Chapter 11 Lab

We’re sure you’ll have plenty of practice debugging your own scripts. But we want to reinforce some of the concepts from this chapter and get you used to following a procedure. Never try to debug a script simply by staring at it, hoping the error will jump out at you. It might, but more than likely it may not be the only one. Follow our guidelines to identify bugs. Fix one thing at a time. If it doesn’t resolve the problem, change it back and repeat the process.

The functions listed here are broken and buggy. We’ve numbered each line for reference purposes; the numbers are not part of the actual function. How would you debug them? Revise them into working solutions. Remember, you will need to dot source the script each time you make a change. We recommend testing in the regular PowerShell console.

The function in Listing 11.8 is supposed to display some properties of running services sorted by the service account.

Listing 11.8 A broken function

1 Function Get-ServiceInfo {

2 [cmdletbinding()]

3 Param([string]$Computername)

4 $services=Get-WmiObject -Class Win32\_Services -filter "state='Running" `

-computername $computernam

5 Write-Host "Found ($services.count) on $computername" –Foreground Green

6 $sevices | sort -Property startname,name Select -property `

startname,name,startmode,computername

7 }

## Commentary

The first step is to clean up the formatting a bit to make it easier to read. We’re also going to delete the backticks so that one line commands show as a single line. Ideally, you are doing this is in a scripting editor or the ISE; something that will show you line numbers. Here’s our first pass:

01 Function Get-ServiceInfo {

02

03 [cmdletbinding()]

04 Param([string]$Computername)

05

06 $services=Get-WmiObject -Class Win32\_Services -filter "state='Running" -computername $computernam

07

08 Write-Host "Found ($services.count) on $computername" -ForegroundColor Green

09

10 $sevices | sort -Property startname,name Select -property startname,name,startmode,computername

11

12 }

This is short enough that problems might jump out at you, but we’re going to dot source the script and try to run the function.

S:\Toolmaking> get-serviceinfo client2

Get-WmiObject : Cannot validate argument on parameter 'ComputerName'. The argument is null or empty. Supply an

argument that is not null or empty and then try the command again.

At S:\Toolmaking\a.ps1:6 char:86

+ ... -computername $computernam

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

+ CategoryInfo : InvalidData: (:) [Get-WmiObject], ParameterBindingValidationException

+ FullyQualifiedErrorId : ParameterArgumentValidationError,Microsoft.PowerShell.Commands.GetWmiObjectCommand

Found (.count) on client2

Sort-Object : Cannot bind parameter because parameter 'Property' is specified more than once. To provide multiple

values to parameters that can accept multiple values, use the array syntax. For example, "-parameter

value1,value2,value3".

At S:\Toolmaking\a.ps1:10 char:50

+ $sevices | sort -Property startname,name Select -property startname,name,startm ...

+ ~~~~~~~~~

+ CategoryInfo : InvalidArgument: (:) [Sort-Object], ParameterBindingException

+ FullyQualifiedErrorId : ParameterAlreadyBound,Microsoft.PowerShell.Commands.SortObjectCommand

So, a number of issues. Take them in order. The first problem is related to the Get-WmiObject expression. It appears there is a problem with the computer name parameter on line 6.

Get-WmiObject : Cannot validate argument on parameter 'ComputerName'. The argument is null or empty. Supply an

argument that is not null or empty and then try the command again.

At S:\Toolmaking\a.ps1:6 char:86

+ ... -computername $computernam

And sure enough we can see here and in the script that the variable is misspelled. It should be $computername. We’ll make this one change, and repeat the test.

S:\Toolmaking> get-serviceinfo client2

Get-WmiObject : Invalid query "select \* from Win32\_Services where state='Running"

At S:\Toolmaking\a.ps1:6 char:11

+ $services=Get-WmiObject -Class Win32\_Services -filter "state='Running" -computer ...

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

+ CategoryInfo : InvalidArgument: (:) [Get-WmiObject], ManagementException

+ FullyQualifiedErrorId : GetWMIManagementException,Microsoft.PowerShell.Commands.GetWmiObjectCommand

Found (.count) on client2

Sort-Object : Cannot bind parameter because parameter 'Property' is specified more than once. To provide multiple

values to parameters that can accept multiple values, use the array syntax. For example, "-parameter

value1,value2,value3".

At S:\Toolmaking\a.ps1:10 char:50

+ $sevices | sort -Property startname,name Select -property startname,name,startm ...

