

# TEC US 2011: PowerShell Deep Dive: Jeffrey Snover – Proxy Functions

---

These are the scripts Richard was using. You can also find his:

- Slides and video here: <http://dmitrysotnikov.wordpress.com/2011/09/22/video-jeffrey-snovor-proxy-functions/>

## Get-Help

### Readme.txt

This contains four script files that make up the demo:

- 1: Creating the proxy function.ps1 - Few lines of script that show you how to get the stub for you proxy function
- 2: Get-HelpProxyBeginning.ps1 - Output from the commands in the script above. It's the stub for the Get-Help proxy function that we will edit
- 3: Get-Help.ps1 - The implemented proxy function with comments showing all the changes that were made from Get-HelpProxyBeginning.ps1
- 4: Using the proxy function.ps1 - A few lines that shows importing and using the newly created proxy function

The proxy function executes a search on the TechNet Wiki. I've created articles there for CommandNotFoundException so you will actually get a search result for the script in "Using the proxy function.ps1"

## 1 - Creating the proxy function.ps1

```
# Get the command
$Cmd = Get-Command Get-Help

# Now get it's metadata
$CmdMetadata = New-Object System.Management.Automation.CommandMetaData $Cmd

# Create our proxy function
[System.Management.Automation.ProxyCommand]::Create($CmdMetadata) | Out-File
GetHelpProxyCommand.ps1

# Now we can edit it to do whatever we want
```

## 2 - Get-HelpProxyBeginning.ps1

```
[CmdletBinding(DefaultParameterSetName='AllUsersView')]
param(
    [Parameter(Position=0, ValueFromPipelineByPropertyName=$true)]
    [string]
    ${Name},

    [string]
    ${Path},

    [string[]]
    ${Category},
```

```

[string[]]
${Component},

[string[]]
${Functionality},

[string[]]
${Role},

[Parameter(ParameterSetName='DetailedView')]
[switch]
${Detailed},

[Parameter(ParameterSetName='AllUsersView')]
[switch]
${Full},

[Parameter(ParameterSetName='Examples')]
[switch]
${Examples},

[Parameter(ParameterSetName='Parameters')]
[string]
${Parameter},

[switch]
${Online})

begin
{
    try {
        $outBuffer = $null
        if ($PSBoundParameters.TryGetValue('OutBuffer', [ref]$outBuffer))
        {
            $PSBoundParameters['OutBuffer'] = 1
        }
        $wrappedCmd = $ExecutionContext.InvokeCommand.GetCommand('Get-Help',
[System.Management.Automation.CommandTypes]::Cmdlet)
        $scriptCmd = {& $wrappedCmd @PSBoundParameters }
        $steppablePipeline = $scriptCmd.GetSteppablePipeline($myInvocation.CommandOrigin)
        $steppablePipeline.Begin($PSCmdlet)
    } catch {
        throw
    }
}

process
{
    try {
        $steppablePipeline.Process($_)
    } catch {
        throw
    }
}

end
{
    try {
        $steppablePipeline.End()
    }
}

```

```

    } catch {
        throw
    }
}
<#

.ForwardHelpTargetName Get-Help
.ForwardHelpCategory Cmdlet

#>

```

### 3 - Get-Help.ps1

```

function Get-Help
{
# Many changes made to the parameters to create a new parameter set that only has the
ErrorRecord parameter
[CmdletBinding(DefaultParameterSetName='AllUsersView')]
param(
    [Parameter(Position=0,
ValueFromPipelineByPropertyName=$true, ParameterSetName='DetailedView')]
    [Parameter(Position=0,
ValueFromPipelineByPropertyName=$true, ParameterSetName='AllUsersView')]
    [Parameter(Position=0,
ValueFromPipelineByPropertyName=$true, ParameterSetName='Examples')]
    [Parameter(Position=0,
ValueFromPipelineByPropertyName=$true, ParameterSetName='Parameters')]
    [string]
    ${Name},

    [Parameter(ParameterSetName='DetailedView')]
    [Parameter(ParameterSetName='AllUsersView')]
    [Parameter(ParameterSetName='Examples')]
    [Parameter(ParameterSetName='Parameters')]
    [string]
    ${Path},

    [Parameter(ParameterSetName='DetailedView')]
    [Parameter(ParameterSetName='AllUsersView')]
    [Parameter(ParameterSetName='Examples')]
    [Parameter(ParameterSetName='Parameters')]
    [string[]]
    ${Category},

    [Parameter(ParameterSetName='DetailedView')]
    [Parameter(ParameterSetName='AllUsersView')]
    [Parameter(ParameterSetName='Examples')]
    [Parameter(ParameterSetName='Parameters')]
    [string[]]
    ${Component},

    [Parameter(ParameterSetName='DetailedView')]
    [Parameter(ParameterSetName='AllUsersView')]
    [Parameter(ParameterSetName='Examples')]
    [Parameter(ParameterSetName='Parameters')]
    [string[]]
    ${Functionality},

