Windows PowerShell 4.0 For the IT Professional - Part 1

Module 7: Objects

Student Lab Manual

Version 2.0

Conditions and Terms of Use

Microsoft Confidential - For Internal Use Only

Contents

Lab 7: Objects

Introduction

Every action you take in Windows PowerShell occurs within the context of objects. As data moves from one command to the next, it moves as one or more identifiable objects. An object, then, is a collection of data that represents an item. An object consists of three types of data: the object's type, its methods, and its properties.

Objectives

After completing this lab, you will be able to:

- Discover object members
- Use object properties and methods
- Compare objects

Prerequisites

Start all VMs provided for the workshop labs.

Logon to WIN8-WS as:

Username: **Contoso\Administrator**

Password: **PowerShell4**

Estimated time to complete this lab

45 minutes

NOTE: These exercises use many Windows PowerShell commands. You can type these commands into the Windows PowerShell Integrated Scripting Environment (ISE) or the Windows PowerShell console. For some exercises, you can load pre-typed lab files into the Windows PowerShell ISE, allowing you to select and execute individual commands. Each lab has its own folder under **C:\PShell\Labs** on **WIN8-WS**.

Some exercises in this workshop may require running the Windows PowerShell console or ISE as an elevated user (Run as Administrator).

We recommend that you connect to the virtual machines (VMs) for these labs through Remote Desktop rather than connecting through the Hyper-V console. This allows you to use copy and paste between VMs and the host machine. If you are using the online hosted labs, then you are already using Remote Desktop.

Exercise 7.1: Types and members

Introduction

Everything in Windows PowerShell is an object. Objects are instances of templates, known as data types. In this exercise, we will investigate how to discover this crucial information and manage any object you may encounter.

Objectives

After completing this exercise, you will be able to:

- Use the Get-Member Cmdlet to return object datatype and member information.
- Access object properties
- Execute object methods

Task 7.1.1: Discovering objects

1. Complete the following statements by choosing the correct words from the box below. Refer to slide 233 in the workshop PowerPoint presentation.

methods, object, action, type, properties, instance, template, members, state

A type represents a construct that defines a _____ of members. Each object has a _____. Objects have data fields called _____. Objects have procedures, known as _____. Properties and methods are collectively known as _____.

A type represents a construct that defines a TEMPLATE of members. Each object has a TYPE. Objects have data fields called PROPERTIES. Objects have procedures, known as METHODS. Properties and methods are collectively known as MEMBERS.

2. Use the Get-Service Cmdlet to return only the 'Spooler' service. How many column headers do you observe?

3. Using the same command as the previous step, create a simple pipeline to format the output as a list and choose to display all properties.

```
Get-Service -Name Spooler | Format-List -Property *
```

How many properties are listed?

Status, Name, DisplayName

4. Get the process called "lsass". Eight properties should be visible in the table output.

5. Pipe the output of the previous command to the Format-List Cmdlet, specifying that all properties are returned with the wildcard character (*). A long list of properties should be visible. Note that the second line uses the popular alias to do the same.

```
Get-Process –Name lsass | Format-List –Property *
Get-Process –Name lsass | fl *
```

The object's parent type (or class) defines the returned properties.

6. There are a number of ways to discover an object's type. The GetType() method exists on every object and shows the name property of a particular object's type.

```
(Get-Process –Name lsass).GetType()

IsPublic IsSerial Name            BaseType
-------- -------- ----            --------
True     False    Process         System.ComponentModel.Component
```

7. To get the fully qualified type name, access the fullname property.

```
(Get-Process –Name lsass).GetType().Fullname
System.Diagnostics.Process
```

8. What is the type of each value in the list below?

Use the GetType() method to complete the type names on the right-hand side of the table below. For example, to find the data type of the string "Microsoft", simply type.

```
"Microsoft".GetType().Name
String
```

Note: Enclose numeric values in smooth parentheses e.g. (3).GetType().Name

Value	Name
"Microsoft"	String
10	
23.7	
$true	
Get-Service –Name Spooler	
Get-Item –Path C:\windows	
Get-Process	

String
Int32
Double
Boolean
ServiceController
DirectoryInfo
Object[]

9. A richer way to investigate the object members is to use the Get-Member Cmdlet. This Cmdlet interrogates the object (or list of objects) and displays their properties and methods.

Type the following command and answer the questions below.

```
Get-ChildItem –Path C:\Windows | Get-Member
```

a) How many different TypeNames were returned?

2

b) List the Type Names below.

1. System.IO.DirectoryInfo

2. System.IO.FileInfo

c) Get help for the Get-Member Cmdlet to find a parameter that will allow you to filter member types using Get-ChildItem –Path C:\Windows as pipeline input.
How would you modify the above command to display only the 'property' member type? Write the command below.