+ ~~~~~~~~~

+ CategoryInfo : InvalidArgument: (:) [Sort-Object], ParameterBindingException

+ FullyQualifiedErrorId : ParameterAlreadyBound,Microsoft.PowerShell.Commands.SortObjectCommand

Well, we at least confirmed that we fixed the first problem. But now we see a new problem. This is where we start backing off and trying parts of our command. In the ISE we can select the first part of the Get-WmiObject command and test it out. Or we can also do it at the prompt.

S:\Toolmaking> Get-WmiObject -Class Win32\_Services

Get-WmiObject : Invalid class "Win32\_Services"

At line:1 char:1

+ Get-WmiObject -Class Win32\_Services

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

+ CategoryInfo : InvalidType: (:) [Get-WmiObject], ManagementException

+ FullyQualifiedErrorId : GetWMIManagementException,Microsoft.PowerShell.Commands.GetWmiObjectCommand

Well that’s a problem. Now it is possible for a WMI class to exist only on a given machine based on an installed application or product. We’re testing this on Windows 7. The first step is to research this problem and either verify this is the right class name or correct it. In this situation, a quick Internet search for Win32\_Services shows that this it should be Win32\_Service. Back to the script, make one change and try again.

S:\Toolmaking> get-serviceinfo client2

Get-WmiObject : Invalid query "select \* from Win32\_Service where state='Running"

At S:\Toolmaking\a.ps1:6 char:11

+ $services=Get-WmiObject -Class Win32\_Service -filter "state='Running" -computern ...

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

+ CategoryInfo : InvalidArgument: (:) [Get-WmiObject], ManagementException

+ FullyQualifiedErrorId : GetWMIManagementException,Microsoft.PowerShell.Commands.GetWmiObjectCommand

Found (.count) on client2

Sort-Object : Cannot bind parameter because parameter 'Property' is specified more than once. To provide multiple

values to parameters that can accept multiple values, use the array syntax. For example, "-parameter

value1,value2,value3".

At S:\Toolmaking\a.ps1:10 char:50

+ $sevices | sort -Property startname,name Select -property startname,name,startm ...

+ ~~~~~~~~~

+ CategoryInfo : InvalidArgument: (:) [Sort-Object], ParameterBindingException

+ FullyQualifiedErrorId : ParameterAlreadyBound,Microsoft.PowerShell.Commands.SortObjectCommand

Looks there is still a problem. The only part left to the command that we haven’t looked at is the filter. One thing we might want to do is use the WBEMTest utility and verify our query outside of PowerShell. From the prompt, type WBEMTEST and press Enter. Click Connect and verify you are connecting to the root\cimv2 namespace. Click connect again to connect to the local machine. Click the Query button. The error message shows the query. Grab everything inside the “”

select \* from Win32\_Service where state='Running

and paste it into the query Window. Click Apply and you’ll get the invalid query message. You probably noticed already that the query is missing the closing quote. Modify the query in WBEMTest:

select \* from Win32\_Service where state='Running'

and try again. Success! Now we know what to fix in the script. Rinse and repeat.

S:\Toolmaking> get-serviceinfo client2

Found (\\CLIENT2\root\cimv2:Win32\_Service.Name="AeLookupSvc" \\CLIENT2\root\cimv2:Win32\_Service.Name="AudioEndpointBuild

er" \\CLIENT2\root\cimv2:Win32\_Service.Name="BFE" \\CLIENT2\root\cimv2:Win32\_Service.Name="BITS" \\CLIENT2\root\cimv2:Wi

n32\_Service.Name="Browser" \\CLIENT2\root\cimv2:Win32\_Service.Name="CertPropSvc" \\CLIENT2\root\cimv2:Win32\_Service.Name

="CryptSvc" \\CLIENT2\root\cimv2:Win32\_Service.Name="DcomLaunch" \\CLIENT2\root\cimv2:Win32\_Service.Name="Dhcp" [\\CLIENT](file:///\\CLIENT)

…

Whoa!! That’s not what we expected. There’s no line number but we can see it displayed in green font and it starts with Found so this must be the result from Line 8:

08 Write-Host "Found ($services.count) on $computername" -ForegroundColor Green

The problem is the ($services.count) sub expression. It is missing a $. Line 8 should probably look like this:

08 Write-Host "Found $($services.count) on $computername" -ForegroundColor Green

Once more from the top:

S:\Toolmaking> get-serviceinfo client2

Found 65 on client2

Sort-Object : Cannot bind parameter because parameter 'Property' is specified more than once. To provide multiple

values to parameters that can accept multiple values, use the array syntax. For example, "-parameter

value1,value2,value3".

At S:\Toolmaking\a.ps1:10 char:50

+ $sevices | sort -Property startname,name Select -property startname,name,startm ...

