Chapter 13 Lab

In this chapter you are going to assemble a module called PSHTools, from the functions and custom views that you’ve been working on for the last several chapters. Create a folder in the user module directory, called PSHTools. Put all of the files you will be creating in the labs into this folder.

* + 1. Lab A

Create a single ps1xml file that contains all of the view definitions from the 3 existing format files. Call the file PSHTools.format.ps1xml. You’ll need to be careful. Each view is defined by the <View></View> tags. These tags, and everything in between should go between the <ViewDefinition></ViewDefinition> tags.

Here is a sample solution:

<?xml version="1.0" encoding="utf-8" ?>

<Configuration>

<ViewDefinitions>

<View>

<Name>MOL.SystemInfo</Name>

<ViewSelectedBy>

<TypeName>MOL.ComputerSystemInfo</TypeName>

</ViewSelectedBy>

<ListControl>

<ListEntries>

<ListEntry>

<ListItems>

<ListItem>

<PropertyName>ComputerName</PropertyName>

</ListItem>

<ListItem>

<PropertyName>Workgroup</PropertyName>

</ListItem>

<ListItem>

<PropertyName>AdminPassword</PropertyName>

</ListItem>

<ListItem>

<Propertyname>Model</Propertyname>

</ListItem>

<ListItem>

<Propertyname>Manufacturer</Propertyname>

</ListItem>

<ListItem>

<Propertyname>SerialNumber</Propertyname>

<Label>BIOSSerialNumber</Label>

</ListItem>

<ListItem>

<Propertyname>Version</Propertyname>

<Label>OSVersion</Label>

</ListItem>

<ListItem>

<Propertyname>ServicePackMajorVersion</Propertyname>

<Label>SPVersion</Label>

</ListItem>

</ListItems>

</ListEntry>

</ListEntries>

</ListControl>

</View>

<View>

<Name>MOL.SystemInfo</Name>

<ViewSelectedBy>

<TypeName>MOL.DiskInfo</TypeName>

</ViewSelectedBy>

<TableControl>

<TableHeaders>

<TableColumnHeader>

<Width>18</Width>

</TableColumnHeader>

<TableColumnHeader/>

<TableColumnHeader>

<Label>FreeSpace(GB)</Label>

<Width>15</Width>

</TableColumnHeader>

<TableColumnHeader>

<Label>Size(GB)</Label>

<Width>10</Width>

</TableColumnHeader>

</TableHeaders>

<TableRowEntries>

<TableRowEntry>

<TableColumnItems>

<TableColumnItem>

<PropertyName>ComputerName</PropertyName>

</TableColumnItem>

<TableColumnItem>

<PropertyName>Drive</PropertyName>

</TableColumnItem>

<TableColumnItem>

<PropertyName>FreeSpace</PropertyName>

</TableColumnItem>

<TableColumnItem>

<Propertyname>Size</Propertyname>

</TableColumnItem>

</TableColumnItems>

</TableRowEntry>

</TableRowEntries>

</TableControl>

</View>

<View>

<Name>MOL.SystemInfo</Name>

<ViewSelectedBy>

<TypeName>MOL.ServiceProcessInfo</TypeName>

</ViewSelectedBy>

<TableControl>

<TableHeaders>

<TableColumnHeader>

<Width>14</Width>

</TableColumnHeader>

<TableColumnHeader>

<Label>Service</Label>

<Width>13</Width>

</TableColumnHeader>

<TableColumnHeader>

<Width>18</Width>

</TableColumnHeader>

<TableColumnHeader>

<Width>17</Width>

</TableColumnHeader>

<TableColumnHeader>

<Label>VM</Label>

<Width>14</Width>

</TableColumnHeader>

</TableHeaders>

<TableRowEntries>

<TableRowEntry>

<TableColumnItems>

<TableColumnItem>

<PropertyName>ComputerName</PropertyName>

</TableColumnItem>

<TableColumnItem>

<PropertyName>Name</PropertyName>

</TableColumnItem>

<TableColumnItem>

<PropertyName>Displayname</PropertyName>

</TableColumnItem>

<TableColumnItem>

<Propertyname>ProcessName</Propertyname>

</TableColumnItem>

<TableColumnItem>

<Propertyname>VMSize</Propertyname>

</TableColumnItem>

</TableColumnItems>

</TableRowEntry>

</TableRowEntries>

</TableControl>

</View>

<View>

<Name>MOL.SystemInfo</Name>

<ViewSelectedBy>

<TypeName>MOL.ServiceProcessInfo</TypeName>

</ViewSelectedBy>

<ListControl>

<ListEntries>

<ListEntry>

<ListItems>

<ListItem>

<PropertyName>ComputerName</PropertyName>

</ListItem>

<ListItem>

<PropertyName>Name</PropertyName>

<Label>Service</Label>

</ListItem>