```

```

[Parameter (ParameterSetName='DetailedView')]
[Parameter (ParameterSetName='AllUsersView')]
[Parameter (ParameterSetName='Examples')]
[Parameter (ParameterSetName='Parameters')]
[string[]]
${Role},

[Parameter (ParameterSetName='DetailedView')]
[switch]
${Detailed},

[Parameter (ParameterSetName='AllUsersView')]
[switch]
${Full},

[Parameter (ParameterSetName='Examples')]
[switch]
${Examples},

[Parameter (ParameterSetName='Parameters')]
[string]
${Parameter},

[Parameter (ParameterSetName='Error', Position=0)]
[System.Management.Automation.ErrorRecord]
${ErrorRecord},

[Parameter (ParameterSetName='MSType', Position=0)]
${MSType},

[Parameter (ParameterSetName='Type', Position=0)]
${Type},

[Parameter (ParameterSetName='DetailedView')]
[Parameter (ParameterSetName='AllUsersView')]
[Parameter (ParameterSetName='Examples')]
[Parameter (ParameterSetName='Parameters')]
[switch]
${Online})

begin
{
    try {
        $outBuffer = $null
        if ($PSBoundParameters.TryGetValue('OutBuffer', [ref]$outBuffer))
        {
            $PSBoundParameters['OutBuffer'] = 1
        }

        # Add the fully qualified name for the old Get-Help command
        $wrappedCmd =
$ExecutionContext.InvokeCommand.GetCommand('Microsoft.PowerShell.Core\Get-Help',
[System.Management.Automation.CommandTypes]::Cmdlet)

        # Our new Error record parameter was specified
        if ($PSBoundParameters['ErrorRecord'])
        {
            # Import System.Web for the HttpUtility to do URL Encoding
            [System.Reflection.Assembly]::LoadWithPartialName("System.Web") | Out-Null

```

```

        # Generate the search url (search URL + encoded FullyQualifiedErrorId)
        $url =
"http://social.technet.microsoft.com/wiki/search/SearchResults.aspx?q=" +
[System.Web.HttpUtility]::UrlEncode($ErrorRecord.FullyQualifiedErrorId)

        # Launch it in IE (you'd probably want to check for default browser, etc...)
        $scriptCmd = {& Start-Process -FilePath "C:\Program Files (x86)\Internet
Explorer\iexplore.exe" -ArgumentList $url}
    }
    elseif($PSBoundParameters['MSType'])
    {
        # Import System.Web for the HttpUtility to do URL Encoding
        [System.Reflection.Assembly]::LoadWithPartialName("System.Web") | Out-Null

        if ($Type -is [System.String])
        {
            $url = "http://msdn.microsoft.com/en-us/library/{0}_members(VS.85).aspx"
-f $MSType
        }else
        {
            $url = "http://msdn.microsoft.com/en-us/library/{0}_members(VS.85).aspx"
-f $MSType.GetType().FullName
        }
        # Launch it in IE (you'd probably want to check for default browser, etc...)
        $scriptCmd = {& Start-Process -FilePath "C:\Program Files (x86)\Internet
Explorer\iexplore.exe" -ArgumentList $url}
    }
    elseif($PSBoundParameters['Type'])
    {
        # Import System.Web for the HttpUtility to do URL Encoding
        [System.Reflection.Assembly]::LoadWithPartialName("System.Web") | Out-Null

        if ($Type -is [System.String])
        {
            $url = "http://www.bing.com/search?q={0}+members" -f $Type
        }else
        {
            $url = "http://www.bing.com/search?q={0}+members" -f
$Type.GetType().FullName
        }
        # Launch it in IE (you'd probably want to check for default browser, etc...)
        $scriptCmd = {& Start-Process -FilePath "C:\Program Files (x86)\Internet
Explorer\iexplore.exe" -ArgumentList $url}
    }
    else # Our record parameter was not provided. Fall back to the original Get-Help
    {
        $scriptCmd = {& $wrappedCmd @PSBoundParameters }
    }

    $steppablePipeline = $scriptCmd.GetSteppablePipeline($myInvocation.CommandOrigin)
    $steppablePipeline.Begin($PSCmdlet)
} catch {
    throw
}
}

process
{

```

```

    try {
        $steppablePipeline.Process($_)
    } catch {
        throw
    }
}

end
{
    try {
        $steppablePipeline.End()
    } catch {
        throw
    }
}
}
}
<#

.ForwardHelpTargetName Get-Help
.ForwardHelpCategory Cmdlet

#>

```

## 4 - Using the proxy function.ps1

```

# Let's look at the syntax of the Get-Help command
Get-Command Get-Help -Syntax

# Dot source the proxy function
. .\Get-Help.ps1

# Our proxy function has replaced Get-Help and it will be called by default
Get-Command Get-Help -Syntax

# You can still get to the original one by specifying the fully qualified name
Microsoft.PowerShell.Core\Get-Help

# Execute a bogus command
Do-SomethingReallyCool

# What does that error mean? Let's pass the error record to Get-Help
Get-Help $error[0]

# There's an article for that...

# Maybe we invoke a command with an invalid parameter
Get-Process -sdfs

Get-Help $error[0]

# There is no article for it... We should add one

```

## MetaProgramming

### MetaProgramming.psd1

```

@{
    #AliasesToExport=""

```

```

Author="NTDEV\jsnover"
CompanyName="Microsoft"
Copyright="© Microsoft. All rights reserved."
Description="MetaProgramming Module"
CLRVersion="2.0"
#FileList=""
#FormatsToProcess=""
#FunctionsToExport=""
GUID="803c4b4c-bf5f-4f0f-b518-ffdf37f9448e"
ModuleToProcess="MetaProgramming.psm1"
ModuleVersion="0.0.0.1"
PowerShellVersion="2.0"
#PrivateData=""
#RequiredAssemblies=""
#RequiredModules=""
#ScriptsToProcess=""
#TypesToProcess=""
#VariablesToExport=""
}

