Chapter 9 Lab

These labs will build on what you’ve already created, applying new concepts from this chapter.

* + 1. Lab A

Add comment-based help to your advanced function from Lab A in Chapter 8. Include at least a synopsis, description, and help for the –ComputerName parameter. Test your help by adding help <function-name> to the end of your script.

Here is a possible solution:

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.

.EXAMPLE

PS C:\> Get-ComputerData Server01

Run the command and query Server01.

.EXAMPLE

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

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

to the command.

#>

[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"

}

}

help Get-Computerdata -full

* + 1. Lab B

Add comment-based help to your advanced function from Lab B in Chapter 8. Include at least a synopsis, description, and help for the –ComputerName parameter. Test your help by adding help <function-name> to the end of your script.

Here is a possible solution:

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.

.EXAMPLE

PS C:\> Get-VolumeInfo Server01

Run the command and query Server01.

.EXAMPLE

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

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

to the command.

#>

[cmdletbinding()]

param(

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

[ValidateNotNullorEmpty()]

[string[]]$ComputerName

)

Begin {

Write-Verbose "Starting Get-VolumeInfo"

}

Process {

foreach ($computer in $computerName) {

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

Foreach ($drive in $data) {

#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"

}

}

help Get-VolumeInfo -full

* + 1. Lab C

Add comment-based help to your advanced function from Lab C in Chapter 8. Include at least a synopsis, description, and help for the –ComputerName parameter. Test your help by adding help <function-name> to the end of your script.

Here is a possible solution:

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.

.EXAMPLE

PS C:\> Get-ServiceInfo Server01

Run the command and query Server01.

.EXAMPLE

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

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

to the command.

#>

[cmdletbinding()]

param(

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

[ValidateNotNullorEmpty()]

[string[]]$ComputerName

)

Begin {

Write-Verbose "Starting Get-ServiceInfo"

}

Process {

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

} #process

End {

Write-Verbose "Ending Get-ServiceInfo"

}

}

help Get-ServiceInfo -full

* + 1. Standalone Lab

Using the script in Listing 9.2 add comment-based help.

List 9.2 Standalone lab starting point

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

}

}

}

Include at least a synopsis, description, and help for the –ComputerName parameter. Test your help by adding help <function-name> to the end of your script.

Here is a possible solution:

function Get-SystemInfo {

<#

.SYNOPSIS

Gets critical system info from one or more computers.

.DESCRIPTION

This command uses WMI, and can accept computer names, CNAME aliases,

and IP addresses. WMI must be enabled and you must run this

with admin rights for any remote computer.

.PARAMETER Computername

One or more names or IP addresses to query.

.EXAMPLE

Get-SystemInfo -computername localhost

#>

[CmdletBinding()]

param(

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

[ValidateNotNullOrEmpty()]

[string[]]$ComputerName

)

PROCESS {

foreach ($computer in $computerName) {

WWrite-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

}

}

}

help Get-SystemInfo