```
<#
.SYNOPSIS
  This module contains some common functions that are used by many HSC PowerShell
files.

.DESCRIPTION
  This module contains the common functions that are used by many HSC PowerShell
files. These functions are:

  1. Set-HSCEnvironment
  2. Set-HSCWindowTitle
  3. Get-HSCParameter
  4. Start-HSCCountdown
  5. Test-HSCVerbose
  6. Write-HSCColorOutput
  7. Get-HSCLogFileName
  8. Test-HSCLogFilePath
  9. Remove-HSCOldLogFile
  10. Write-HSCLogFileSummaryInformation
  11. Send-HSCEmail
  12. Get-HSCPasswordFromSecureStringFile
  13. Get-HSCRandomPassword
  14. Exit-HSCCommand
  15. Test-HSCValidWVUEmail
  16. Get-HSCServerName
  17. Get-HSCEncryptedFilePath
  18. New-HSCSecureStringFile

.NOTES
  HSC-CommonCodeModule.psm1
  Last Modified by: Jeff Brusoe
  Last Modified: June 23, 2020

  Version: 2.0
#>

[CmdletBinding()]
[Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSAvoidTrailingWhiteSpace","",Jus
ification = "Not relevant")]
param ()

function Set-HSCEnvironment
{
  <#
  .SYNOPSIS
    This function configures the HSC PowerShell environment.

  .DESCRIPTION
    This function configures the environment for files that use this module. It
performs the follwing tasks.
    1. Sets strictmode to the latest version
    2. Clear $Error variable
    3. Clear PS window
    4. Sets the PowerShell window title
    5. Set location to root of ps1 directory
    6. Generates transcript log file path
    7. Start transcript log file
    8. Removes old .txt log files
    9. Set $ErrorActionPreference

  .PARAMETER NoSessionTranscript
    By default, a session transcript is created. This parameter prevents creating tha
file.

  .PARAMETER LogFilePath
    THe path to write any logs files and the session transcript. It defaults to
$PSScriptRoot\Logs

  .PARAMETER StopOnError
    Stops program execution if an error is detected

  .PARAMETER DaysToKeepLogFiles
    Determines how long old log files should be kept for

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: June 23, 2020
  #>

  [CmdletBinding()]
[Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChangir
Functions","",Justification = "Doesn't make serious state changes")]
  [Alias("Set-Environment")]
  param (
    [bool]$NoSessionTranscript=$false,
    [string]$LogFilePath = $($MyInvocation.PSScriptRoot + "\Logs\"),
    [bool]$StopOnError=$false,
    [int]$DaysToKeepLogFiles = 5
  )

  process
  {
    #1. Set strict mode
    Set-StrictMode -Version Latest #Configures for current scope (Probably not neede
    Set-PSDebug -Strict #Configures struct mode for global scope

    #2. Clear Error Variable
    $global:Error.Clear()

    #3. Clear PS Window
    Clear-Host

    #4. Set Window Title
    Set-HSCWindowTitle -WindowTitle $MyInvocation.PSCommandPath

    #5. Set location to $PSScriptRoot
    Set-Location $MyInvocation.PSScriptRoot #Don't use $PSScriptRoot here since that
puts it in the common code directory instead of the script root directory.

    #6. Generate transcript log file path
    #Parse file location to determine program name
    Write-Output $("PSCommandPath: " + $MyInvocation.PSCommandPath) | Out-Host
    $ProgramName = $MyInvocation.PSCommandPath
    $ProgramName = $ProgramName.substring(0,$ProgramName.indexOf("."))