```

## MetaProgramming.psm1

```

<#
.Synopsis
    Creates a PowerShell ParameterAttribute.
.Description
    This command is used in metaprogramming scenarios where you dynamically
    generate a cmdlet on the fly. Parameters of Cmdlets must have a Parameter
    Attribute attached to them to tell PowerShell what to do with the parameter.
    This command generates that attribute.
.Parameter Position
    What position to map this parameter to when users don't use named parameters.
.Parameter ParameterSetName
    Name of the parameterSet
.Parameter Mandatory
    Is this parameter Mandatory
.Parameter ValueFromPipeline
    Does this parameter take a value from the pipeline object
.Parameter ValueFromPipelineByPropertyName
    Does the parameter take a value from a property of the pipeline object
.Parameter ValueFromRemainingArguments
    Does this collect the remaining arguments on the command line
.Example
    New-ParameterAttribute
.Outputs
    System.Management.Automation.ParameterAttribute
.Link
    New-ProxyCommand
.Notes
    NAME:      New-ParameterAttribute
    AUTHOR:    NTDEV\jsnover
    LASTEDIT:  1/4/2009 8:53:35 AM
#>
function New-ParameterAttribute
{
[CmdletBinding(
    SupportsShouldProcess=$False,
    SupportsTransactions=$False,
    ConfirmImpact="None",

```

```

        DefaultParameterSetName="" ]
param(
[Parameter()]
[Int32]
$Position = [int32]::MinValue,

[Parameter()]
[String]
$ParameterSetName,

[Parameter()]
[Switch]
$Mandatory,

[Parameter()]
[Switch]
$ValueFromPipeline,

[Parameter()]
[Switch]
$ValueFromPipelineByPropertyName,

[Parameter()]
[Switch]
$ValueFromRemainingArguments
)
New-Object System.Management.Automation.ParameterAttribute -Property @{
    Position = $Position
    ParameterSetName = $ParameterSetName
    Mandatory = $Mandatory
    ValueFromPipeline = $ValueFromPipeline
    ValueFromPipelineByPropertyName = $ValueFromPipelineByPropertyName
    ValueFromRemainingArguments = $ValueFromRemainingArguments
}
} # New-ParameterAttribute

#####
<#
.Synopsis
    Generate a script for a ProxyCommand to call a base Cmdlet adding or removing
parameters.
.Description
    This command generates command which calls another command (a ProxyCommand).
    In doing so, it can add additional attributes to the existing parameters.
    This is useful for things like enforcing corporate naming standards.
    It can also ADD or REMOVE parameters.  If you ADD a parameter, you'll have
to implement the semantics of that parameter in the code that gets generated.
.Parameter Name
    Name of the Cmdlet to proxy.
.Parameter CommandType
    Type of Command we are proxying.  In general you dont' need to specify this but
it becomes necessary if there is both a cmdlet and a function with the same
name
.Parameter AddParameter
    List of Parameters you would like to add. NOTE - you have to edit the resultant

```

```

code to implement the semantics of these parameters. ALSO - you need to remove
them from $PSBOUND
.Parameter RemoveParameter
.Example
New-ProxyCommand get-process -CommandType all -RemoveParameter `
FileVersionInfo,Module,ComputerName -AddParameter SortBy > c:\ps\get-myprocess.ps1

.Example
New-ProxyCommand -Name Format-Table -addParameter "IncludeIndex" > c:\ps\Format-
Table.ps1

.Outputs
System.String
.Link
New-ParameterAttribute
.Notes
NAME:      New-ProxyCommand
AUTHOR:    NTDEV\jsnover
ToDo:      Need to modify script to emit template help for the proxy command.
           Probably should add a -AsFunction switch
LASTEDIT:  1/4/2009 8:53:35 AM
#####
#Requires -Version 2.0
#>

function New-ProxyCommand
{
[CmdletBinding(
    SupportsShouldProcess=$False,
    SupportsTransactions=$False,
    ConfirmImpact="None",
    DefaultParameterSetName="")]
param(
[Parameter(Position=0, Mandatory=$True)]
[String]$Name ,

[Parameter(Position=1)]
[Alias("Type")]
[System.Management.Automation.CommandTypes]$CommandType="All" ,

[Parameter(Position=2)]
[String[]]$AddParameter ,

[Parameter(Position=3)]
[String[]]$RemoveParameter
)

$Cmd = Get-Command -Name $Name -CommandType $CommandType
if (!$cmd)
{
    Throw "No such Object [$Name : $CommandType]"
}elseif (@($cmd).Count -ne 1)
{
    Throw "Ambiguous reference [$Name : $CommandType]`n$($Cmd |Out-String)"
}
$MetaData = New-Object System.Management.Automation.CommandMetaData $cmd
if ($RemoveParameter)
{
    foreach ($p in @($RemoveParameter))

```

```

    {
        [Void]$MetaData.Parameters.Remove($p)
    }
}
if ($AddParameter)
{
    @'
    <#
    You are responsible for implementing the logic for added parameters.  These
    parameters are bound to $PSBoundParameters so if you pass them on the the
    command you are proxying, it will almost certainly cause an error.  This logic
    should be added to your BEGIN statement to remove any specified parameters
    from $PSBoundParameters.