Get-ChildItem -Path C:\Windows | Get-Member -MemberType Property

d) Use the pipeline to filter the output of the command below.

```
Get-ChildItem –Path C:\Windows
```

Filter the 'Method' member type for the objects returned by this Cmdlet and then measure their total number.

```
Get-ChildItem -Path C:\Windows |
Get-Member -MemberType Method |
Group-Object -Property TypeName
```

How many methods does a FileInfo object have? _____

24

10. Start Calc.exe
11. Get all the processes with the process name "Calc"
12. Using Get-Member parameters, find all properties with the string 'Memory' in the property Name.

```
Get-Process calc | Get-Member -Name *memory*
```

Get-Process -Name calc | Get-Member -MemberType Properties -Name *memory*

TIP: Most Cmdlet properties accept the wildcard characters: *,?

13. Study the Definition column from the output. What difference do you notice between the properties, other than "64" appended to their names?

The Type of the *64 properties is 'long', whereas the 32 bit properties use an 'int'.

14. Display the methods available on a string object, using the code below.

```
"Hello World" | Get-Member –MemberType Method
```

15. Which member can you use to convert the string to Upper case characters?

.ToUpper()

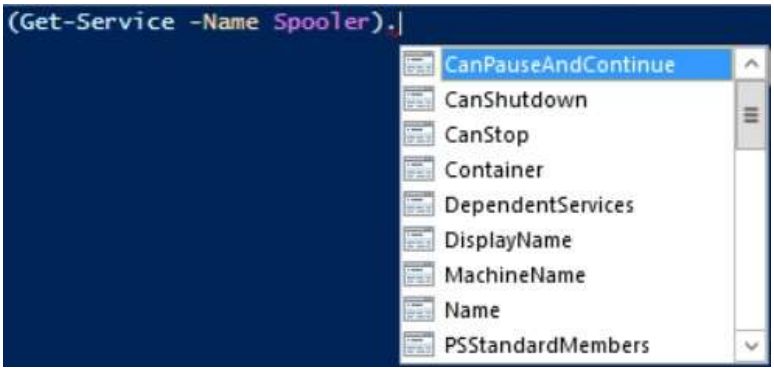
Task 7.1.2: Accessing object properties

In the previous task, we saw objects interrogated for their member information. In this section, we will access the data stored in object properties.

1. Use Get-Service Cmdlet to return only the 'Spooler' service and pipe the object to the Get-Member Cmdlet.

```
Get-Service –Name Spooler | Get-Member
```

2. Modify the Get-Member command to list only the "property" member types.
3. If we enclose the Cmdlet in smooth parentheses, we are able to cycle through the objects members by typing a 'dot' (.) character after the right-hand parenthesis. The dot character is known as the "dereference operator".



NOTE: The Windows PowerShell ISE will display a drop down list containing all of the object's members.

The Windows PowerShell console (not the ISE) requires you to press the <TAB> key after typing the dot. You can then cycle through the members by continually pressing <TAB>. Pressing <Shift> <TAB> will cycle in reverse order.

4. Select the "status" property and press <Enter>. Return the value of the Status property by pressing <Enter> again.

```
(Get-Service –Name Spooler).Status
Running
```

5. Find the property that holds a list of the services that this service depends on. Execute the command to return the service names and record them below.

(Get-Service –Name Spooler).ServicesDependedOn

1. _____
2. _____

6. Type the examples below and review their output.

```
(Get-Process).MainModule
(Get-Volume).DriveLetter
(Get-NetIPAddress).IPv4Address
(Get-ADDomain).PDCEmulator
```

Task 7.1.3: Executing object methods

1. Methods can also be enumerated using the Dereference (dot) operator.
   Type the commands below, pressing <Enter> after each command.

```
Calc.exe
(Get-Process –Name calc).Kill
```

   Notice that the calc.exe process does not stop.
2. This illustrates one important difference between properties and methods.
   Methods ***require*** a pair of smooth parentheses immediately following the method's name (no spaces are allowed!). Try executing the method again, but this time include the parentheses. Did the process exit?

```
(Get-Process –Name calc).Kill()
```

3. Get the service named "Spooler".
4. Use Get-Member to list the ServiceController object's method members.
5. Find the Stop() method.
6. Execute this method against the Spooler service to stop it.

```
(Get-Service –Name Spooler).Stop()
```

7. Finally, start the Spooler service. Write the command used to do this below.

(Get-Service –Name Spooler).Start()

Task 7.1.4: Executing object methods with a parameter

1. Some methods require one or more arguments to be supplied within their smooth parentheses. Execute the Get-Date Cmdlet to return the current date and time.

```
Get-Date
```

2. The output from this Cmdlet is a string representing the current date and time. What this returns is actually an object, ***not*** what appears to be a simple string.
   This object type contains many methods for executing common date and time operations. Using the pipeline and the Get-Member Cmdlet, discover the data type of the object returned by the Get-Date Cmdlet and write the TypeName below.

