



source: <http://www.MSSQLTips.com/tip.asp?id=3509> -- printed: 3/4/2015 9:20:02 PM

# Error handling with try-catch-finally in PowerShell for SQL Server

Written By: Diana Moldovan -- 2/16/2015

## Problem

PowerShell V2 introduces the "try-catch-finally" statements, similar to those you already use when you write .NET code. "Try-catch-finally" encloses a block of script that is likely to produce errors under specific circumstances and therefore helps you to keep your code organized. Below you'll find a short usage guide for this error handling construct. This was done using Windows 8.1 Pro x64 / PowerShell v4 / SQL Server 2012 SP1 environment.

## Solution

Open a new PowerShell session and make sure that the [SQLPS module](#) is not imported. If you run [Get-Module](#), SQLPS should not be in the result list. For example, if you use the Windows PowerShell ISE:

```
Get-Module
```

```
1 Get-Module
```

```
PS C:\windows\System32\WindowsPowerShell\v1.0> Get-Module

ModuleType Version      Name                                ExportedCommands
-----
Script      1.0.0.0      ISE                                {Get-Isesnippet, I
Manifest    3.1.0.0      Microsoft.PowerShell.Management   {Add-Computer, Add
Manifest    3.1.0.0      Microsoft.PowerShell.Utility      {Add-Member, Add-T
```

Compare this result with the one you obtain after importing SQLPS:

```
Import-Module SQLPS -DisableNameChecking
Get-Module
```

```
PS SQLSERVER:\> Get-Module

ModuleType Version      Name                                ExportedCommands
-----
Script      1.0.0.0      ISE                                {Get-Isesnippet,
Manifest    3.1.0.0      Microsoft.PowerShell.Management   {Add-Computer, A
Manifest    3.1.0.0      Microsoft.PowerShell.Utility      {Add-Member, Add
Manifest    1.0          SQLASCMDLETS |                    {Add-RoleMember,
Script      0.0          Sqlps                             {Add-SqlAvailabi
Manifest    1.0          SQLPS                             {Add-SqlAvailabi
```

Now when we run this piece of code - remember that SQLPS is not loaded.

```
Set-Location SQLSERVER:\SQL\MyServer\DEFAULT\DATABASES
Write-Host -ForegroundColor Green "Done"
```

The result should look like:

```

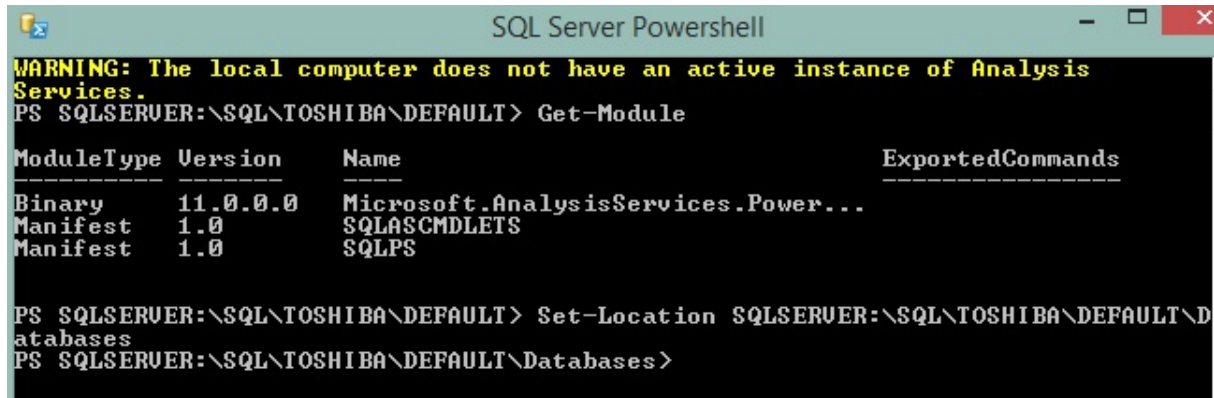
PS C:\windows\system32\windowspowershell\v1.0> Set-Location SQLSERVER:\SQL\Toshiba\DEFAULT\DATABASES
Write-Host -ForegroundColor Green "Done"
Set-Location : Cannot find drive. A drive with the name 'SQLSERVER' does not exist.
At line:1 char:1
+ Set-Location SQLSERVER:\SQL\Toshiba\DEFAULT\DATABASES
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (SQLSERVER:String) [Set-Location], DriveNotFoundException
+ FullyQualifiedErrorId : DriveNotFound,Microsoft.PowerShell.Commands.SetLocationCommand

Done

```

Since SQLPS is not loaded, PowerShell returns a "drive not found" exception. Notice that this error is a [non-terminating error](#) and the "Done" message will be part of the result.