```

In general, the way you are going to implement additional parameters is by modifying the way you generate the \$scriptCmd variable. Here is an example of how you would add a -SORTBY parameter to a cmdlet:

```

    if ($SortBy)
    {
        [Void]$PSBoundParameters.Remove("SortBy")
        $scriptCmd = {& $wrappedCmd @PSBoundParameters |Sort-Object -Property
$SortBy}
    }else
    {
        $scriptCmd = {& $wrappedCmd @PSBoundParameters }
    }

```

```

#####
New ATTRIBUTES
'@
    foreach ($p in @($AddParameter))
    {
        $attribute = $AddParameterAttribute.$p
        [Void]$MetaData.Parameters.Add($p, $(New-object
System.Management.Automation.ParameterMetadata $p ))
    }
"@
    if (`$p)
    {
        [Void]`$PSBoundParameters.Remove("$p")
    }
"@
}
#####
#>
[System.Management.Automation.ProxyCommand]::create($MetaData)
}#New-ProxyCommand

```

```

if (0)
{
    $cl = Get-WmiObject -List win32_Service -Amended
    $cl

    $s = New-ProxyCommand -Name Get-WmiObject -RemoveParameter
class,list,amended,asjob,namespace,property,query,recurse
    $s -replace '@PSBoundParameters','@PSBoundParameters | \%{$_}.'

```

```
New-ProxyCommand -Name Get-WmiObject -AddParameter @{test="value";test2="value2"}  
  
}
```

## test.ps1

```
function test-x
```

```
{  
<#  
You are responsible for implementing the logic for added parameters. These  
parameters are bound to $PSBoundParameters so if you pass them on the the  
command you are proxying, it will almost certainly cause an error. This logic  
should be added to your BEGIN statement to remove any specified parameters  
from $PSBoundParameters.
```

In general, the way you are going to implement additional parameters is by  
modifying the way you generate the \$scriptCmd variable. Here is an example  
of how you would add a -SORTBY parameter to a cmdlet:

```
    if ($SortBy)  
    {  
        [Void]$PSBoundParameters.Remove("SortBy")  
        $scriptCmd = {& $wrappedCmd @PSBoundParameters |Sort-Object -Property  
$SortBy}  
    }else  
    {  
        $scriptCmd = {& $wrappedCmd @PSBoundParameters }  
    }  
  
#####  
New ATTRIBUTES  
    if ($OnComplete)  
    {  
        [Void]$PSBoundParameters.Remove("OnComplete")  
    }  
  
#####  
#>
```

```
[CmdletBinding(DefaultParameterSetName='Default')]  
param(  
    [Parameter(ParameterSetName='Default')]  
    [Switch]  
    ${AsJob},  
  
    [System.Management.AuthenticationLevel]  
    ${Authentication},  
  
    [Alias('Size','Bytes','BS')]  
    [ValidateRange(0, 65500)]  
    [System.Int32]  
    ${BufferSize},  
  
    [Parameter(Mandatory=$true, Position=0, ValueFromPipelineByPropertyName=$true)]  
    [Alias('CN','IPAddress','__SERVER','Destination')]  
    [ValidateNotNullOrEmpty()]  
    [System.String[]]
```

```

    ${ComputerName},

    [ValidateRange(1, 4294967295)]
    [System.Int32]
    ${Count},

    [ValidateNotNullOrEmpty()]
    [System.Management.Automation.PSCredential]
    ${Credential},

    [Parameter(Position=1)]
    [Alias('FCN','SRC')]
    [ValidateNotNullOrEmpty()]
    [System.String[]]
    ${Source},

    [System.Management.ImpersonationLevel]
    ${Impersonation},

    [Parameter(ParameterSetName='Default')]
    [ValidateRange(-2147483648, 1000)]
    [System.Int32]
    ${ThrottleLimit},

    [Alias('TTL')]
    [ValidateRange(1, 255)]
    [System.Int32]
    ${TimeToLive},

    [ValidateRange(1, 60)]
    [System.Int32]
    ${Delay},

    [Parameter(ParameterSetName='Quiet')]
    [Switch]
    ${Quiet},

    [Parameter(ParameterSetName='Default')]
    [ScriptBlock]
    ${OnComplete}
)

begin
{
    try {
        $outBuffer = $null
        if ($PSBoundParameters.TryGetValue('OutBuffer', [ref]$outBuffer))
        {
            $PSBoundParameters['OutBuffer'] = 1
        }
        if ($OnComplete)
        {
            [Void]$PSBoundParameters.Remove("OnComplete")
            if (!( $PSBoundParameters["AsJob"] ))
            {
                $PSBoundParameters.Add("AsJob", $true)
            }
        }
    }
}

```

```

        $wrappedCmd = $ExecutionContext.InvokeCommand.GetCommand('Test-Connection',
[System.Management.Automation.CommandTypes]::Cmdlet)
        $scriptCmd = {& $wrappedCmd @PSBoundParameters }
        if ($OnComplete)
        {
            $ScriptCmd = $ExecutionContext.InvokeCommand.NewScriptBlock( {
                $Scriptcmd + "| % {$_"
            }
        )
        }
        $steppablePipeline = $scriptCmd.GetSteppablePipeline($myInvocation.CommandOrigin)
        $steppablePipeline.Begin($PSCmdlet)
    } catch {
        throw
    }
}

process
{
    try {
        $steppablePipeline.Process($_)
    } catch {
        throw
    }
}

end
{
    try {
        $steppablePipeline.End()
    } catch {
        throw
    }
}
<#

.ForwardHelpTargetName Test-Connection
.ForwardHelpCategory Cmdlet

#>

}

```

## Root folder

### Format-Table.ps1

```

function Format-Table
{
    [CmdletBinding()]
    param(

        [switch]
        ${AutoSize},

        [switch]