    $ProgramName = $ProgramName.substring($ProgramName.lastindexOf("\")+1)
    Write-Output "Program Name: $ProgramName" | Out-Host

    $TranscriptLogFile = Get-HSCLogFileName -ProgramName $ProgramName
    Write-Output "Transcript File Path: $TranscriptLogFile" | Out-Host

    #7 & 8. Start transcript and remove old log files
    if (Test-HSCLogFilePath -LogFilePath $LogFilePath)
    {
      if (!$NoSessionTranscript)
      {
        Write-Verbose "Starting transcript log file" | Out-Host
        Start-Transcript $TranscriptLogFile | Out-Host
      }
      else
      {
        Write-Output "Transcript log file will not be created..." | Out-Host
      }

      Write-Output "Removing old log files" | Out-Host
      Remove-HSCOldLogFile -CSV -TXT -Path $LogFilePath -Days $DaysToKeepLogFiles
-Verbose -Delete
    }
    else
    {
      Write-Output "Log file path doesn't exist..." | Out-Host
    }

    #9. Set $ErrorActionPreference
    if ($StopOnError)
    {
      $global:ErrorActionPreference = "Stop"
    }
    else
    {
      $global:ErrorActionPreference = "Continue"
    }
  }
}

function Set-HSCWindowTitle
{
  <#
  .SYNOPSIS
    Sets the PowerShell window title

  .DESCRIPTION
    The purpose of this function is to change the title in the PowerShell window.
    It can do this by either passing in a value or by parsing up the file path.

  .PARAMETER WindowTitle
    This is a string parameter that specifies the PowerShell window title. If it
    isn't provided, it will be determined by the $PSCommandPath variable.

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: June 2, 2020
  #>

  [CmdletBinding()]
[Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChangir
Functions","", Justification = "Start-Sleep Doesn't Change System State.")]
  [Alias("Set-WindowTitle")]
  param (
    [string]$WindowTitle=$MyInvocation.PSCommandPath #Full path with file name
  )

  process
  {
    if (![string]::IsNullOrEmpty($WindowTitle))
    {
      try
      {
        Write-Verbose "Setting window title" | Out-Host
        $WindowTitle = $WindowTitle.substring($WindowTitle.lastindexOf("\")+1)
        Write-Verbose $WindowTitle | Out-Host

        $Host.UI.RawUI.WindowTitle = $WindowTitle
      }
      catch
      {
        Write-Warning "Unable to set the window title" | Out-Host
      }
    }
  }
}


function Get-HSCParameter
{
  <#
  .SYNOPSIS
    The purpose of this function is to display any nondefault parameters that were
passed to the originating function.

  .PARAMETER ParameterList
    This parameter comes from the built-in $PSBoundParameters variable.
    See: https://blogs.msdn.microsoft.com/timid/2014/08/12/psboundparameters-and-
commonparameters-whatif-debug-etc/

  .NOTES
```

```
    Written by: Jeff Brusoe
    Last  Updated by: Jeff Brusoe
    Last Updated: June 3, 2020
  #>

  [CmdletBinding()]
  [Alias("Get-Parameter")]
  param (
    [Parameter(Mandatory=$true)][hashtable]$ParameterList
  )

  process
  {
    try
    {
      if (($ParameterList.keys | Measure-Object).Count -eq 0)
      {
        Write-Output "All input parameters are set to default values." | Out-Host
      }
      else
      {
        Write-Output "The following parameters have nondefault values:" | Out-Host

        foreach ($key in $ParameterList.keys)
        {
          $param = Get-Variable -Name $key -ErrorAction SilentlyContinue

          if($null -ne $param)
          {
            Write-Output "$($param.name): $($param.value)" | Out-Host
          }
        }
      }
    }
    catch
    {
      Write-Warning "There was an error generating the parameter list." | Out-Host
    }

    Write-Output "`n" | Out-Host
  }
}

Function Start-HSCCountdown
{
  <#
  .SYNOPSIS
    This function displays a progress bar and message stating the reason for the
delay.
    It is basically a more user friendly version of Start-Sleep which may look like
the window
    has locked up if it is used.

  .PARAMETER Seconds
    This is the integer value that tells how long the pause should occur for.
```

```powershell

  .PARAMETER Messsage
    The actdual message to dispaly in the countdown box

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Originally Written: October 21, 2016
    Last Updated; June 23, 2020
  #>

  [CmdletBinding(PositionalBinding=$false)]
[Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChangir
Functions","", Justification = "Start-Sleep Doesn't Change System State.")]