You won't be able to reproduce this situation when running PowerShell from within SQL Server Management Studio, because the SQL Server specific cmdlets and provider are already loaded on start. Right click, for example, on the Server node in the object view pane, click on "Start PowerShell" and run Get-Module.



```

SQL Server Powershell
WARNING: The local computer does not have an active instance of Analysis Services.
PS SQLSERVER:\SQL\TOSHIBA\DEFAULT> Get-Module

ModuleType Version Name ExportedCommands
-----
Binary 11.0.0.0 Microsoft.AnalysisServices.Power...
Manifest 1.0 SQLASCMDLETS
Manifest 1.0 SQLPS

PS SQLSERVER:\SQL\TOSHIBA\DEFAULT> Set-Location SQLSERVER:\SQL\TOSHIBA\DEFAULT\D
atabases
PS SQLSERVER:\SQL\TOSHIBA\DEFAULT\Databases>

```

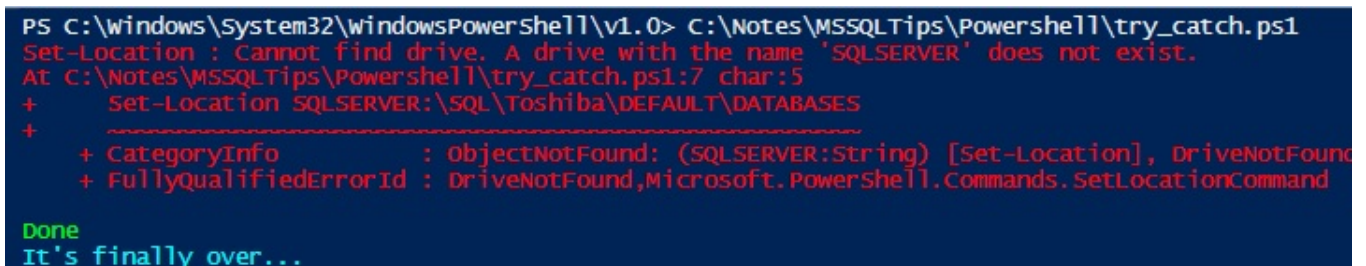
Now let's enclose the above error prone line of code in a "try-catch" as follows:

```

try{
    Set-Location SQLSERVER:\SQL\MyServer\DEFAULT\DATABASES
    Write-Host -ForegroundColor Green "Done"
}
catch{
    Write-Host -ForegroundColor DarkYellow "You're WRONG"
    Write-Host -ForegroundColor Magenta $Error[0].Exception
}
finally{
    Write-Host -ForegroundColor Cyan "It's finally over..."
}

```

As you can see, the result is similar to what you've obtained before, except the last line which comes from the "finally" line of code. The "catch" code is not executed.



```

PS C:\windows\system32\windowspowershell\v1.0> C:\Notes\MSSQLTips\PowerShell\try_catch.ps1
Set-Location : Cannot find drive. A drive with the name 'SQLSERVER' does not exist.
At C:\Notes\MSSQLTips\PowerShell\try_catch.ps1:7 char:5
+ Set-Location SQLSERVER:\SQL\Toshiba\DEFAULT\DATABASES
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (SQLSERVER:String) [Set-Location], DriveNotFoundException
+ FullyQualifiedErrorId : DriveNotFound,Microsoft.PowerShell.Commands.SetLocationCommand

Done
It's finally over...