```

```

    ${HideTableHeaders},

    [switch]
    ${Wrap},

    [Parameter(Position=0)]
    [System.Object[]]
    ${Property},

    [System.Object]
    ${GroupBy},

    [string]
    ${View},

    [switch]
    ${ShowError},

    [switch]
    ${DisplayError},

    [switch]
    ${Force},

    [ValidateSet('CoreOnly','EnumOnly','Both')]
    [string]
    ${Expand},

    [Parameter(ValueFromPipeline=$true)]
    [psobject]
    ${InputObject},

    [Alias("ii")]
    [Switch]
    $IncludeIndex
)
begin
{
    ## Access the REAL Foreach-Object command, so that command
    ## wrappers do not interfere with this script
    $foreachObject = $ExecutionContext.InvokeCommand.GetCmdlet(
        "Microsoft.PowerShell.Core\Foreach-Object")

    $wrappedCmd = $ExecutionContext.InvokeCommand.GetCommand(
        'Format-Table',
        [System.Management.Automation.CommandTypes]::Cmdlet)

    $null = $PSBoundParameters.Remove("IncludeIndex")

    ## finalPipeline represents the pipeline we wil ultimately run
    $newPipeline = { & $wrappedCmd @PSBoundParameters }
    $finalPipeline = $newPipeline.ToString()

    if($IncludeIndex)
    {
        function Add-IndexParameter
        {
            begin
            {

```

```

        $psIndex = 0
    }
    process
    {
        ## If this is the Format-Table header
        if($_.GetType().FullName -eq `
            "Microsoft.PowerShell.Commands.Internal." +
            "Format.FormatStartData")
        {
            ## Take the first column and create a copy of it
            $formatStartType =
                $_.shapeInfo.tableColumnInfoList[0].GetType()
            $clone =
                $formatStartType.GetConstructors()[0].Invoke($null)

            ## Add a PSIndex property
            $clone.PropertyName = "PSIndex"
            $clone.Width = $clone.PropertyName.Length

            ## And add its information to the header information
            $_.shapeInfo.tableColumnInfoList.Insert(0, $clone)
        }

        ## If this is a Format-Table entry
        if($_.GetType().FullName -eq `
            "Microsoft.PowerShell.Commands.Internal." +
            "Format.FormatEntryData")
        {
            ## Take the first property and create a copy of it
            $firstField =
                $_.formatEntryInfo.formatPropertyFieldList[0]
            $formatFieldType = $firstField.GetType()
            $clone =
                $formatFieldType.GetConstructors()[0].Invoke($null)

            ## Set the PSIndex property value
            $clone.PropertyValue = $psIndex
            $psIndex++

            ## And add its information to the entry information
            $_.formatEntryInfo.formatPropertyFieldList.Insert(
                0, $clone)
        }

        $_
    }
}

$newPipeline = { __ORIGINAL_COMMAND__ | Add-IndexParameter }

## Replace the __ORIGINAL_COMMAND__ tag with the code
## that represents the original command
$alteredPipeline = $newPipeline.ToString()
$finalPipeline = $alteredPipeline.Replace(
    '__ORIGINAL_COMMAND__', $finalPipeline)
}

$steppablePipeline = [ScriptBlock]::Create(
    $finalPipeline).GetSteppablePipeline()

```

```

        $steppablePipeline.Begin($PSCmdlet)
    }

    process
    {
        $steppablePipeline.Process($_)
    }

    end
    {
        $steppablePipeline.End()
    }

    <#

    .ForwardHelpTargetName Format-Table
    .ForwardHelpCategory Cmdlet

    #>
}

```

## get-help.ps1

```

function Get-Help
{
    # Many changes made to the parameters to create a new parameter set that only has the
    ErrorRecord parameter
    [CmdletBinding(DefaultParameterSetName='AllUsersView')]
    param(
        [Parameter(Position=0,
        ValueFromPipelineByPropertyName=$true, ParameterSetName='DetailedView')]
        [Parameter(Position=0,
        ValueFromPipelineByPropertyName=$true, ParameterSetName='AllUsersView')]
        [Parameter(Position=0,
        ValueFromPipelineByPropertyName=$true, ParameterSetName='Examples')]
        [Parameter(Position=0,
        ValueFromPipelineByPropertyName=$true, ParameterSetName='Parameters')]
        [string]
        ${Name},

        [Parameter(ParameterSetName='DetailedView')]
        [Parameter(ParameterSetName='AllUsersView')]
        [Parameter(ParameterSetName='Examples')]
        [Parameter(ParameterSetName='Parameters')]
        [string]
        ${Path},

        [Parameter(ParameterSetName='DetailedView')]
        [Parameter(ParameterSetName='AllUsersView')]
        [Parameter(ParameterSetName='Examples')]
        [Parameter(ParameterSetName='Parameters')]
        [string[]]
        ${Category},

        [Parameter(ParameterSetName='DetailedView')]
        [Parameter(ParameterSetName='AllUsersView')]
        [Parameter(ParameterSetName='Examples')]
        [Parameter(ParameterSetName='Parameters')]