  [Alias("Start-Countdown")]
  Param(
    [Parameter(ValueFromPipeline=$true)][Int32]$Seconds = 10,
    [Parameter(ValueFromPipeline=$true)][string]$Message = "Pausing for $Seconds seconds..."
  )

  process
  {
    for ($Count=1; $Count -le $Seconds; $Count++)
    {
      Write-Progress -Id 1 -Activity $Message -Status "Waiting for $Seconds seconds, $($Seconds - $Count) left" -PercentComplete (($Count / $Seconds) * 100)
      Start-Sleep -Seconds 1
    }

    Write-Progress -Id 1 -Activity $Message -Status "Completed" -PercentComplete 100 -Completed
  }
}

function Test-HSCVerbose
{
  <#
  .SYNOPSIS
    This function determines if the verbose common parameter has been used.

  .DESCRIPTION
    The purpose of this function is to return true/false depending on whether
    the verbose parameter has been passed to the calling PowerShell file.

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: August 10, 2018
  #>

  [cmdletbinding()]
  [Alias("Test-Verbose")]
  [OutputType([bool])]
  param ()

  begin
  {
    Write-Output "Test-Verbose: Testing for verbose parameter" | Out-Host
  }

  process
  {
    if ($PSCmdlet.MyInvocation.BoundParameters["Verbose"].IsPresent)
    {
      Write-Output "Test-Verbose: Verbose is present" | Out-Host
      return $true
    }
    else
    {
      Write-Output "Test-Verbose: Verbose is not present" | Out-Host
      return $false
    }
  }
}

Function Write-HSCColorOutput
{
  <#
    .SYNOPSIS
      This function changes the output color and uses Write-Output to log stuff to tl
session transcript.

    .DESCRIPTION
      This function allows color output in combination with Write-Output.
      It's needed since Write-Output doesn't support this feature found in Write-Host
      Write-Output is used due to some issues writing log files.

      In this code, ForegroundColor refers to the color of the text.

    .NOTES
      Written by: Jeff Brusoe
      Last Updated: June 5, 2020
  #>

  [CmdletBinding(PositionalBinding=$false)]
  [Alias("Write-ColorOutput")]
  param (
    [Parameter(Mandatory=$true,ValueFromPipeline=$true)][string[]]$Message,
    [string]$ForegroundColor = "Green"
  )

  begin
  {
    if ([string]::IsNullOrEmpty($Message))
    {
      Write-Warning "A null message value was passed into the function." | Out-Host
      return $null
    }
```

```powershell
    elseif ([Enum]::GetValues([System.ConsoleColor]) -NotContains $ForegroundColor)
    {
      Write-Verbose "An invalid system color was passed into the function. The defaul value of green is being used." | Out-Host
      $ForegroundColor = "Green"
    }

    $CurrentColor = [Console]::ForegroundColor
    $BackgroundColor = [Console]::BackgroundColor

    if ($CurrentColor -eq $ForegroundColor)
    {
      Write-Verbose "Current color matches input foreground color." | Out-Host
    }

    if ($BackgroundColor -eq "ForegroundColor")
    {
      Write-Verbose "Foreground color matches background color and will not be changed." | Out-Host
    }
  }

  process
  {
    [Console]::ForegroundColor = $ForegroundColor

    foreach ($m in $Message)
    {
      Write-Output $m
    }
  }

  end
  {
    [Console]::ForegroundColor = $CurrentColor
  }
}

function Get-HSCLogFileName
{
  <#
  .SYNOPSIS
    This function generates the names of the various log files.

  .DESCRIPTION
    The purpose of this function is to generate the names of log files used by the calling file. It is used
    with the Start-Transcript cmdlet. The path used is the one supplied by the $LogFilePath parameter
    which is being passed into the function. The date format used is Year-Month-Day(2 digit)-Hour(24 hour time)-Minute(2 digit).