<ListItem>

<PropertyName>Displayname</PropertyName>

</ListItem>

<ListItem>

<Propertyname>ProcessName</Propertyname>

</ListItem>

<ListItem>

<Propertyname>VMSize</Propertyname>

</ListItem>

<ListItem>

<Propertyname>ThreadCount</Propertyname>

</ListItem>

<ListItem>

<Propertyname>PeakPageFile</Propertyname>

</ListItem>

</ListItems>

</ListEntry>

</ListEntries>

</ListControl>

</View>

</ViewDefinitions>

</Configuration>

* + 1. Lab B

Create a single module file that contains the functions from the Labs A, B and C in Chapter 12, which should be the most current version. Export all functions in the module. Be careful to copy the function only. In your module file, also define aliases for your functions and export them as well.

Here is a sample solution:

#The PSHTools module file

Function Get-ComputerData {

<#

.SYNOPSIS

Get computer related data

.DESCRIPTION

This command will query a remote computer and return a custom object

with system information pulled from WMI. Depending on the computer

some information may not be available.

.PARAMETER Computername

The name of a computer to query. The account you use to run this function

should have admin rights on that computer.

.PARAMETER ErrorLog

Specify a path to a file to log errors. The default is C:\Errors.txt

.EXAMPLE

PS C:\> Get-ComputerData Server01

Run the command and query Server01.

.EXAMPLE

PS C:\> get-content c:\work\computers.txt | Get-ComputerData -Errorlog c:\logs\errors.txt

This expression will go through a list of computernames and pipe each name

to the command. Computernames that can't be accessed will be written to

the log file.

#>

[cmdletbinding()]

param(

[Parameter(Position=0,ValueFromPipeline=$True)]

[ValidateNotNullorEmpty()]

[string[]]$ComputerName,

[string]$ErrorLog="C:\Errors.txt"

)

Begin {

Write-Verbose "Starting Get-Computerdata"

}

Process {

foreach ($computer in $computerName) {

Write-Verbose "Getting data from $computer"

Try {

Write-Verbose "Win32\_Computersystem"

$cs = Get-WmiObject -Class Win32\_Computersystem -ComputerName $Computer -ErrorAction Stop

#decode the admin password status

Switch ($cs.AdminPasswordStatus) {

1 { $aps="Disabled" }

2 { $aps="Enabled" }

3 { $aps="NA" }

4 { $aps="Unknown" }

}

#Define a hashtable to be used for property names and values

$hash=@{

Computername=$cs.Name

Workgroup=$cs.WorkGroup

AdminPassword=$aps

Model=$cs.Model

Manufacturer=$cs.Manufacturer

}

} #Try

Catch {

#create an error message

$msg="Failed getting system information from $computer. $($\_.Exception.Message)"

Write-Error $msg

Write-Verbose "Logging errors to $errorlog"

$computer | Out-File -FilePath $Errorlog -append

} #Catch

#if there were no errors then $hash will exist and we can continue and assume

#all other WMI queries will work without error

If ($hash) {

Write-Verbose "Win32\_Bios"

$bios = Get-WmiObject -Class Win32\_Bios -ComputerName $Computer

$hash.Add("SerialNumber",$bios.SerialNumber)

Write-Verbose "Win32\_OperatingSystem"

$os = Get-WmiObject -Class Win32\_OperatingSystem -ComputerName $Computer

$hash.Add("Version",$os.Version)

$hash.Add("ServicePackMajorVersion",$os.ServicePackMajorVersion)

#create a custom object from the hash table

$obj=New-Object -TypeName PSObject -Property $hash

#add a type name to the custom object

$obj.PSObject.TypeNames.Insert(0,'MOL.ComputerSystemInfo')

Write-Output $obj

#remove $hash so it isn't accidentally re-used by a computer that causes

#an error

Remove-Variable -name hash

} #if $hash

} #foreach

} #process

End {

Write-Verbose "Ending Get-Computerdata"

}

}

Function Get-VolumeInfo {

<#

.SYNOPSIS

Get information about fixed volumes

.DESCRIPTION

This command will query a remote computer and return information about fixed

volumes. The function will ignore network, optical and other removable drives.