```

```

[string[]]
${Component},

[Parameter (ParameterSetName='DetailedView')]
[Parameter (ParameterSetName='AllUsersView')]
[Parameter (ParameterSetName='Examples')]
[Parameter (ParameterSetName='Parameters')]
[string[]]
${Functionality},

[Parameter (ParameterSetName='DetailedView')]
[Parameter (ParameterSetName='AllUsersView')]
[Parameter (ParameterSetName='Examples')]
[Parameter (ParameterSetName='Parameters')]
[string[]]
${Role},

[Parameter (ParameterSetName='DetailedView')]
[switch]
${Detailed},

[Parameter (ParameterSetName='AllUsersView')]
[switch]
${Full},

[Parameter (ParameterSetName='Examples')]
[switch]
${Examples},

[Parameter (ParameterSetName='Parameters')]
[string]
${Parameter},

[Parameter (ParameterSetName='Error', Position=0)]
[System.Management.Automation.ErrorRecord]
${ErrorRecord},

[Parameter (ParameterSetName='MSType', Position=0)]
${MSType},

[Parameter (ParameterSetName='Type', Position=0)]
${Type},

[Parameter (ParameterSetName='DetailedView')]
[Parameter (ParameterSetName='AllUsersView')]
[Parameter (ParameterSetName='Examples')]
[Parameter (ParameterSetName='Parameters')]
[switch]
${Online})

begin
{
    try {
        $outBuffer = $null
        if ($PSBoundParameters.TryGetValue('OutBuffer', [ref]$outBuffer))
        {
            $PSBoundParameters['OutBuffer'] = 1
        }
    }
}

```

```

# Add the fully qualified name for the old Get-Help command
$wrappedCmd =
$ExecutionContext.InvokeCommand.GetCommand('Microsoft.PowerShell.Core\Get-Help',
[System.Management.Automation.CommandTypes]::Cmdlet)

# Our new Error record parameter was specified
if($PSBoundParameters['ErrorRecord'])
{
    # Import System.Web for the HttpUtility to do URL Encoding
    [System.Reflection.Assembly]::LoadWithPartialName("System.Web") | Out-Null

    # Generate the search url (search URL + encoded FullyQualifiedErrorId)
    $url =
"http://social.technet.microsoft.com/wiki/search/SearchResults.aspx?q=" +
[System.Web.HttpUtility]::UrlEncode($ErrorRecord.FullyQualifiedErrorId)

    # Launch it in IE (you'd probably want to check for default browser, etc...)
    $scriptCmd = {& Start-Process -FilePath "C:\Program Files (x86)\Internet
Explorer\iexplore.exe" -ArgumentList $url}
}
elseif($PSBoundParameters['MSType'])
{
    # Import System.Web for the HttpUtility to do URL Encoding
    [System.Reflection.Assembly]::LoadWithPartialName("System.Web") | Out-Null

    if ($Type -is [System.String])
    {
        $url = "http://msdn.microsoft.com/en-us/library/{0}_members(VS.85).aspx"
-f $MSType
    }else
    {
        $url = "http://msdn.microsoft.com/en-us/library/{0}_members(VS.85).aspx"
-f $MSType.GetType().FullName
    }
    # Launch it in IE (you'd probably want to check for default browser, etc...)
    $scriptCmd = {& Start-Process -FilePath "C:\Program Files (x86)\Internet
Explorer\iexplore.exe" -ArgumentList $url}
}
elseif($PSBoundParameters['Type'])
{
    # Import System.Web for the HttpUtility to do URL Encoding
    [System.Reflection.Assembly]::LoadWithPartialName("System.Web") | Out-Null

    if ($Type -is [System.String])
    {
        $url = "http://www.bing.com/search?q={0}+members" -f $Type
    }else
    {
        $url = "http://www.bing.com/search?q={0}+members" -f
$Type.GetType().FullName
    }
    # Launch it in IE (you'd probably want to check for default browser, etc...)
    $scriptCmd = {& Start-Process -FilePath "C:\Program Files (x86)\Internet
Explorer\iexplore.exe" -ArgumentList $url}
}
else # Our record parameter was not provided. Fall back to the original Get-Help
{

```

```

        $scriptCmd = {& $wrappedCmd @PSBoundParameters }
    }

    $steppablePipeline = $scriptCmd.GetSteppablePipeline($myInvocation.CommandOrigin)
    $steppablePipeline.Begin($PSCmdlet)
} catch {
    throw
}
}

process
{
    try {
        $steppablePipeline.Process($_)
    } catch {
        throw
    }
}

end
{
    try {
        $steppablePipeline.End()
    } catch {
        throw
    }
}
}
<#

.ForwardHelpTargetName Get-Help
.ForwardHelpCategory Cmdlet

#>

```

## Get-Service.ps1

```

[CmdletBinding(DefaultParameterSetName='Default')]
param(
    [Parameter(ParameterSetName='Default', Position=0, ValueFromPipeline=$true,
ValueFromPipelineByPropertyName=$true)]
    [Alias('ServiceName')]
    [ValidateSet('ALG', 'WINRM')]
    [System.String[]]
    ${Name},

    [Parameter(ValueFromPipelineByPropertyName=$true)]
    [Alias('Cn')]
    [ValidateNotNullOrEmpty()]
    [System.String[]]
    ${ComputerName},

    [Alias('DS')]
    [Switch]
    ${DependentServices},

    [Alias('SDO', 'ServicesDependedOn')]
    [Switch]

```

```

    ${RequiredServices},

    [Parameter(ParameterSetName='DisplayName', Mandatory=$true)]
    [System.String[]]
    ${DisplayName},

    [ValidateNotNullOrEmpty()]
    [System.String[]]
    ${Include},

    [ValidateNotNullOrEmpty()]
    [System.String[]]
    ${Exclude},

    [Parameter(ParameterSetName='InputObject', ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [System.ServiceProcess.ServiceController[]]
    ${InputObject})

begin
{
    try {
        $outBuffer = $null
        if ($PSBoundParameters.TryGetValue('OutBuffer', [ref]$outBuffer))
        {
            $PSBoundParameters['OutBuffer'] = 1
        }
        $wrappedCmd = $ExecutionContext.InvokeCommand.GetCommand('Get-Service',
[System.Management.Automation.CommandTypes]::Cmdlet)
        $scriptCmd = {@ $wrappedCmd @PSBoundParameters }
        $steppablePipeline = $scriptCmd.GetSteppablePipeline($myInvocation.CommandOrigin)
        $steppablePipeline.Begin($PSCmdlet)
    } catch {
        throw
    }
}

process
{
    try {
        $steppablePipeline.Process($_)
    } catch {
        throw
    }
}

end
{
    try {
        $steppablePipeline.End()
    } catch {
        throw
    }
}
<#

.ForwardHelpTargetName Get-Service
.ForwardHelpCategory Cmdlet