  .PARAMETER ProgramName
    ProgramName is the user provided name of the program. It is used to help
    build the session transcript log name. If it is null, then its use is omitted.

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: June 3, 2020

    See this link for information about PowerShell's return values and why Out-Null i
used here.
    https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/abou
/about_return?view=powershell-6
  #>

  [Cmdletbinding()]
  [Alias("Get-LogFileName")]
  [OutputType([string])]
  Param(
    [string]$ProgramName=$null,
    [ValidateSet("SessionTranscript","Error","Output","Other")]
[string]$LogFileType="SessionTranscript",
    [ValidateSet("txt","csv","log")][string]$FileExtension="txt"
  )

  Write-Output "Generating $LogFileType log file name..." | Out-Null

  [string]$LogFile=$null

  if ([string]::IsNullOrEmpty($ProgramName))
  {
    $LogFile = $LogFilePath + "\" + (Get-Date -format yyyy-MM-dd-HH-mm) +
"-$LogFileType.$FileExtension"
  }
  else
  {
    $LogFile = $LogFilePath + "\" + (Get-Date -format yyyy-MM-dd-HH-mm) +
"-$ProgramName-$LogFileType.$FileExtension"
  }

  Write-Verbose "Log File: $LogFile" | Out-Host

  return $LogFile
}

function Test-HSCLogFilePath
{
  <#
  .DESCRIPTION
    This function verifies that the log file path exists.
    An option exists to create the path if it doesn't exist.

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: April 10, 2018
  #>

  [Cmdletbinding()]
  [Alias("Test-LogFilePath")]
  [OutputType([bool])]
  Param(
    [string]$LogFilePath,
    [switch]$CreatePath
  )

  if ([string]::IsNullOrEmpty($LogFilePath))
  {
    Write-Warning "Log file path is empty." | Out-Null

    return $false
  }

  if (!(Test-Path -Path $LogFilePath))
  {
    if ($CreatePath)
    {
      Write-Output "Log file path doesn't exist and is being created..." | Out-Null

      try
      {
        New-Item -Path $LogFilePath -ItemType "Directory" -ErrorAction "Stop"
        return $true
      }
      catch
      {
        Write-Error "Unable to create log file path directory" | Out-Null
        return $false
      }
    }
  }
  else
  {
    return $true
  }
}

Function Remove-HSCOldLogFile
{
  <#
  .DESCRIPTION
    This function searches for log files older than three days (or a value specified
by the user)
    and removes (or copies) the files from a specified directory.

  .NOTES
    Written by Kevin Russell
    Last updated by: Jeff Brusoe
    Last Updated: August 26, 2019

    Function Status: Working, but making changes to improve functionality.