+ ~~~~~~~~~

+ CategoryInfo : InvalidArgument: (:) [Sort-Object], ParameterBindingException

+ FullyQualifiedErrorId : ParameterAlreadyBound,Microsoft.PowerShell.Commands.SortObjectCommand

Better. Now there is a problem on line 10.

10 $sevices | sort -**Property** startname,name Select –**property** startname,name,startmode,computername

PowerShell is complaining we are using –Property more than once. Which we are as you can see from the boldfaced listing; once for Sort-Object and again for Select-Object. Hold the phone. The function is supposed to sort and then pipe to Select-Object but there’s no pipe character!

10 $sevices | sort -**Property** startname,name **|** Select –**property** startname,name,startmode,computername

We’ll make that change and test again.

S:\Toolmaking> get-serviceinfo client2

Found 65 on client2

Interesting. No errors but also no results. There are 65 services so we should have gotten something. Line 10 is supposed to take all the services, sort them and then select a few properties. We know what line is problematic so let’s try a breakpoint

PS S:\Toolmaking> Set-PSBreakpoint -Script .\a.ps1 -Line 10

We’ll also include a new line with Write-Debug so we can check our variable.

07 Write-Debug "got services"

Running the function automatically puts us in debug mode.

PS S:\Toolmaking> Get-ServiceInfo client2

Hit Variable breakpoint on 'S:\Toolmaking\a.ps1:$services' (Write access)

At S:\Toolmaking\a.ps1:6 char:1

+ $services=Get-WmiObject -Class Win32\_Service -filter "state='Running'" -computer ...

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

PS S:\Toolmaking>>

At the nested prompt you can type ? to get help.

PS S:\Toolmaking>> ?

s, stepInto Single step (step into functions, scripts, etc.)

v, stepOver Step to next statement (step over functions, scripts, etc.)

o, stepOut Step out of the current function, script, etc.

c, continue Continue operation

q, quit Stop operation and exit the debugger

k, Get-PSCallStack Display call stack

l, list List source code for the current script.

Use "list" to start from the current line, "list <m>"

to start from line <m>, and "list <m> <n>" to list <n>

lines starting from line <m>

<enter> Repeat last command if it was stepInto, stepOver or list

?, h displays this help message.

Or we can check variables.

PS S:\Toolmaking>> $services[0]

ExitCode : 0

Name : AeLookupSvc

ProcessId : 200

StartMode : Manual

State : Running

Status : OK

That variable looks ok. Press ‘c’ to continue.

PS S:\Toolmaking>> c

Found 66 on client2

Hit Line breakpoint on 'S:\Toolmaking\a.ps1:10'

At S:\Toolmaking\a.ps1:10 char:1

+ $sevices | sort -Property startname,name | Select -property startname,name,start ...

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

PS S:\Toolmaking>>

Let’s check the variable:

PS S:\Toolmaking>> $sevices

What?

PS S:\Toolmaking>> get-variable $sevices

Get-Variable : Cannot validate argument on parameter 'Name'. The argument is null. Supply a non-null argument and try

the command again.