```

For a non-terminating error, adding a "try-catch" construct has no effect unless you add an "error action" parameter or preference set to "stop", forcing it to become a terminating error as shown below:

```

try{
    Set-Location SQLSERVER:\SQL\MyServer\DEFAULT\DATABASES -ErrorAction Stop
    Write-Host -ForegroundColor Green "Done"
}
catch{
    Write-Host -ForegroundColor DarkYellow "You're WRONG"
}

```

```

    Write-Host -ForegroundColor Magenta $Error[0].Exception
}
finally{
    Write-Host -ForegroundColor Cyan "It's finally over..."
}

```

```

PS C:\windows\System32\WindowsPowerShell\v1.0> C:\Notes\MSSQLTips\PowerShell\try_catch_all.ps1
You're WRONG
System.Management.Automation.DriveNotFoundException: Cannot find drive. A drive with the name 'SQL'
at System.Management.Automation.SessionStateInternal.GetDrive(String name, Boolean automount)
at System.Management.Automation.SessionStateInternal.GetDrive(String name, Boolean automount)
at System.Management.Automation.SessionStateInternal.SetLocation(String path, CmdletProviderContext)
at Microsoft.PowerShell.Commands.SetLocationCommand.ProcessRecord()
It's finally over...

```

When it encounters the terminating error, PowerShell writes the error to the [\\$Error array](#) and transfers the execution to the "catch" block, if any. The "catch" block contains the error-handling code. If there is no "catch" block, PowerShell simply writes the error to the error stream. The "finally" block is optional. It will execute after both the "try" and "catch" have completed, regardless of the occurrence of any error. The non-terminating error example result contains the "It's finally over" message even if the execution was not transferred to the "catch" block. Use "finally" to perform clean-up tasks such as deleting temporary output files you no more need. You can't use a "try" block alone; you need one "catch" block or one "finally" block to run the code.

It is a good practice to write several "catch" blocks to catch specific exceptions. You should place the most specific blocks first, and end with a "catch all" block. PowerShell will search the "catch" blocks from the top to the bottom and stops when it finds a match. Use `$Error[0].Exception.GetType()` to find the type of the exception you are dealing with. In the following example the exception type is written in bright yellow. After the "catch" code, PowerShell will execute the "finally" block and will display the light blue message.

```

try{
    Set-Location SQLSERVER:\SQL\MyServer\DEFAULT\DATABASES -ErrorAction Stop
    Write-Host -ForegroundColor Green "Done"
}
catch [System.Management.Automation.DriveNotFoundException]{
    Write-Host -ForegroundColor DarkYellow "You're WRONG. Here is why:"
    Write-Host -ForegroundColor Yellow $Error[0].Exception.GetType()
    Write-Host -ForegroundColor Magenta $Error[0].Exception
}
catch{
    Write-Host -ForegroundColor DarkYellow "You're WRONG"
    Write-Host -ForegroundColor Yellow "General Exception"
}
finally{
    Write-Host -ForegroundColor Cyan "It's finally over..."
}

```

```

PS C:\windows\System32\WindowsPowerShell\v1.0> C:\Notes\MSSQLTips\PowerShell\try_catch.ps1
You're WRONG. Here is why:
System.Management.Automation.DriveNotFoundException
System.Management.Automation.DriveNotFoundException: Cannot find drive. A drive with the name 'SQL'
at System.Management.Automation.SessionStateInternal.GetDrive(String name, Boolean automount)
at System.Management.Automation.SessionStateInternal.GetDrive(String name, Boolean automount)
at System.Management.Automation.SessionStateInternal.SetLocation(String path, CmdletProviderContext)
at Microsoft.PowerShell.Commands.SetLocationCommand.ProcessRecord()
It's finally over...

```

## Next Steps

- Review the [PowerShell section on MSSQLTips.com](#) for more SQL Server related PowerShell tips and tricks.
- Explore this [library of free PowerShell books](#). "The Big Book of PowerShell Error Handling" is a complete error handling guide, which offers useful details on dealing with terminating and non-terminating errors.

### Follow

Get Free SQL  
Tips

### Learning

DBAs  
Developers

### Resources

Tutorials  
Webcasts

### Search

Tip Categories  
Search By TipID

### Community

First Timer?  
Pictures

### More Info

Join  
About

[Twitter](#)[BI Professionals](#)[Whitepapers](#)[Top Ten](#)[Free T-shirt](#)[Copyright](#)[LinkedIn](#)[Careers](#)[Tools](#)[Authors](#)[Contribute](#)[Privacy](#)[Google+](#)[Q and A](#)[Events](#)[Disclaimer](#)[Facebook](#)[Today's Tip](#)[User Groups](#)[Feedback](#)[Pinterest](#)[Author of the  
Year](#)[Advertise](#)[RSS](#)

---

Copyright (c) 2006-2015 [Edgewood Solutions, LLC](#) All rights reserved

Some names and products listed are the registered trademarks of their respective owners.