    To Do
    1. Add ability to copy files instead of delete
    2. Loop around if/then statements for file paths until a valid path is entered.
    3. Add ability to accept custom file extensions
  #>

  [Cmdletbinding()]
[Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChangir
Functions","", Justification = "Just needed to remove old log files")]
  [Alias("Remove-OldLogFiles")]
  [Alias("Remove-OldLogFile")]
  [OutputType([string])]

  Param(
    [string]$path = $($MyInvocation.PSScriptRoot + "\Logs\"),
    [switch]$CSV,
    [switch]$TXT,
    [switch]$LOG,
    [switch]$LBB, #Generated from SAN encryption key backup
    [switch]$Delete,
    #[string]$CopyPath = $null - Needs to be implemented
    [int]$Days = 3
  )

  Write-Verbose "Days to keep log files: $Days" | Out-Host

  if ($Days -gt 0)
  {
    $Days = -1*$Days
  }

  if ($Delete)
  {
    Write-Output "Files will be deleted." | Out-Host
  }
  else
  {
    Write-Output "Files will not be deleted." | Out-Host
  }

  $time = (Get-Date).AddDays($Days)

  Write-Verbose "Removing old log files" | Out-Host

  $ValidPath = $false

  while(!$ValidPath)
  {
    if ([string]::IsNullorEmpty($path))
    {
      $path = Read-Host "Please enter the directory path"
    }
    elseif (!(Test-Path $path))
    {
      $path = Read-Host "Please enter a valid directory path"
    }
    else
    {
      $ValidPath = $true
    }
  }

  $RemoveString = @() #Array of file extensions to remove

  if ($CSV)
  {
    Write-Verbose "Adding csv files to remove string." | Out-Host
    $RemoveString += "*.csv"
  }

  if ($TXT)
  {
    Write-Verbose "Adding txt files to remove string" | Out-Host
    $RemoveString += "*.txt"
  }

  if ($LOG)
  {
    $RemoveString += "*.log"
  }

  if ($LBB)
  {
    $RemoveString += "*.lbb"
  }

  if (($RemoveString | Measure-Object).Count -eq 0)
  {
    Write-Output "No files to remove" | Out-Host

    return $null
  }

  Write-Verbose "RemoveString: $RemoveString" | Out-Host

  if ([string]::IsNullOrEmpty($RemoveString))
  {
    Write-Output "Unable to remove any files." | Out-Host
  }
  else
  {
    Write-Output "Path: $path" | Out-Host
    $files = Get-ChildItem -path $path\* -Include $RemoveString

    if ($null -eq $files)
    {
      #Nothing is found
      Write-Verbose "No files in directory" | Out-Host
    }
    else
  {
   Write-Verbose $("File Count: " + ($files | Measure-Object).Count) | Out-Host

   foreach ($file in $files)
   {
    #Write-Output $file.FullName

    if($file.LastWriteTime -lt $time)
    {
     if (!$Delete)
     {
      Write-Output $("Potential Delete: " + $file.FullName) | Out-Host
     }
     else
     {
      Write-Verbose $("Removing: " + $file.FullName) | Out-Host

      Remove-Item -Path $file.fullname -Force
     }
    }
   }
  }
 }
}

Function Write-HSCLogFileSummaryInformation
{
 <#
 .SYNOPSIS
  This function writes common information to log files used for Active Directory
  and Exchange PowerShell files.

 .NOTES
  Written By: Matt Logue
  Last Updated:November 13, 2016
 #>

 [cmdletbinding()]
 [Alias("Write-LogFileSummaryInformation")]
 Param(
  [string]$FilePath = $null, #A null path will just put this information on the screen
  [switch]$ComputerName, #$true = include computer name in log file
  [switch]$ExcludedUsers, #$true = display list of users excluded from processing
  [string]$Summary = $null #if not null output summary
 )

   $dateTime = Get-Date -Format G

 if (([string]::IsNullOrEmpty($FilePath)) -or ((Test-Path -Path $FilePath) -eq $false))
 {
     Write-Verbose $("*-------------- "+$dateTime+"--------------*") | Out-Host
     Write-ColorOutput -Message "File Path is Empty" -ForegroundColor "Green" -Verbose | Out-Host
```

```

          if ($ComputerName -eq $true)
          {
        Write-Verbose $("Computer Name: "+ $env:computername) | Out-Host
          }

    if ($ExcludedUsers -eq $true)
    {
      Write-Verbose $("Excluded Users: ") | Out-Host
    }

    if (![string]::IsNullOrEmpty($Summary))
    {
      Write-Verbose $("Summary: "+ $Summary) | Out-Host
    }

    }
  else
  {

    Write-Verbose ("*-------------- $dateTime --------------*`n`r") | Out-Host
    Add-Content -Value ("*-------------- $dateTime --------------*`n`r") -Path $FilePath

    Write-Verbose $("File: "+ $FilePath) | Out-Host
    Add-Content -Value "File: $FilePath`n`r" -Path $FilePath

    if ($ComputerName -eq $true)
    {
      Write-Verbose $("Computer Name: "+ $env:computername) | Out-Host
      Add-Content -Value "`r`nComputerName: $env:computername`n`r" -Path $FilePath
    }

    if ($ExcludedUsers -eq $true)
    {
      Write-Verbose $("Excluded Users: ") | Out-Host
      Add-Content -Value "`r`nExcluded Users: `r`n" -Path $FilePath
    }

    if (![string]::IsNullOrEmpty($Summary))
    {
      Write-Verbose $("Summary: "+ $Summary) | Out-Host
      Add-Content -Value "Summary:`r`n$Summary" -Path $FilePath
    }
  }
}

Function Send-HSCEmail
{
  <#
  .DESCRIPTION
    The purpose of this function is to serve as a wrapper for the Send-MailMessage cmdlet. This is done to handle
    decrypting the encrypted password file which is needed to relay mail with Send-MailMessage.