.PARAMETER Computername

The name of a computer to query. The account you use to run this function

should have admin rights on that computer.

.PARAMETER ErrorLog

Specify a path to a file to log errors. The default is C:\Errors.txt

.EXAMPLE

PS C:\> Get-VolumeInfo Server01

Run the command and query Server01.

.EXAMPLE

PS C:\> get-content c:\work\computers.txt | Get-VolumeInfo -errorlog c:\logs\errors.txt

This expression will go through a list of computernames and pipe each name

to the command. Computernames that can't be accessed will be written to

the log file.

#>

[cmdletbinding()]

param(

[Parameter(Position=0,ValueFromPipeline=$True)]

[ValidateNotNullorEmpty()]

[string[]]$ComputerName,

[string]$ErrorLog="C:\Errors.txt",

[switch]$LogErrors

)

Begin {

Write-Verbose "Starting Get-VolumeInfo"

}

Process {

foreach ($computer in $computerName) {

Write-Verbose "Getting data from $computer"

Try {

$data = Get-WmiObject -Class Win32\_Volume -computername $Computer -Filter "DriveType=3" -ErrorAction Stop

Foreach ($drive in $data) {

Write-Verbose "Processing volume $($drive.name)"

#format size and freespace

$Size="{0:N2}" -f ($drive.capacity/1GB)

$Freespace="{0:N2}" -f ($drive.Freespace/1GB)

#Define a hashtable to be used for property names and values

$hash=@{

Computername=$drive.SystemName

Drive=$drive.Name

FreeSpace=$Freespace

Size=$Size

}

#create a custom object from the hash table

$obj=New-Object -TypeName PSObject -Property $hash

#Add a type name to the object

$obj.PSObject.TypeNames.Insert(0,'MOL.DiskInfo')

Write-Output $obj

} #foreach

#clear $data for next computer

Remove-Variable -Name data

} #Try

Catch {

#create an error message

$msg="Failed to get volume information from $computer. $($\_.Exception.Message)"

Write-Error $msg

Write-Verbose "Logging errors to $errorlog"

$computer | Out-File -FilePath $Errorlog -append

}

} #foreach computer

} #Process

End {

Write-Verbose "Ending Get-VolumeInfo"

}

}

Function Get-ServiceInfo {

<#

.SYNOPSIS

Get service information

.DESCRIPTION

This command will query a remote computer for running services and write

a custom object to the pipeline that includes service details as well as

a few key properties from the associated process. You must run this command

with credentials that have admin rights on any remote computers.

.PARAMETER Computername

The name of a computer to query. The account you use to run this function

should have admin rights on that computer.

.PARAMETER ErrorLog

Specify a path to a file to log errors. The default is C:\Errors.txt

.PARAMETER LogErrors

If specified, computer names that can't be accessed will be logged

to the file specified by -Errorlog.

.EXAMPLE

PS C:\> Get-ServiceInfo Server01

Run the command and query Server01.

.EXAMPLE

PS C:\> get-content c:\work\computers.txt | Get-ServiceInfo -logerrors

This expression will go through a list of computernames and pipe each name

to the command. Computernames that can't be accessed will be written to

the log file.

#>

[cmdletbinding()]

param(

[Parameter(Position=0,ValueFromPipeline=$True)]

[ValidateNotNullorEmpty()]

[string[]]$ComputerName,

[string]$ErrorLog="C:\Errors.txt",

[switch]$LogErrors

)

Begin {

Write-Verbose "Starting Get-ServiceInfo"

#if -LogErrors and error log exists, delete it.

if ( (Test-Path -path $errorLog) -AND $LogErrors) {

Write-Verbose "Removing $errorlog"

Remove-Item $errorlog

}

}

Process {

foreach ($computer in $computerName) {

Write-Verbose "Getting services from $computer"

Try {

$data = Get-WmiObject -Class Win32\_Service -computername $Computer -Filter "State='Running'" -ErrorAction Stop

foreach ($service in $data) {

Write-Verbose "Processing service $($service.name)"

$hash=@{

Computername=$data[0].Systemname

Name=$service.name

Displayname=$service.DisplayName

}

#get the associated process

Write-Verbose "Getting process for $($service.name)"

$process=Get-WMIObject -class Win32\_Process -computername $Computer -Filter "ProcessID='$($service.processid)'" -ErrorAction Stop

$hash.Add("ProcessName",$process.name)

$hash.add("VMSize",$process.VirtualSize)

$hash.Add("PeakPageFile",$process.PeakPageFileUsage)

$hash.add("ThreadCount",$process.Threadcount)

#create a custom object from the hash table

$obj=New-Object -TypeName PSObject -Property $hash

#add a type name to the custom object

$obj.PSObject.TypeNames.Insert(0,'MOL.ServiceProcessInfo')

Write-Output $obj

} #foreach service

}

Catch {

#create an error message

$msg="Failed to get service data from $computer. $($\_.Exception.Message)"

Write-Error $msg

if ($LogErrors) {

Write-Verbose "Logging errors to $errorlog"

$computer | Out-File -FilePath $Errorlog -append

}

}

} #foreach computer

} #process

End {

Write-Verbose "Ending Get-ServiceInfo"

}

}

#Define some aliases for the functions

New-Alias -Name gcd -Value Get-ComputerData

New-Alias -Name gvi -Value Get-VolumeInfo

New-Alias -Name gsi -Value Get-ServiceInfo

#Export the functions and aliases

Export-ModuleMember -Function \* -Alias \*

* + 1. Lab C

Create a module manifest for the PSHTools that loads the module and custom format files. Test the module following these steps:

1. Import the module
2. Use Get-Command to view the module commands
3. Run help for each of your aliases
4. Run each command alias using localhost as the computername and verify formatting
5. Remove the module
6. Are the commands and variables gone?

Here is a sample manifest:

#

# Module manifest for module 'PSHTools'

#

# Generated by: Don Jones & Jeff Hicks

#

@{

# Script module or binary module file associated with this manifest.

RootModule = '.\PSHTools.psm1'

# Version number of this module.

ModuleVersion = '1.0'

# ID used to uniquely identify this module

GUID = '67afb568-1807-418e-af35-a296a43b6002'

# Author of this module

Author = 'Don Jones & Jeff Hicks'

# Company or vendor of this module

CompanyName = 'Month ofLunches'

# Copyright statement for this module

Copyright = '(c)2012 Don Jones and Jeffery Hicks'

# Description of the functionality provided by this module

Description = 'Chapter 13 Module for Month of Lunches'

# Minimum version of the Windows PowerShell engine required by this module

PowerShellVersion = '3.0'

# Name of the Windows PowerShell host required by this module

# PowerShellHostName = ''

# Minimum version of the Windows PowerShell host required by this module

# PowerShellHostVersion = ''

# Minimum version of the .NET Framework required by this module

# DotNetFrameworkVersion = ''

# Minimum version of the common language runtime (CLR) required by this module

# CLRVersion = ''

# Processor architecture (None, X86, Amd64) required by this module

# ProcessorArchitecture = ''

# Modules that must be imported into the global environment prior to importing this module

# RequiredModules = @()

# Assemblies that must be loaded prior to importing this module

# RequiredAssemblies = @()

# Script files (.ps1) that are run in the caller's environment prior to importing this module.

# ScriptsToProcess = @()

# Type files (.ps1xml) to be loaded when importing this module

# TypesToProcess = @()

# Format files (.ps1xml) to be loaded when importing this module

FormatsToProcess = '.\PSHTools.format.ps1xml'

# Modules to import as nested modules of the module specified in RootModule/ModuleToProcess

# NestedModules = @()

# Functions to export from this module

FunctionsToExport = '\*'

# Cmdlets to export from this module

CmdletsToExport = '\*'

# Variables to export from this module

VariablesToExport = '\*'

# Aliases to export from this module

AliasesToExport = '\*'

# List of all modules packaged with this module.

# ModuleList = @()

# List of all files packaged with this module

# FileList = @()

# Private data to pass to the module specified in RootModule/ModuleToProcess

# PrivateData = ''

# HelpInfo URI of this module

# HelpInfoURI = ''

# Default prefix for commands exported from this module. Override the default prefix using Import-Module -Prefix.

# DefaultCommandPrefix = ''

}