Chapter 10 Lab

You are going to continue with the functions you’ve been building the last few chapters. The next step is to begin incorporating some error handling using Try/Catch/Finally. If you haven’t done so, take a few minutes to read the help content on Try/Catch/Finally. For any changes you make, don’t forget to update your comment-based help.

* + 1. Lab A

Using Lab A from Chapter 9, add a –ErrorLog parameter to your advanced function, which accepts a filename for an error log and defaults to C:\Errors.txt. When the function is run with this parameter, failed computer names should be appended to the error log file.

Next, if the first WMI query fails, the function should output nothing for that computer and should not attempt a second or third WMI query. Write an error to the pipeline containing each failed computer name.

Test all of this by adding this line <function-name> -ComputerName localhost,NOTONLINE –verbose to the end of your script. A portion of 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

SerialNumber : 0

Model : VirtualBox

AdminPassword : NA

ServicePackMajorVersion : 1

VERBOSE: Getting data from notonline

VERBOSE: Win32\_Computersystem

Get-Computerdata : Failed getting system information from notonline. The RPC server is

unavailable. (Exception from HRESULT: 0x800706BA)

At S:\Toolmaking\Ch10-LabA.ps1:115 char:40

+ 'localhost','notonline','localhost' | Get-Computerdata -logerrors -verbose

+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

+ CategoryInfo : NotSpecified: (:) [Write-Error], WriteErrorException

+ FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorException,Get-Comp

uterData

VERBOSE: Getting data from localhost

Here is a sample 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.

.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

New-Object -TypeName PSObject -Property $hash

#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"

}

}

'localhost','notonline','localhost' | Get-Computerdata -verbose

* + 1. Lab B

Using Lab B from Chapter 9, add a –ErrorLog parameter to your advanced function, which accepts a filename for an error log and defaults to C:\Errors.txt. When the function is run with this parameter, failed computer names should be appended to the error log file.

Test all of this by adding this line <function-name> -ComputerName localhost,NOTONLINE –verbose to the end of your script. A portion of the output should look something like this:

VERBOSE: Starting Get-VolumeInfo

VERBOSE: Getting data from localhost

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

VERBOSE: Getting data from NotOnline

Get-VolumeInfo : Failed to get volume information from NotOnline. The RPC server is

unavailable. (Exception from HRESULT: 0x800706BA)

At S:\Toolmaking\Ch10-LabB.ps1:96 char:27

+ 'localhost','NotOnline' | Get-VolumeInfo -Verbose -logerrors

+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

+ CategoryInfo : NotSpecified: (:) [Write-Error], WriteErrorException

+ FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorException,Get-Volu

meInfo

VERBOSE: Logging errors to C:\Errors.txt

VERBOSE: Ending Get-VolumeInfo

Here is a sample 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.

.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

New-Object -TypeName PSObject -Property $hash

} #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"

}

}

'localhost','NotOnline' | Get-VolumeInfo -Verbose

* + 1. Lab C

Using Lab C from Chapter 9, add a –LogErrors switch parameter to your advanced function. Also add a –ErrorFile parameter, which accepts a filename for an error log and defaults to C:\Errors.txt. When the function is run with the -LogErrors parameter, failed computer names should be appended to the error log file. .Also, if –LogErrors is used, the log file should be deleted at the start of the function if it exists, so that each time the command starts with a fresh log file.

Test all of this by adding this line <function-name> -ComputerName localhost,NOTONLINE –verbose –logerrors to the end of your script. A portion of the output should look something like this:

VERBOSE: Processing service wuauserv

VERBOSE: Getting process for wuauserv

Computername : CLIENT2

ThreadCount : 45

ProcessName : svchost.exe

Name : wuauserv

VMSize : 499363840

PeakPageFile : 247680

Displayname : Windows Update

VERBOSE: Getting services from NOTOnline

Get-ServiceInfo : Failed to get service data from NOTOnline. The RPC server is

unavailable. (Exception from HRESULT: 0x800706BA)

At S:\Toolmaking\Ch10-LabC.ps1:109 char:39

+ "localhost","NOTOnline","localhost" | Get-ServiceInfo -logerrors -verbose

+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

+ CategoryInfo : NotSpecified: (:) [Write-Error], WriteErrorException

+ FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorException,Get-Serv

iceInfo

VERBOSE: Logging errors to C:\Errors.txt

VERBOSE: Getting services from localhost

VERBOSE: Processing service AudioEndpointBuilder

VERBOSE: Getting process for AudioEndpointBuilder

Here is a sample 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.

.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

New-Object -TypeName PSObject -Property $hash

} #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"

}

}

Get-ServiceInfo -ComputerName "localhost","NOTOnline","localhost" -logerrors

* + 1. Standalone Lab

Use the code in Listing 10.4 as a starting point.

Listing 10.4 Standalone lab starting point

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

}

}

}

Add a –LogErrors switch to this advanced function. When the function is run with this switch, failed computer names should be logged to C:\Errors.txt. This file should be deleted at the start of the function each time it is run, so that it starts out fresh each time. If the first WMI query fails, the function should output nothing for that computer and should not attempt a second WMI query. Write an error to the pipeline containing each failed computer name.

Test your script by adding this line to the end of your script.

Get-SystemInfo -computername localhost,NOTONLINE,localhost -logerrors

A portion of the output should look something like this:

Model : VirtualBox

ComputerName : localhost

Manufacturer : innotek GmbH

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

OSVersion : 6.1.7601

Get-SystemInfo : NOTONLINE failed

At S:\Toolmaking\Ch10-Standalone.ps1:51 char:1

+ Get-SystemInfo -computername localhost,NOTONLINE,localhost -logerrors

+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

+ CategoryInfo : NotSpecified: (:) [Write-Error], WriteErrorException

+ FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorException,Get-Syst

emInfo

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 {

<#

.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,

[switch]$logErrors

)

BEGIN {

if (Test-Path c:\errors.txt) {

del c:\errors.txt

}

}

PROCESS {

foreach ($computer in $computerName) {

WWrite-Verbose "Getting WMI data from $computer"

try {

$continue = $true

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

} catch {

$continue = $false

$computer | Out-File c:\errors.txt -append

Write-Error "$computer failed"

}

if ($continue) {

$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,NOTONLINE,localhost -logerrors