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: April 16, 2018

    This function probably isn't needed anymore. Need to see if it is still being use
before removing it though.
  #>

  [CmdletBinding()]
  [Alias("Send-Email")]
  Param (
    [string[]]$To,
    [string]$From,
    [string]$Subject,
    [string]$MessageBody,
    [string[]]$Attachments,
    [string]$SMTPServer = "Hssmtp.hsc.wvu.edu"
  )

  Write-Verbose "Preparing to send email..." | Out-Host
  $Error.Clear()

  try
  {
    Send-MailMessage -to $To -From $From -SMTPServer $SMTPServer -Subject $Subject
-UseSSL -port 587 -Attachments $Attachments -Body $MessageBody -ErrorAction Stop
  }
  catch
  {
    Write-Warning "Unable to send email message" | Out-Host
  }
}

function Get-HSCPasswordFromSecureStringFile
{
  <#
    .SYNOPSIS
      The purpose of this function is to decrypt a secure string file to handle user
authentication to Office 365
      or other HSC protected environments.

    .DESCRIPTION
       This function decrypts a secure string file in order to use that for
authentication. In order to decrypt it,
       the file must have been encrypted on the same machine with the same logged in
user used as the one being used
       for decryption. There are also options to change the secure string file as wel
as prompt the user for credentials.

    .PARAMETER Prompt
      Causes the function to prompt the user for credentials instead of reading them
from a file.

```

```
    .PARAMETER ChangeSecureStringFile
      Writes a new secure string file

    .PARAMETER EncryptedFileDirectory
      This parameter is the path to the encrypted files directory. It defaults to:
      C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-Repository\HSCCustomModules
\EncryptedFiles\.

    .PARAMETER PWFile
      The actual name of the password file to be decrypted.

    .NOTES
      Written by: Jeff Brusoe
      Last Updated by: Jeff Brusoe
      Last Updated: June 23, 2020
  #>

  [CmdletBinding()]
  [Alias("Get-PasswordFromSecureStringFile")]
  param (
    [bool]$Prompt=$false,
    [bool]$ChangeSecureStringFile=$false,
    [string]$EncryptedFileDirectory = "C:\Users\microsoft\Documents\GitHub\HSC-
PowerShell-Repository\HSCCustomModules\EncryptedFiles\",
    [string]$PWFile = "normal2.txt" #Mandatory
  )

  begin
  {
    [string]$Password=$null
    $PWFile = $EncryptedFileDirectory + $PWFile

    Write-Verbose "Password File Path: $PWFile" | Out-Host
  }

  process
  {
    if ($ChangeSecureStringFile)
    {
      try
      {
        Read-Host "Enter Current Password" -AsSecureString | ConvertFrom-SecureStrin
| Out-File $PWFile
        Write-HSCColorOutput -foregroundcolor "Green" -Message "Successfully updated
secure string file.`n" | Out-Host
      }
      catch
      {
        $Prompt = $false
        Write-Error "There was an error generating the secure string file.`n" | Out-
Host
      }
    }

    if ($Prompt)
```

```powershell
    {
      $Password = Read-Host "Enter Password" | Out-Host
    }
    else
    {
      if (Test-Path $PasswordFile)
      {
        Write-HSCColorOutput -ForegroundColor "Green" -Message "Decrypting Password..." | Out-Host

        try
        {
          $securestring = convertto-securestring -string (get-content $PWFile)
          $bstr = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($securestring)
          $Password = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr

          Write-HSCColorOutput -ForegroundColor "Green" -Message "Password decrypted successfully." | Out-Host
        }
        catch
        {
          Write-Error "There was an error decrypting the password. Exiting file." | Out-Host
        }
      }
    }
  } #End process block

  end
  {
    return $Password
  }
}

function Get-HSCRandomPassword
{
  <#
  .SYNOPSIS
    The purpose of this function is to generate a random password.