At line:1 char:14

+ get-variable $sevices

+ ~~~~~~~~

+ CategoryInfo : InvalidData: (:) [Get-Variable], ParameterBindingValidationException

+ FullyQualifiedErrorId : ParameterArgumentValidationError,Microsoft.PowerShell.Commands.GetVariableCommand

We know $services exist so upon closer examination we see there is a typo. We’ll enter ‘q’ to quit and revise the script. We’ll also remove the breakpoints.

PS S:\Toolmaking> Get-ServiceInfo client2

Found 65 on client2

startname name startmode computername

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

LocalSystem AudioEndpointBuilder Auto

LocalSystem BITS Manual

…

So close. No errors but we’re also not getting all of the expected properties which most likely means we have the wrong name. Research again, or testing with WBEMTest shows we need should be using Systemname. There’s nothing left in the script so hopefully this is the last problem.

PS S:\Toolmaking> Get-ServiceInfo client2

Found 65 on client2

startname name startmode systemname

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

LocalSystem AudioEndpointBuilder Auto CLIENT2

LocalSystem BITS Manual CLIENT2

LocalSystem Browser Manual CLIENT2

LocalSystem CertPropSvc Manual CLIENT2

…

Finally, success. As you discovered if you stuck with us, often times fixing one problem reveals another. Take it one step and one bug at a time. Don’t change half dozen parts of your code at once. Sure, you might fix one problem but unintentionally create 2 more.

The function in Listing 11.9 is a bit more involved. It is designed to get recent event log entries for a specified log on a specified computer. Events are sorted by the event source and added to a log file. The filename is based on the date, computername and event source. At the end, the function, displays a directory listing of the logs. Hint: clean up the formatting first.

Listing 11.9 Buggy Export Function

01 Function Export-EventLogSource {

02

03 [cmdletbinding()]

04 Param (

05 [Parameter(Position=0,Mandatory=$True,Helpmessage="Enter a computername",ValueFromPipeline=$True)]

06 [string]$Computername,

07 [Parameter(Position=1,Mandatory=$True,Helpmessage="Enter a classic event log name like System")]

08 [string]$Log,

09 [int]$Newest=100

10 )

11 Begin {

12 Write-Verbose "Starting export event source function"

13 #the date format is case-sensitive"

14 $datestring=Get-Date -Format "yyyyMMdd"

15 $logpath=Join-path -Path "C:\Work" -ChildPath $datestring

16 if (! (Test-Path -path $logpath) {

17 Write-Verbose "Creating $logpath"

18 mkdir $logpath

19 }

20 Write-Verbose "Logging results to $logpath"

21 }

22 Process {

23 Write-Verbose "Getting newest $newest $log event log entries from $computername"

24 Try {

25 Write-Host $computername.ToUpper -ForegroundColor Green

26 $logs=Get-EventLog -LogName $log -Newest $Newest -Computer $Computer -ErrorAction Stop

27 if ($logs) {

28 Write-Verbose "Sorting $($logs.count) entries"

29 $log | sort Source | foreach {

30 $logfile=Join-Path -Path $logpath -ChildPath "$computername-$($\_.Source).txt"

31 $\_ | Format-List TimeWritten,MachineName,EventID,EntryType,Message |

32 Out-File -FilePath $logfile -append

33

34 #clear variables for next time

35 Remove-Variable -Name logs,logfile

36 }

37 else {Write-Warning "No logged events found for $log on $Computername"}

38 }

39 Catch { Write-Warning $\_.Exception.Message }

40 }

41 End {dir $logpath

42 Write-Verbose "Finished export event source function"

