```
<#
   .SYNOPSIS
      The New-CCCommandWrapper function uses .NET Framework
      functionality to create the code for a wrapper around
      the identified PowerShell command.

   .DESCRIPTION
      The New-CCCommandWrapper function takes the name of an existing
      PowerShell command and creates a function definition which is a
      wrapper around the the command, preserving its metadata and functionality
      while providing the access needed to add paramaters and funtionality or
      alter exisitng functionality.

   .PARAMETER CommandName
      The CommandName parameter takes the name of the exisiting PowerShell command
      to wrap.

   .PARAMETER NounPrefix
      The NounPrefix is used to rename the command to identify the source of the
      alteration.

   .PARAMETER CopyToClipboard
      The CopyToClipboard switch instructs the command to copy the produced code to
      the clipboard rather than outputting it.

   .PARAMETER RawCode
      The RawCode switch instructs the command to simply return the code generated
      from the metadata rather than wrapping it in a function definition.

   .EXAMPLE
      PS C:\> New-CCCommandWrapper -CommandName ConvertTo-HTML

   .EXAMPLE
      PS C:\> New-CCCommandWrapper -CommandName ConvertTo-HTML -NounPrefix Test

   .EXAMPLE
      PS C:\> New-CCCommandWrapper -CommandName ConvertTo-HTML -NounPrefix Test -Path c:\Test.ps1

#>
function New-CCCommandWrapper
{
   [CmdletBinding()]
   Param
   (
      [Parameter(Mandatory = $True)]
      [String]$CommandName,
        [String]$NounPrefix = "CC",
      [Switch]$CopyToClipboard,
      [Switch]$RawCode
   )
   [System.Text.StringBuilder]$sb = New-Object System.Text.StringBuilder | Out-Null
    $functionName = $CommandName.Replace("-","-$NounPrefix")
    $sb = "function " + $functionName + "`n{`n`t"
   $command = Get-Command $CommandName
   $metadata = New-Object System.Management.Automation.CommandMetaData($command)
    [String]$code = [System.Management.Automation.ProxyCommand]::Create($metadata)
```

```
    [String]$formattedCode = $code.Replace("`n","`n`t")
    if ($formattedCode.EndsWith("`t"))
    {
        $sb.Append($formattedCode.Substring(0,($formattedCode.Length - 1))) | Out-Null
    }
    else
    {
        $sb.Append($formattedCode) | Out-Null
    }

   $sb.Append("}") | Out-Null

   if ($RawCode)
   {
      $codeToReturn = $code
   }
   else
   {
      $codeToReturn = $sb.ToString()
   }
    if ($CopyToClipboard)
    {
      Set-Clipboard -Value $codeToReturn
        Write-Verbose "$functionName copied to clipboard"
    }
    else
    {
      Write-Output $codeToReturn
    }
}

function ConvertTo-CCHTML
{
   [CmdletBinding(DefaultParameterSetName = 'Page', HelpUri = 'http://go.microsoft.com/fwlink/?LinkID=113290', RemotingCapability = 'None')]
   param
   (
      [Parameter(ValueFromPipeline = $true)]
      [psobject]${InputObject},
      [Parameter(Position = 0)]
      [System.Object[]]${Property},
      [Parameter(ParameterSetName = 'Page', Position = 3)]
      [string[]]${Body},
      [Parameter(ParameterSetName = 'Page', Position = 1)]
      [string[]]${Head},
      [Parameter(ParameterSetName = 'Page', Position = 2)]
      [ValidateNotNullOrEmpty()]
      [string]${Title},
      [ValidateSet('Table', 'List')]
      [ValidateNotNullOrEmpty()]
      [string]${As},
      [Parameter(ParameterSetName = 'Page')]
      [Alias('cu', 'uri')]
      [ValidateNotNullOrEmpty()]
      [uri]${CssUri},
      [Parameter(ParameterSetName = 'Fragment')]
      [ValidateNotNullOrEmpty()]
```

```
    [switch]${Fragment},
    [ValidateNotNullOrEmpty()]
    [string[]]${PostContent},
    [ValidateNotNullOrEmpty()]
    [string[]]${PreContent}
)

begin
{
    try
    {
        $defaultTableStyle = '<style>
            th {
                  VERTICAL-ALIGN:  TOP;
                  COLOR:  #018AC0;
                  TEXT-ALIGN:  left;
                  background-color:LightSteelBlue;
                  color:Black;
                  BORDER: 1px  solid black;
                  }
            table, td
            {
                border: 1px solid black;
            }
            td
            {
                padding: 5px;
            }
            tr:nth-child(even) {background-color:White;}
            tr:nth-child(odd) {background-color:AliceBlue;}
            </style>'
        if ([String]::IsNullOrWhiteSpace($Head))
        {
            $PSBoundParameters['Head'] = $defaultTableStyle
        }
        $outBuffer = $null
        if ($PSBoundParameters.TryGetValue('OutBuffer', [ref]$outBuffer))
        {
            $PSBoundParameters['OutBuffer'] = 1
        }
        $wrappedCmd = $ExecutionContext.InvokeCommand.GetCommand('Microsoft.PowerShell.Utility\ConvertTo-Html',
[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
        }
    }
}
```