  .DESCRIPTION
    The password generated meets WVU password complexity requirements:
    1. Must be between 8 and 20 characters in length.
    2. Must contain characters from at least three of the following four categories:
       a. Uppercase letters: A-Z
       b. Lowercase letters: a-z
       c. Numbers: 0-9
       d. Only these special characters: ! ^ ? : . ~ - _

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: June 23, 2020
  #>

  [CmdletBinding()]
  [ALias("Get-RandomPassword")]
  [OutputType([string])]
  param (
    [int]$PasswordLength = 19
  )

  begin
  {
    Write-Verbose "Generating random password..." | Out-Host
  }

  process
  {
    try
    {
      #https://blogs.technet.microsoft.com/undocumentedfeatures/2016/09/20/powershell-random-password-generator/
      [string]$Password = ([char[]]([char]33..[char]95) + ([char[]]([char]97..[char]126)) + 0..9 | Sort-Object {Get-Random})[0..$PasswordLength] -join ''

      Write-Vervose "Password: $Password" | Out-Host
    }
    catch
    {
      Write-Warning "Error generating random password" | Out-Host
      [string]$Password = $null
    }
    finally
    {
      Write-Verbose "Done generating random password" | Out-Host
    }
  }

  end
  {
    return $Password
  }
}

function Exit-HSCCommand
{
  <#
  .DESCRIPTION
    This function is called to handle error conditions where a PS file should exit.

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: June 4, 2020
  #>

  [CmdletBinding()]
  [Alias("Exit-Commands")]
  [Alias("Exit-Command")]
  param ()

  #To do: Display way program is stopping (Complete, error & location, etc.)
  process
  {
    try
    {
      Write-Verbose "Stopping Transcript" | Out-Host
      Stop-Transcript -ErrorAction Stop
    }
    catch
    {
      Write-Verbose "Unable to stop transcript" | Out-Host
    }
    finally
    {
      Write-Verbose "Exiting file" | Out-Host
    }
  }

  end
  {
    exit
  }
}

function Test-HSCValidWVUEmail
{
  <#
  .SYNOPSIS
    This function tests whether an email is a valid WVU email address. It only checks
if
    it's possible, but not that the account actually exists.

  .DESCRIPTION
    There are currently only two tests to determine if the email address is valid.
    1. Does the email address contain wvu.edu?
    2. Is there an @ symbol in the email address string

  .PARAMETER EmailAddress
    This email address is what will be tested by the logic of hte code.

  .OUTPUTS
    Returns a boolean value to indicate whether the email address is valid

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: June 23, 2020
  #>

  [CmdletBinding()]
  [Alias("Test-ValidWVUEmail")]
  [OutputType([bool])]
  param (
    [Parameter(Mandatory=$True)][string]$EmailAddress
  )

  begin
  {
    $ValidEmail = $false
  }

  process
  {
    Write-Verbose "Attempting to verify: $EmailAddress" | Out-Host

    if (($EmailAddress.indexOf("wvu.edu") -gt 0) -AND ($EmailAddress.indexOf("@")
-gt0))
    {
      Write-Verbose "The email is valid" | Out-Host
      $ValidEmail = $true
    }
    else
    {
      Write-Verbose "The email is invalid" | Out-Host
    }
  }

  end
  {
    return $ValidEmail
  }
}

function Get-HSCServerName
{
  <#
  .SYNOPSIS
    This function returns the name of the server currently running the ps1 file.

