Writing advanced PowerShell functions

mark wilkinson - @m82labs - mark@m82labs.com - https://m82labs.com

Who is Mark Wilkinson?

(the guy talking and waving his hands around)

- Father of 4
- Lives in Raleigh NC
- Loves:
  - Beer
  - Performance Tuning
  - Monitoring
  - PowerShell, Python, and Go
- DBA @ ChannelAdvisor
- Mentor @ speakingmentors.com

The agenda

- A basic function
- What is an advanced function?
  - Parameter features
  - Function features
  - Output streams
- Demos!

Git Repository

https://github.com/m82labs/AdvancedPowerShellFunction.git

A simple function

A simple function

```
function Get-SqlVersion {
    param(
        [string]$Instance
    )
    Write-Host "Getting SQL Server Version: " -NoNewLine
    try {
        $Version = Invoke-SqlCmd -Query 'SELECT @@VERSION' `
                                 -ServerInstance $Instance `
                                 -ErrorAction Stop
        Write-Host "done" -ForegroundColor Green
        Write-Host "$($Instance) - $($Version)"
    }
    catch {
        Write-Host "failed - $($_.Exception.Message)" -ForegroundColor Red
        return
    }
}
```

What is an advanced function?

An advanced function (super impressive)

```
function Get-SqlVersion {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True)]
        [string]$Instance
    )
    Write-Host "Getting SQL Server Version: " -NoNewLine
    try {
        $Version = Invoke-SqlCmd -Query 'SELECT @@VERSION' `
                                 -ServerInstance $Instance `
                                 -ErrorAction Stop
        Write-Host "done" -ForegroundColor Green
        Write-Host "$($Instance) - $($Version)"
    }
    catch {
        Write-Host "failed - $($_.Exception.Message)" -ForegroundColor Red
        return
    }
}
```

That’s all. Any questions?

Thanks for coming!

mark wilkinson - @m82labs - mark@m82labs.com - https://m82labs.com

Parameter features

Mandatory

```
function Get-SqlVersion {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True)]
        [string]$Instance
    )
    ...
}
```

Validation Set

```
function Get-SqlVersion {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True)]
        [string]$Instance,
        [ValidateSet(2012,2014,2016,2017)]
        [int]$CurrentMajorVersion
    )
    ...
}
```

Validation Range

```
function Get-SqlVersion {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True)]
        [string]$Instance,
        [ValidateRange(2012,2017)]
        [int]$CurrentMajorVersion
    )
    ...
}
```

Aliases

```
function Get-SqlVersion {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True)]
        [Alias('Instance')]
        [string]$SqlInstance,
        [ValidSet(2012,2014,2016,2017)]
        [int]$CurrentMajorVersion
    )
    ...
}
```

Pipeline input

```
function Get-SqlVersion {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True,ValueFromPipeline=$True)]
        [string]$Instance,
        [ValidSet(2012,2014,2016,2017)]
        [int]$CurrentMajorVersion
    )

    Begin {}

    Process {
        ...
    }

    End {}
}
```

Pipeline input continued

- `ValueFromPipeline=$True`
- `Begin {}`
  - Whatever is in this code block is executed before anything else happens
    - Print a start message
    - Set global variables
- `Process {}`
  - This block executes for every input object
  - Reuses the variable name
- `End {}`
  - Executes once at the end, when Process has completed
    - Clean up temp resources
    - Print a message

Pipeline input continued

- `ValueFromPipelineByPropertyName =$True`
- Allows you to configure multiple parameters to take input from the pipeline
- Uses property names to assign variables, extra properties discarded
- Can be harder to keep track of and use, but can be useful in some cases
- Can work in conjunction with aliases for ease of use

Dynamic parameters

```
function Get-SqlVersion {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True,ValueFromPipeline=$True)]
        [string]$Instance,
        [ValidSet(2012,2014,2016,2017)]
        [int]$CurrentMajorVersion
    )

    DynamicParam {
        ...
    }

    Begin {}

    Process {}

    End {}
}
```

Function features

WhatIf and Confirm

```
function Remove-Database {
    [CmdletBinding(SupportsShouldProcess=$True,ConfirmImpact='Low')]
    param(
        ...
    )

    if ($pscmdlet.ShouldProcess("$SqlInstance:$Database", "Drop Database")){
        Write-Host "Dropping database $($Database): " -NoNewline
        try {
            Invoke-SqlCmd -ServerInstance $SqlInstance -Query "DROP DATABASE $($Database);"
            Write-Host "done" -ForegroundColor Green
        } except {
            Write-Host "failed - $($_.Exception.Message)" -ForegroundColor Red
            return
        }
    }
}
```

WhatIf and confirm continued

- Enable with: `SupportsShouldProcess=$True,ConfirmImpact='Low'`
- Example:

```
if ($pscmdlet.ShouldProcess("Thing affected", "Action taken")) {
    [some code]
}
```

- `-WhatIf` will output:

```
What if: Performing the operation "Action taken" on target "Thing affected"
```

- `-Confirm` (or if `$ConfirmPreference` < `ConfirmImpact`)will output:

```
Are you sure you want to perform this action?
Performing the operation "test" on target "test".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Yes"):
```

Output streams

DO NOT CROSS THE STREAMS

mark wilkinson - @m82labs - mark@m82labs.com - https://m82labs.com

Verbose and Debug

```
[string]$VersionQuery = @"
SELECT  @@SERVERNAME As ser,
        SERVERPROPERTY('productversion') AS ver,
        CASE
          WHEN @@VERSION LIKE '%Windows%' THEN 'Windows'
          ELSE 'Linux'
        END AS platform
WHERE   CAST(SERVERPROPERTY('productmajorversion') AS VARCHAR(2)) LIKE '%$($CurrentMajor)%'
        AND @@VERSION LIKE '%$($OS)%'
"@

Write-Verbose "Query:`n$($VersionQuery)"
Write-Debug -Message "Query:`n$($VersionQuery)"
```

Verbose and Debug Notes

- `Write-Verbose`, activated with `-Verbose`
  - Outputs:

```
VERBOSE: [message]
```

- `Write-Debug`, activated with `-Debug`
  - Outputs:

```
DEBUG: [message]
Confirm
Continue with this operation?
[Y] Yes [A] Yes to All [H] Halt Command [S] Suspend [?] Help (default is "Yes"):
```

Now go contribute to dbatools!

More resources

- https://dbatools.io/ - Great module with tons of useful functions
- #powershellhelp @ sqlcommunity Slack - Great community
- PowerShell Core on GitHub - Cross-platform version of PowerShell
- PowerShell Team Blog - Great place to find news and interesting projects

Contact Info

Twitter	@m82labs
Blog	https://m82labs.com
Email	mark@m82labs.com
Slack	@mark.w-m82labs
GitHub	m82labs

Sign up for SQL Community Slack: https://dbatools.io/slack/