43 }

44 }

## Commentary

This is a much more complicated example. Ideally you won’t format your code so poorly. But you might find a script on the Internet that you want to try out, in a test environment of course. So learning how to reformat is a good skill. Here’s our revised script with line numbers.

01 Function Export-EventLogSource {

02

03 [cmdletbinding()]

04

05 Param (

06 [Parameter(Position=0,Mandatory=$True,Helpmessage="Enter a computername",ValueFromPipeline=$True)]

07 [string]$Computername,

08 [Parameter(Position=1,Mandatory=$True,Helpmessage="Enter a classic event log name like System")]

09 [string]$Log,

10 [int]$Newest=100

11 )

12

13 Begin {

14 Write-Verbose "Starting export event source function"

15

16 #the date format is case-sensitive"

17 $datestring=Get-Date -Format "yyyyMMdd"

18 $logpath=Join-path -Path "C:\Work" -ChildPath $datestring

19

20 if (! (Test-Path -path $logpath) {

21 Write-Verbose "Creating $logpath"

22 mkdir $logpath

23 }

24

25 Write-Verbose "Logging results to $logpath"

26

27 }

28

29 Process {

30 Write-Verbose "Getting newest $newest $log event log entries from $computername"

31

32 Try {

33 Write-Host $computername.ToUpper -ForegroundColor Green

34 $logs=Get-EventLog -LogName $log -Newest $Newest -Computer $Computer -ErrorAction Stop

35 if ($logs) {

36 Write-Verbose "Sorting $($logs.count) entries"

37 $log | sort Source | foreach {

38 $logfile=Join-Path -Path $logpath -ChildPath "$computername-$($\_.Source).txt"

39 $\_ | Format-List TimeWritten,MachineName,EventID,EntryType,Message | Out-File -FilePath $logfile -append

40

41 #clear variables for next time

42 Remove-Variable -Name logs,logfile

43 }

44 else {

45 Write-Warning "No logged events found for $log on $Computername"

46 }

47 }

48 Catch {

49 Write-Warning $\_.Exception.Message

50 }

51 }

52

53 End {

54 dir $logpath

55 Write-Verbose "Finished export event source function"

56 }

57 }

First, off you can save yourself some time by using a script editor or the ISE that includes syntax highlighting. If so, you can fix some problems pretty easily. Look at Figure 1.

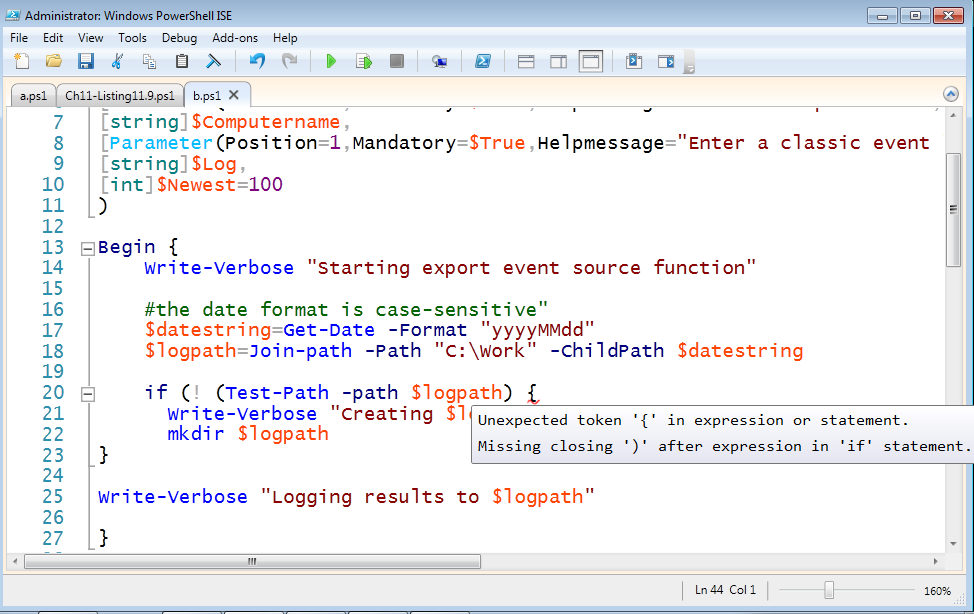


Figure 1

The ISE tells you exactly what the potential problem is. As you scroll through the rest of the file you’ll also see a message that the Try statement is missing its Catch or Finally script block.

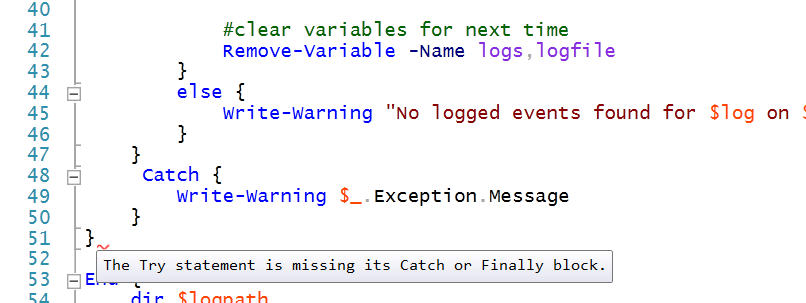


Figure 2

As you can see in Figure 2 there is a Catch block so most likely we are missing a closing curly brace. Notice the – boxes to the left? These indicate collapsible sections. We can scroll up, collapsing sections as we go along and making sure that we are collapsing code between a set of curly braces. Other script editors might have other ways of matching braces. Figure 3 shows where we end up.

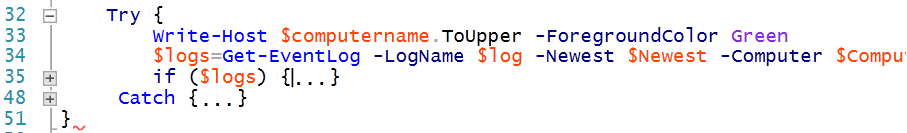


Figure 3

Interesting. What happened to the Else scriptblock? It seems to have disappeared into the If scriptblock which is wrong. Let’s expand it.

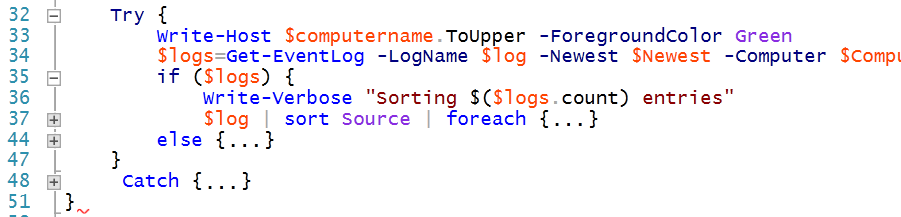


Figure 4

The If script block must be using the brace after the Else scriptblock which leaves nothing for the Try block to use. The only thing we haven’t look at is the foreach. Perhaps it is missing a closing curly brace. We’ll put one in at line 40 and see that the red squiggle is gone. We can also start at the bottom and collapse each section and it looks good. But all of this was the easy part. We still don’t know what problems there are when we run it.

