Chapter 7 Lab

Using your design notes from the previous chapter, start building your tools. You won’t have to address every single design point right now. We’ll revise and expand these functions a bit more in the next few chapters. For this chapter your functions should complete without error, even if they are only using temporary output.

* + 1. Lab A

Using your notes from Lab A in Chapter 6, write an advanced function that accepts one or more computer names. For each computer name, use CIM or WMI to query the specified information. For now, keep each property’s name, using ServicePackMajorVersion, Version, SerialNumber, etc. But go ahead and “translate” the value for AdminPasswordStatus to the appropriate text equivalent.

Test the function by adding <function-name> -computerName localhost to the bottom of your script, and then running the script. The output for a single computer should look something like this:

Workgroup :

Manufacturer : innotek GmbH

Computername : CLIENT2

Version : 6.1.7601

Model : VirtualBox

AdminPassword : NA

ServicePackMajorVersion : 1

BIOSSerial : 0

It is possible that some values may be empty.

Here is a possible solution:

Function Get-ComputerData {

[cmdletbinding()]

param( [string[]]$ComputerName )

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

}

Get-Computerdata -computername localhost

* + 1. Lab B

Using your notes for Lab B from Chapter 6, write an advanced function that accepts one or more computer names. For each computer name, use CIM or WMI to query the specified information. Format the Size and FreeSpace property values in GB to 2 decimal points. Test the function by adding <function-name> -computerName localhost to the bottom of your script, and then running the script. The output for a single service should look something like this:

FreeSpace Drive Computername Size

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

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

9.78 C:\Temp\ CLIENT2 10.00

2.72 C:\ CLIENT2 19.90

2.72 D:\ CLIENT2 4.00

Here is a possible solution:

Function Get-VolumeInfo {

[cmdletbinding()]

param( [string[]]$ComputerName )

foreach ($computer in $computerName) {

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

Foreach ($drive in $data) {

#format size and freespace in GB to 2 decimal points

$Size="{0:N2}" -f ($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

}

Get-VolumeInfo -ComputerName localhost

* + 1. Lab C

Using your notes for Lab C from Chapter 6, write an advanced function that accepts one or more computer names. For each computer name, use CIM or WMI to query all instances of Win32\_Service where the State property is “Running.” For each service, get the ProcessID property. Then query the matching instance of the Win32\_Process class – that is, the instance with the same ProcessID. Write a custom object to the pipeline that includes the service name and display name, the computer name, and the process name, ID, virtual size, peak page file usage, and thread count. Test the function by adding <function-name> -computerName localhost to the end of the script.

The output for a single service should look something like this:

Computername : CLIENT2

ThreadCount : 52

ProcessName : svchost.exe

Name : wuauserv

VMSize : 499138560

PeakPageFile : 247680

Displayname : Windows Update

Here is a possible solution:

Function Get-ServiceInfo {

[cmdletbinding()]

param( [string[]]$ComputerName )

foreach ($computer in $computerName) {

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

foreach ($service in $data) {

$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

}

Get-ServiceInfo -ComputerName localhost

* + 1. Standalone Lab

If time is limited, you can skip the 3 labs above and work on this single, stand-alone lab. Write an advanced function named Get-SystemInfo. This function should accept one or more computer names via a –ComputerName parameter. It should then use WMI or CIM to query the Win32\_OperatingSystem class and Win32\_ComputerSystem class for each computer. For each computer queried, display the last boot time (in a standard date/time format), the computer name, and operating system version (all from Win32\_OperatingSystem). Also, display the manufacturer and model (from Win32\_ComputerSystem). You should end up with a single object with all of this information for each computer.

Note that the last boot time property does not contain a human-readable date/time value; you will need to use the class’ ConvertToDateTime() method to convert that value to a normal-looking date/time. Test the function by adding Get-SystemInfo -computerName localhost to the end of the script.

You should get a result like this:

Model : VirtualBox

ComputerName : localhost

Manufacturer : innotek GmbH

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

OSVersion : 6.1.7601

Here is a possible solution:

function Get-SystemInfo {

[CmdletBinding()]

param(

[string[]]$ComputerName

)

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

}

}

Get-SystemInfo -ComputerName localhost