Chapter 6 Lab

In this lab, we aren’t going to have you write any actual scripts or functions. Instead, we want you to think about the design aspect, something many people overlook. Let’s say you’ve been asked to develop the following PowerShell tools. Even though the tool will be running from PowerShell 3.0, you don’t have to assume that any remote computer is running PowerShell 3.0. Assume at least PowerShell v2.

* + 1. Lab A

Design a command that will retrieve the following information from one or more remote computers, using the indicated WMI classes and properties:

* Win32\_ComputerSystem:
  + Workgroup
  + AdminPasswordStatus; display the numeric values of this property as text strings.
    - For 1, display Disabled
    - For 2, display Enabled
    - For 3, display NA
    - For 4, display Unknown
  + Model
  + Manufacturer
* From Win32\_BIOS
  + SerialNumber
* From Win32\_OperatingSystem
  + Version
  + ServicePackMajorVersion

Your function’s output should also include each computer’s name.

Ensure that your function’s design includes a way to log errors to a text file, allowing the user to specify an error file name but defaulting to C:\Errors.txt. Also plan ahead to create a custom view so that your function always outputs a table, using the following column headers:

• ComputerName

• Workgroup

• AdminPassword (for AdminPasswordStatus in Win32\_ComputerSystem)

• Model

• Manufacturer

• BIOSSerial (for SerialNumber in Win32\_BIOS)

• OSVersion (for Version in Win32\_OperatingSystem)

• SPVersion (for ServicePackMajorVersion in Win32\_OperatingSystem)

Again, you aren’t writing the script only outlining what you might do..

* + 1. Lab B

Design a tool that will retrieve the WMI Win32\_Volume class from one or more remote computers. For each computer and volume, the function should output the computer’s name, the volume name (such as C:\), and the volume’s free space and size in GB (using no more than 2 decimal places). Only include volumes that represent fixed hard drives – do not include optical or network drives in the output. Keep in mind that any given computer may have multiple hard disks; your function’s output should include one object for each disk.

Ensure that your function’s design includes a way to log errors to a text file, allowing the user to specify an error file name but defaulting to C:\Errors.txt. Also, plan to create a custom view in the future so that your function always outputs a table, using the following column headers:

• ComputerName

• Drive

• FreeSpace

• Size

* + 1. Lab C

Design a command that will retrieve all running services on one or more remote computers. This command will offer the option to log the names of failed computers to a text file. It will produce a list that includes each running service’s name and display name, along with information about the process that represents each running service. That information will include the process name, virtual memory size, peak page file usage, and thread count. However, peak page file usage and thread count will not display by default.

For each tool, think about the following design questions:

* What would be a good name for your tool.
* What sort of information do you need for each tool? (These might be potential parameters)
* How do you think the tool would be run from a command prompt or what type of data will it write to the pipeline??

Answers

**Lab A**

Because we are getting information from a variety of WMI sources, a good function name might be Get-ComputerData. We’ll need a string parameter for the name, a string for the log file and maybe a switch parameter indicating that we want to log data. The function will need to make several WMI queries and then it can write a custom object to the pipeline. We can get the computername from one of the WMI classes. We could use the computername parameter, but by using something from WMI we’ll get the “official” computer name which is better if we test with something like localhost.

Since the AdminStatus property value an integer we can use a Switch statement to define a variable with the interpretation as a string.

When creating a custom object, especially one where we need to make sure property names will match the eventual custom view, a hash table will come in handy because we can use it with New-Object.

We can probably start out by having the function take computer names as parameters:

Get-Computerdata –computername server01,server02

But eventually we’ll want to be able to pipe computernames to it. Each computername should produce a custom object.

**Lab B**

Since the command will get volume data information, a likely name would be Get-VolumeInfo or Get-VolumeData. Like Lab A we’ll need a string parameter for a computername, as well as a parameter for the eventlog and a switch to indicate whether or not to log errors. A sample command might look like:

Get-VolumeInfo –computername Server01 –ErrorLog C:\work\errors.txt –LogError

Also like Lab A, using a hash table with the new properties will make it easier to create and write a custom object to the pipeline. We’ll also need to convert the size and free space by dividing the size in bytes by 1GB. One way to handle the formatting requirement is to use the –f operator.

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

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

**Lab C**

This lab can follow the same outline as the first two in terms of computername, error log name and whether or not to log files. Because we need to get the process id of each service, we’ll need to use WMI or CIM. The Get-Service cmdlet returns a service object, but it doesn’t include the process id. Once we have the service object we can execute another WMI query to get the process object.

It will most likely be easiest to create a hash table with all of the required properties from the 2 WMI classes. For now, we’ll include all the properties. Later we can create a custom view with only the desired, default properties.

Since this function is getting service information, a good name might be Get-ServiceInfo.