tested prior to writing your first configuration that utilizes the resource. Now we have tested the logic within our custom DSC resource using known-good values.

CHALLENGE: If you finish this lab early, return to this exercise and test the resource using invalid data. Then add error handling logic to the resource functions.

Exercise 6.3: Consuming the Custom DSC Resource

Objectives

In this exercise, you will:

- Write a configuration script that uses a custom resource

Scenario

This exercise is similar to previous labs where you authored a DSC configuration. However, this time you are using your own custom resource. Observe how the resource attributes you created are now exposed as properties by Intellisense.

Task 6.3.1: Create a DSC Configuration

1. Now that the custom resource has been created and tested, we can discover and utilize the resource like any other DSC resource. Execute the following command:

```
Get-DscResource -Name *LogFiles*
```

Question J: What is the name of the Module?

2. Discover the capabilities of the resource by viewing the syntax:

```
Get-DscResource -Name xClearLogFiles –Syntax
```

Question K: What are the required properties that you must supply when consuming this resource?

NOTE: Required properties are those shown without the surrounding brackets [ ].

3. Open the lab file template configuration: **C:\PShell\Labs\Lab06\Lab_06_03_01_ClearAppLogFiles.ps1**
4. Review the important sections of the document
   a. Configuration block with name
   b. Node block with name
   c. Import the resource module
5. Next we will use the **xClearLogFiles** resource. All elements for writing this configuration can be discovered with Intellisense (**CTRL**+**SPACE**) and **TAB** complete. Click on the next line after the green comment. Press **CTRL**+**SPACE** to

begin Intellisense. Scroll to the bottom of the resource list using the arrow keys. Select **xClearLogFiles** and press **ENTER**.

6. Delete the resource name you typed just now. Then type **xClear** and press **TAB**.
7. The resource name will be completed to **xClearLogFiles**.
8. Now provide the name **Clear7Days**. Open and close the scriptblock for the resource with curly braces as shown below.

```
        # Resources go here.
        xClearLogFiles Clear7Days
        {

        }
```

9. Click inside the resource scriptblock and press **CTRL**+**SPACE** to view the available properties. These properties are the same ones that we previously saw with output from the syntax command.
10. Define the two required parameters and their values as below.

```
        # Resources go here.
        xClearLogFiles Clear7Days
        {
            Path        = 'C:\MyAppLogs'
            FileAgeDays = 7
        }
```

11. Now press **F5** to load the configuration. Select **Save** if prompted.
12. The configuration is now loaded into memory. Execute the following to confirm:

```
Get-Command -Name ClearAppLogFiles
```

13. Type and Execute the following line below the configuration:

```
ClearAppLogFiles
```

14. Type and Execute the following to push the configuration to the localhost:

```
Start-DscConfiguration -Path .\ClearAppLogFiles -Verbose -Wait
```

Question L: How long does the configuration take to complete?

____

Question M: How many files were deleted?

____

15. Select the open file in the ISE named **Lab_06_02_01_CreateAppLogFiles.ps1**.
16. Run the script twice to seed some new log files.
17. Select the open Tab in the ISE named **Lab_06_03_01_ClearAppLogFiles.ps1**.

18. Type and Execute the following:

```
Get-DSCConfiguration
```

Question N: What command and method does this output resemble from the previous exercise?

19. Type and execute the following:

```
Test-DscConfiguration
```

Question O: Is the system in desired state?

20. Type and execute the following:

```
Test-DscConfiguration –Detailed
```

Question P: What is the name of the **ResourcesNotInDesiredState**?

21. Execute the following command again to bring the system back into the desired state:

```
Start-DscConfiguration -Path .\ClearAppLogFiles -Verbose –Wait
```

22. Type and execute the following to confirm the system is now back in the desired state:

```
Test-DscConfiguration –Detailed
```