  .PARAMETER MandatoryServerNames
    This paramter tells the function to only return the server name if the name is
included
    in the $AllowedServerNames array. Currently, this array contains the three
sysscript
    servers (sysscript2, sysscript3, and sysscript4).

  .OUTPUTS
    Returns the server name as a string. If the server name can't be determined, ther
    it returns $null.

  .EXAMPLE
    Get-HSCServerName
    <return server name>

  .EXAMPLE
    Get-HSCServeName -MandatoryServerNames
    - sysscript2 (if on sysscript2)
    - $null if not on sysscript2, 3, or 4

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: June 23, 2020
  #>

  [CmdletBinding()]
  [Alias("Get-ServerName")]
  [OutputType([string])]
  param(
    [switch]$MandatoryServerNames
  )

  begin
  {
    $AllowedServerNames = @("sysscript2","sysscript3","sysscript4")
  }

  process
  {
    try
    {
      [string]$ServerName = (Get-ChildItem env:computername).Value
      Write-Verbose "Server Name: $ServerName" | Out-Host

      if ($MandatoryServerNames -AND $AllowedServerNames -contains $ServerName)
      {
        return $ServerName
      }
      elseif ($MandatoryServerNames -AND $AllowedServerNames -notcontains $ServerName
      {
        Write-Warning "Server name is not in server name array" | Out-Host
        return $null
      }
      elseif ([string]::IsNullOrEmpty($ServerName))
      {
        Write-Warning "Error retrieving server name" | Out-Host
        return $null
      }
      else
      {
        return $ServerName
      }
    }
    catch
    {
      Write-Warning "Error retrieving server name" | Out-Host
      return $null
    }
  } #end process
}

function Get-HSCEncryptedFilePath
```

```
{
  <#
    .SYNOPSIS
      Returns the path to the encrypted files used to establish O365 tenant connectic

    .NOTES
      Written by: Jeff Brusoe
      Last Updated: June 16, 2020
  #>

  [CmdletBinding()]
  [Alias("Get-EncryptedFilePath")]
  [OutputType([String])]
  param (
    [ValidateSet("sysscript2", "sysscript3", "sysscript4")]
    [string]$ServerName = (Get-HSCServerName -MandatoryServerName)
  )

  if ([string]::IsNullOrEmpty($ServerName))
  {
    Write-Warning "Unable to get encrypted file path" | Out-Host
    return $null
  }

  try
  {
    Write-Output "`n`nServer Name: $ServerName" | Out-Host

    $ServerNumber = $ServerName.substring($ServerName.Length - 1)
    Write-Verbose "ServerNumber: $ServerNumber" | Out-Host

    $EncryptedFilePath = "C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-
Repository\1HSCCustomModules\EncryptedFiles\normal$ServerNumber.txt"

    Write-Output "Encrypted File Path: $EncryptedFilePath" | Out-Host

    return $EncryptedFilePath
  }
  catch
  {
    Write-Warning "Unable to get encrypted file path" | Out-Host
    return $null
  }
}

function New-HSCSecureStringFile
{
  <#
    .SYNOPSIS
      Creates a new secure string file.

    .NOTES
      Written by: Jeff Brusoe
      Last Updated: June 18, 2020
  #>

  [CmdletBinding()]
  [OutputType([bool])]
  param(
    [string]$OutputFilePath = "C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-Repository\1HSCCustomModules\EncryptedFiles\",
    [parameter(Mandatory=$true)]
    [string]$UserName
  )

  if (Test-Path $OutputFilePath)
  {
    try
    {
      Read-Host "Enter Current Password" -assecurestring | ConvertFrom-SecureString | Out-File "$OutputFilePath\$UserName.txt"
      Write-Verbose "Successfully updated secure string file." | Out-Host
    }
    catch
    {

      Write-Warning "There was an error generating the secure string file." | Out-Hos
    }
  }
  else
  {
    Write-Warning "File output path doesn't exist" | