Chapter 8 Lab

In this chapter we’re going to build on the functions you created in the last chapter using the concepts you hopefully picked up today. As you work through these labs, add verbose messages to display key steps or progress information.

* + 1. Lab A

Modify your advanced function from Chapter 7 Lab A to accept pipeline input for the –ComputerName parameter. Also, add verbose input that will display the name of each computer contacted. Include code to verify that that the –ComputerName parameter will not accept a null or empty value. Test the function by adding 'localhost' | <function-name> -verbose to the end of your script. The output should look something like this:

VERBOSE: Starting Get-Computerdata

VERBOSE: Getting data from localhost

VERBOSE: Win32\_Computersystem

VERBOSE: Win32\_Bios

VERBOSE: Win32\_OperatingSystem

Workgroup :

Manufacturer : innotek GmbH

Computername : CLIENT2

Version : 6.1.7601

Model : VirtualBox

AdminPassword : NA

ServicePackMajorVersion : 1

BIOSSerial : 0

VERBOSE: Ending Get-Computerdata

Here is a possible solution

Function Get-ComputerData {

[cmdletbinding()]

param(

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

[ValidateNotNullorEmpty()]

[string[]]$ComputerName

)

Begin {

Write-Verbose "Starting Get-Computerdata"

}

Process {

foreach ($computer in $computerName) {

Write-Verbose "Getting data from $computer"

Write-Verbose "Win32\_Computersystem"

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

#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

}

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

New-Object -TypeName PSObject -Property $hash

} #foreach

} #process

End {

Write-Verbose "Ending Get-Computerdata"

}

}

"localhost" | Get-Computerdata -verbose

* + 1. Lab B

Modify your advanced function from Chapter 7 Lab B to accept pipeline input for the –ComputerName parameter. Add verbose output that will display the name of each computer contacted. Ensure that the –ComputerName parameter will not accept a null or empty value. Test the function by adding 'localhost' | <function-name> -verbose to the end of your script. The output should look something like this:

VERBOSE: Starting Get-VolumeInfo

VERBOSE: Getting volume data from localhost

VERBOSE: Procssing volume \\?\Volume{8130d5f3-8e9b-11de-b460-806e6f6e6963}\

FreeSpace Drive Computername Size

--------- ----- ------------ ----

0.07 \\?\Volume{8130d5f3... CLIENT2 0.10

VERBOSE: Procssing volume C:\Temp\

9.78 C:\Temp\ CLIENT2 10.00

VERBOSE: Procssing volume C:\

2.72 C:\ CLIENT2 19.90

VERBOSE: Procssing volume D:\

2.72 D:\ CLIENT2 4.00

VERBOSE: Ending Get-VolumeInfo

Here is a sample solution:

Function Get-VolumeInfo {

[cmdletbinding()]

param(

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

[ValidateNotNullorEmpty()]

[string[]]$ComputerName

)

Begin {

Write-Verbose "Starting Get-VolumeInfo"

}

Process {

foreach ($computer in $computerName) {

Write-Verbose "Getting volume data from $computer"

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

Foreach ($drive in $data) {

Write-Verbose "Procssing 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

New-Object -TypeName PSObject -Property $hash

} #foreach

#clear $data for next computer

Remove-Variable -Name data

} #foreach computer

} #Process

End {

Write-Verbose "Ending Get-VolumeInfo"

}

}

"localhost" | Get-VolumeInfo -verbose

* + 1. Lab C

Modify your advanced function from Lab C in Chapter 7 to accept pipeline input for the –ComputerName parameter. Add verbose output that will display the name of each computer contacted, and the name of each service queried. Ensure that the –ComputerName parameter will not accept a null or empty value. Test the function by running 'localhost' | <function-name> -verbose. The output for two services should look something like this:

VERBOSE: Starting Get-ServiceInfo

VERBOSE: Getting services from localhost

VERBOSE: Processing service AudioEndpointBuilder

Computername : CLIENT2

ThreadCount : 13

ProcessName : svchost.exe

Name : AudioEndpointBuilder

VMSize : 172224512

PeakPageFile : 83112

Displayname : Windows Audio Endpoint Builder

Here is a sample solution:

Function Get-ServiceInfo {

[cmdletbinding()]

param(

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

[ValidateNotNullorEmpty()]

[string[]]$ComputerName

)

Begin {

Write-Verbose "Starting Get-ServiceInfo"

}

Process {

foreach ($computer in $computerName) {

Write-Verbose "Getting services from $computer"

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

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

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

$hash.Add("ProcessName",$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

New-Object -TypeName PSObject -Property $hash

} #foreach service

} #foreach computer

} #process

End {

Write-Verbose "Ending Get-ServiceInfo"

}

}

"localhost" | Get-ServiceInfo -verbose

* + 1. Standalone Lab

Use this script as your starting point:

function Get-SystemInfo {

[CmdletBinding()]

param(

[string[]]$ComputerName

)

PROCESS {

foreach ($computer in $computerName) {

$os = Get-WmiObject -class Win32\_OperatingSystem `

-computerName $computer

$cs = Get-WmiObject -class Win32\_ComputerSystem `

-computerName $computer

$props = @{'ComputerName'=$computer;

'LastBootTime'=($os.ConvertToDateTime($os.LastBootupTime));

'OSVersion'=$os.version;

'Manufacturer'=$cs.manufacturer;

'Model'=$cs.model}

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

Write-Output $obj

}

}

}

Modify this function to accept pipeline input for the –ComputerName parameter. Add verbose output that will display the name of each computer contacted. Ensure that the –ComputerName parameter will not accept a null or empty value. Test the script by adding this line to the end of the script file:

'localhost','localhost' | Get-SystemInfo -verbose

The output for should look something like this:

VERBOSE: Getting WMI data from localhost

Model : VirtualBox

ComputerName : localhost

Manufacturer : innotek GmbH

LastBootTime : 6/19/2012 8:55:34 AM

OSVersion : 6.1.7601

Here is a sample solution:

function Get-SystemInfo {

[CmdletBinding()]

param(

[Parameter(Mandatory=$True,ValueFromPipeline=$True)]

[ValidateNotNullOrEmpty()]

[string[]]$ComputerName

)

PROCESS {

foreach ($computer in $computerName) {

Write-Verbose "Getting WMI data from $computer"

$os = Get-WmiObject -class Win32\_OperatingSystem -computerName $computer

$cs = Get-WmiObject -class Win32\_ComputerSystem -computerName $computer

$props = @{'ComputerName'=$computer

'LastBootTime'=($os.ConvertToDateTime($os.LastBootupTime))

'OSVersion'=$os.version

'Manufacturer'=$cs.manufacturer

'Model'=$cs.model

}

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

Write-Output $obj

}

}

}

'localhost','localhost' | Get-SystemInfo -verbose