PS S:\Toolmaking> Export-EventLogSource -Computername client2 -Log System -Newest 10

Directory: C:\Work

Mode LastWriteTime Length Name

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

d---- 5/31/2012 2:01 PM 20120531

string ToUpper(), string ToUpper(cultureinfo culture)

WARNING: Cannot validate argument on parameter 'ComputerName'. The argument is null or empty. Supply an argument that

is not null or empty and then try the command again.

Well it started out ok. The script is supposed to create a text log for each event source in the new directory and then display the files. We can see that the folder was created so we got at least as far as Line 22. Because the function has verbose messages, let’s run it again and see if we can narrow down where the first problem begins.

PS S:\Toolmaking> Export-EventLogSource -Computername client2 -Log System -Newest 10 -verbose

VERBOSE: Starting export event source function

VERBOSE: Logging results to C:\Work\20120531

VERBOSE: Getting newest 10 System event log entries from client2

string ToUpper(), string ToUpper(cultureinfo culture)

WARNING: Cannot validate argument on parameter 'ComputerName'. The argument is null or empty. Supply an argument that

is not null or empty and then try the command again.

VERBOSE: Finished export event source function

Ok. Just after the verbose line that says what we’re doing is some odd line about string toUpper(). Searching the script we find these lines:

30 Write-Verbose "Getting newest $newest $log event log entries from $computername"

31

32 Try {

33 Write-Host $computername.ToUpper -ForegroundColor Green

The script is trying to call the ToUpper method so that the computer name is in upper case. But we need to include () when calling a method. Line 33 should be:

33 Write-Host $computername.ToUpper() -ForegroundColor Green

Remember: one change at a time. Let’s try again.

PS S:\Toolmaking> Export-EventLogSource -Computername client2 -Log System -Newest 10 -verbose

VERBOSE: Starting export event source function

VERBOSE: Logging results to C:\Work\20120531

VERBOSE: Getting newest 10 System event log entries from client2

CLIENT2

WARNING: Cannot validate argument on parameter 'ComputerName'. The argument is null or empty. Supply an argument that

is not null or empty and then try the command again.

VERBOSE: Finished export event source function

We corrected the first problem. Now there is a parameter problem. We know line 33 was the last thing successfully run so we might need to take a look at line 34.

33 Write-Host $computername.ToUpper -ForegroundColor Green

34 $logs=Get-EventLog -LogName $log -Newest $Newest -Computer $Computer -ErrorAction Stop

Sure enough there is a –Computername parameter, even though the function is using a shortened version. That’s legal. And the value is $Computer. That is most likely the culprit. The error message says that it can’t validate it because it is null or empty. Well, we just wrote the variable in upper case so we know it has a value. Oh. We used $computername on line 33 and on line 34 we are using $computer. Those are two different variables. Most likely it should be $computername. We also could have used a breakpoint to step through the script.

PS S:\Toolmaking> Export-EventLogSource -Computername client2 -Log System -Newest 10 -verbose

VERBOSE: Starting export event source function