```

```
#>
```

## GetHelpProxyCommand.ps1

```
[CmdletBinding(DefaultParameterSetName='AllUsersView')]
param(
    [Parameter(Position=0, ValueFromPipelineByPropertyName=$true)]
    [System.String]
    ${Name},

    [System.String]
    ${Path},

    [System.String[]]
    ${Category},

    [System.String[]]
    ${Component},

    [System.String[]]
    ${Functionality},

    [System.String[]]
    ${Role},

    [Parameter(ParameterSetName='DetailedView')]
    [Switch]
    ${Detailed},

    [Parameter(ParameterSetName='AllUsersView')]
    [Switch]
    ${Full},

    [Parameter(ParameterSetName='Examples')]
    [Switch]
    ${Examples},

    [Parameter(ParameterSetName='Parameters')]
    [System.String]
    ${Parameter},

    [Switch]
    ${Online})

begin
{
    try {
        $outBuffer = $null
        if ($PSBoundParameters.TryGetValue('OutBuffer', [ref]$outBuffer))
        {
            $PSBoundParameters['OutBuffer'] = 1
        }
        $wrappedCmd = $ExecutionContext.InvokeCommand.GetCommand('Get-Help',
[System.Management.Automation.CommandTypes]::Cmdlet)
        $scriptCmd = {& $wrappedCmd @PSBoundParameters }
        $steppablePipeline = $scriptCmd.GetSteppablePipeline($myInvocation.CommandOrigin)
        $steppablePipeline.Begin($PSCmdlet)
    }
}
```

```

    } catch {
        throw
    }
}

process
{
    try {
        $steppablePipeline.Process($_)
    } catch {
        throw
    }
}

end
{
    try {
        $steppablePipeline.End()
    } catch {
        throw
    }
}
<#

.ForwardHelpTargetName Get-Help
.ForwardHelpCategory Cmdlet

#>

```

## Out-Default.ps1

```

function Out-Default
{
    [CmdletBinding()]
    param(

        [Parameter(ValueFromPipeline=$true)]
        [psobject]
        ${InputObject}
    )
    begin
    {
        $cachedOutput = New-Object System.Collections.ArrayList

        $wrappedCmd = $ExecutionContext.InvokeCommand.GetCommand(
            'Out-Default',
            [System.Management.Automation.CommandTypes]::Cmdlet)

        $newPipeline = { & $wrappedCmd @PSBoundParameters }
        $steppablePipeline = $newPipeline.GetSteppablePipeline()
        $steppablePipeline.Begin($PSCmdlet)
    }

    process
    {
        ## If we get an input object, add it to our list of objects
        if($_ -ne $null) { $null = $cachedOutput.Add($_) }
    }
}

```

```

while($cachedOutput.Count -gt 500) { $cachedOutput.RemoveAt(0) }

$steppablePipeline.Process($_)
}

end
{
  ## Be sure we got objects that were not just errors (
  ## so that we don't wipe out the saved output when we get errors
  ## trying to work with it.)
  ## Also don't capture formatting information, as those objects
  ## can't be worked with.
  $uniqueOutput = $cachedOutput | Foreach-Object {
    $_.GetType().FullName } | Select -Unique
  $containsInterestingTypes = ($uniqueOutput -notcontains `
    "System.Management.Automation.ErrorRecord") -and
    ($uniqueOutput -notlike `
      "Microsoft.PowerShell.Commands.Internal.Format.*")

  ## If we actually had output, and it was interesting information,
  ## save the output into the $ll variable
  if(($cachedOutput.Count -gt 0) -and $containsInterestingTypes)
  {
    $GLOBAL:ll = $cachedOutput | % { $_ }
  }

  $steppablePipeline.End()
}

<#
  .ForwardHelpTargetName Out-Default
  .ForwardHelpCategory Cmdlet
#>
}

```

## Start-Job.ps1

```

function Start-Job {
  <#
    To create a proxy function for the Start-Job cmdlet, paste the results of the
    following command into the body of this function and then remove this comment:
    [Management.Automation.ProxyCommand]::Create((New-Object
    Management.Automation.CommandMetaData (Get-Command Start-Job)))
  #>

  [CmdletBinding(DefaultParameterSetName='ComputerName')]
  param(
    [Parameter(ValueFromPipelineByPropertyName=$true)]
    [System.String]
    ${Name},

    [Parameter(ParameterSetName='ComputerName', Mandatory=$true, Position=0)]
    [Alias('Command')]
    [System.Management.Automation.ScriptBlock]
    ${ScriptBlock},

    [System.Management.Automation.PSCredential]