   String

   _____

3. Display just the method members of the DateTime data type that start with the string "Add".

```
Get-Date | Get-Member –MemberType method –name Add*
```

4. Notice that these methods require a value, used within the method parentheses. The value must be of the specified data type.

```
Name            MemberType Definition
----            ---------- ----------
Add             Method     datetime Add(timespan value)
AddDays         Method     datetime AddDays(double value)
AddHours        Method     datetime AddHours(double value)
AddMilliseconds Method     datetime AddMilliseconds(double value)
AddMinutes      Method     datetime AddMinutes(double value)
AddMonths       Method     datetime AddMonths(int months)
AddSeconds      Method     datetime AddSeconds(double value)
AddTicks        Method     datetime AddTicks(long value)
AddYears        Method     datetime AddYears(int value)
```

5. We can use these methods to manipulate the current date and time. Type the following commands.
   a) Return the current date and time

```
Get-Date
```

   b) Add seven days to today's date

```
(Get-Date).AddDays(7)
```

   c) Subtract one hour from the current time

```
(Get-Date).AddHours(-1)
```

   d) Add a timespan to the current time – A timespan can be represented as "hh:mm:ss"

```
(Get-Date).Add("10:30:00")
```

   e) Create a new DateTime object representing 25th of December 2014, using the Get-Date Cmdlet's -Date Parameter. Here we use the date format of "Year-Month-Day". Use the Subtract() method to calculate the timespan between the current date and that date. (You can also do this using New-TimeSpan.

```
(Get-Date -Date "2014-12-25").Subtract( (Get-Date) )
New-TimeSpan -End "2014-12-25"
```

Task 7.1.5: Executing object methods with multiple parameters

1. Get a folder object that represents C:\Pshell, using the Get-Item Cmdlet .

```
Get-Item –Path C:\PShell
```

2. List the method members on the DirectoryInfo object

```
Get-Item –Path C:\PShell | Get-Member –MemberType Method
```

3. Find the "CreateSubDirectory" method and execute the method name without typing the parentheses, then press <Enter>.

```
(Get-Item –Path C:\PShell).CreateSubdirectory
```

4. Windows PowerShell conveniently displays the method definition. We can see that there are two different sets of parameter definitions for this particular method – known as "overloaded definitions".

```
CreateSubdirectory(string path)
CreateSubdirectory(string path, DirectorySecurity directorySecurity)
```

   The first overload requires a string representing the path to the new sub-directory. The second requires the path string and an additional DirectorySecurity object, used to customize the new folder's Access Control List.

   This is how you determine valid parameter sets when calling a method on an object.

Exercise 7.2: Comparing objects

Introduction

Comparing two sets of data is a common task. Windows PowerShell provides the Compare-Object Cmdlet for this purpose.

Task 7.2.1: Comparing text files

1. Get help for the Compare-Object Cmdlet. Read the DESCRIPTION section of the help to understand what the Cmdlet does.
2. Type the following commands in the order shown below.

> **NOTE:** In the example below the objects returned from the Get-Content Cmdlet are stored in a variable. All variable names in Windows PowerShell are prefixed by a dollar ($) character. Module 12 explains variables in detail.

```
$ref = Get-Content C:\PShell\Labs\Lab_7\Users_A.txt
$diff = Get-Content C:\PShell\Labs\Lab_7\Users_B.txt

Compare-Object -ReferenceObject $ref -DifferenceObject $diff
```

3. What information does the "SideIndicator" column hold?

"The result of the comparison indicates whether a property value appeared only in the object from the reference set (indicated by the <= symbol), only in the object from the difference set (indicated by the => symbol) or, if the IncludeEqual parameter is specified, in both objects (indicated by the == symbol)."

4. Which indicator symbol shows that a value only appeared in the reference list?

<=

5. Which indicator symbol shows a value appeared in both files?

==

6. Which parameters should you add to the command in step 2, to return only duplicate items in the two files? Execute the command and write it below.

-IncludeEqual -ExcludeDifferent

7. How many duplicate names were found in both files?

Task 7.2.2: Comparing process data

Compare-Object can also compare data that is more complex. In this task, we will export a list of running processes to an xml file, start two new processes and then compare the saved list with a current list of processes.

Complete the steps below in order.

1.

```
Get-Process | Export-Clixml -Path C:\PShell\Labs\Lab_7\procs.xml
```

2.

```
Notepad.exe
```

3.

```
Calc.exe
```

4.

```
$savedProcs = Import-Clixml -Path C:\PShell\Labs\Lab_7\procs.xml
```

5.

```
Compare-Object –ReferenceObject $savedProcs -DifferenceObject (Get-process) `
-Property Name
```

6.

```
Compare-Object –ReferenceObject $savedProcs -DifferenceObject (Get-process) `
-Property Name -IncludeEqual
```