VERBOSE: Logging results to C:\Work\20120531

VERBOSE: Getting newest 10 System event log entries from client2

CLIENT2

VERBOSE: Sorting 10 entries

Directory: C:\Work\20120531

Mode LastWriteTime Length Name

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

-a--- 5/31/2012 2:17 PM 18 client2-.txt

VERBOSE: Finished export event source function

Well, no more errors but we didn’t get the expected results either. Each log is supposed to be sorted and content added to a text file.

38 $logfile=Join-Path -Path $logpath -ChildPath "$computername-$($\_.Source).txt"

39 $\_ | Format-List TimeWritten,MachineName,EventID,EntryType,Message | Out-File -FilePath $logfile -append

Line 38 is defining the text file which is supposed to be comprised of the computername and the source. From the output we can see the file is created in the right location but the name is wrong and it is missing the correct content:

PS S:\Toolmaking> cat C:\work\20120531\client2-.txt

System

One thing we might do is manually run the code that is getting the logs and verify there are values for the Source property.

PS S:\Toolmaking> get-eventlog -LogName system -ComputerName client2 -newest 10 | Select source

Source

------

Service Control Manager

Service Control Manager

Service Control Manager

Service Control Manager

Service Control Manager

EventLog

Service Control Manager

Service Control Manager

Service Control Manager

Service Control Manager

Ok. The command is good and this means we should have had 2 files created. We can tell from our previous output that these lines of code are good:

34 $logs=Get-EventLog -LogName $log -Newest $Newest -Computer $Computer -ErrorAction Stop

35 if ($logs) {

36 Write-Verbose "Sorting $($logs.count) entries"

What we’re not getting is a proper file created so these lines are suspect:

37 $log | sort Source | foreach {

38 $logfile=Join-Path -Path $logpath -ChildPath "$computername-$($\_.Source).txt"

39 $\_ | Format-List TimeWritten,MachineName,EventID,EntryType,Message | Out-File -FilePath $logfile -append

It looks like line 37 is sorting all of the event logs and then doing something with each on in ForEach. But the script isnt’ piping $logs it is piping $log. A simple typo but it makes sense. Because $log didn’t exist, at least as an event log object, there was no source property which is why our file name was incorrect. We’ll fix and retry.

PS S:\Toolmaking> Export-EventLogSource -Computername client2 -Log System -Newest 10 -verbose

VERBOSE: Starting export event source function

VERBOSE: Logging results to C:\Work\20120531

VERBOSE: Getting newest 10 System event log entries from client2

CLIENT2

VERBOSE: Sorting 10 entries

Directory: C:\Work\20120531

Mode LastWriteTime Length Name

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

-a--- 5/31/2012 2:35 PM 18 client2-.txt

-a--- 5/31/2012 2:44 PM 364 client2-EventLog.txt

-a--- 5/31/2012 2:44 PM 3898 client2-Service Control Manager.txt

VERBOSE: Finished export event source function

That looks good. The function is supposed to write the time written, the computername, the event id, entry type and message to the file. Checking one of the files:

PS S:\Toolmaking> get-content C:\work\20120531\client2-EventLog.txt

TimeWritten : 5/31/2012 12:00:05 PM

MachineName : CLIENT2.jdhlab.local

EventID : 6013

EntryType : Information

Message : The system uptime is 90460 seconds.

This is what we expect so the script has been cleaned up and debugged.