```

```

    ${Credential},

    [Parameter(ParameterSetName='FilePathComputerName', Position=0)]
    [Alias('PSPath')]
    [System.String]
    ${FilePath},

    [System.Management.Automation.Runspace.AuthenticationMechanism]
    ${Authentication},

    [Parameter(Position=1)]
    [System.Management.Automation.ScriptBlock]
    ${InitializationScript},

    [Switch]
    ${RunAs32},

    [System.Management.Automation.ScriptBlock]
    ${OnCompletionAction},

    [Parameter(ValueFromPipeline=$true)]
    [System.Management.Automation.PSObject]
    ${InputObject},

    [Alias('Args')]
    [System.Object[]]
    ${ArgumentList})

begin
{
    try {
        $outBuffer = $null
        if ($PSBoundParameters.TryGetValue('OutBuffer', [ref]$outBuffer))
        {
            $PSBoundParameters['OutBuffer'] = 1
        }
        $wrappedCmd = $ExecutionContext.InvokeCommand.GetCommand('Start-Job',
[System.Management.Automation.CommandTypes]::Cmdlet)

        # define string variable to become the target command line
        #region Initialize helper variable to create command
        $scriptCmdPipeline = ''
        #endregion

        # add new parameter handling
        #region Process and remove the OnCompletionAction parameter if it is present
        if ($OnCompletionAction) {
            $PSBoundParameters.Remove('OnCompletionAction') | Out-Null
            $scriptCmdPipeline += " | foreach-object{
`$job = Register-ObjectEvent -InputObject `$_ -EventName StateChanged -
SourceIdentifier JobEndAlert -Action {
    if(`$sender.State -eq 'Completed')
    {
        `& {
            $OnCompletionAction
        }
        Unregister-Event -SourceIdentifier JobEndAlert -Force
    }
}
}

```

```

    }"
}
#endregion

    $scriptCmd = {& $wrappedCmd @PSBoundParameters }

    $scriptCmd = $ExecutionContext.InvokeCommand.NewScriptBlock(
        [string]$scriptCmd + $scriptCmdPipeline
    )

    $steppablePipeline = $scriptCmd.GetSteppablePipeline($myInvocation.CommandOrigin)
    $steppablePipeline.Begin($PSCmdlet)
} catch {
    throw
}
}

process
{
    try {
        $steppablePipeline.Process($_)
    } catch {
        throw
    }
}

end
{
    try {
        $steppablePipeline.End()
    } catch {
        throw
    }
}
<#

.ForwardHelpTargetName Start-Job
.ForwardHelpCategory Cmdlet

#>

}

```

## Story.ps1

```

Get-Help -mstype -type
format-table -Includeindex
Start-Job -OnCompletionAction
Out-default $!l

```

```

#####
#& call operator
&"hostname"
&'C:\Program Files\Windows NT\Accessories\wordpad.exe'
&"get-process -name *ss"

$x="OLD"
{$x="NEW"; $x}
&{$x="NEW"; $x}

```

```

$X

$P = Get-Command Get-Process
&$P

."hostname"
.{$X="NEW"; $X}
$X
.$P
#####
#Modulename cmdlets
function Get-UICulture {"NOT What you wanted"}
&"Get-UICulture"
&"Microsoft.PowerShell.Utility\Get-UICulture"
gcm -type cmdlet |select *Name |ogv

#####
#Splattting
Get-Process -name lsass -FileVersionInfo
$h = @{"name="Lsass"; FileVersionInfo=1}
Get-Process @h

$h1 = @{"name="LsaSS"}
$h2 = @{"FileVersionInfo=1"}
Get-Process @h1 @h2

$h2 = @{"File=1"}
Get-Process @h1 @h2

$h2=@{"test=1"}
Get-Process @h1 @h2

#####
#$PSboundParameters
function test {$psBoundParameters}
test 1 2 3
function test {param($a,$b,$c) $psBoundParameters}
test 1 2 3
function test-Service {param($Name) Get-Service @PSBoundParameters}
test-Service -Name alg
Test-Service -Name alg -ov b
$b
Get-Service alg |get-Service
Get-Service alg |Test-Service

#####
#SteppablePipelines
function test {begin{"B"} process {"P: $_"} End{"E"}}
gps *ss |Test
$s = {test}
$sp = $s.GetSteppablePipeline()
$sp.Begin(1)
$sp.Process("I")
$sp.Process("LOVE")
$sp.Process("PowerShell")
$sp.End()

function test-Service {

```

```

param([Parameter()]$Name)
Begin
    {
        $WrappedCmd = Get-Command Get-Service -CommandType Cmdlet
        $cmd = { & $wrappedCmd @PSBoundParameters }
        $sp = $cmd.GetSteppablePipeline()
        $sp.Begin($pscmdlet)
    }
Process { $sp.Process($_) }
End { $sp.End() }
}
test-Service -Name alg
Test-Service -Name alg -ov b
$b
Get-Service alg |Test-Service

#Create a proxy for get-service
$Cmd = Get-Command Get-Service
$Metadata = New-Object System.Management.Automation.CommandMetaData $Cmd
$Metadata.Name = "Get-MyService"
[System.Management.Automation.ProxyCommand]::Create($Metadata) > Get-MyService.ps1
.\Get-MyService -Name ap*

$Metadata = New-Object System.Management.Automation.CommandMetaData $Cmd
$Metadata.Parameters.Clear()
$Metadata.Name = "Get-MyService"
[System.Management.Automation.ProxyCommand]::Create($Metadata) > Get-MyService.ps1
.\Get-MyService -Name ap*

$Metadata = New-Object System.Management.Automation.CommandMetaData $Cmd
$Metadata.Name = "Get-MyService"
$Metadata.Parameters.Name.Attributes.Add([System.Management.Automation.ValidateSetAttribute][string[]] ("ALG", "WINRM"))
[System.Management.Automation.ProxyCommand]::Create($Metadata) > Get-MyService.ps1
.\Get-MyService -Name appinfo
.\Get-MyService -Name alg

```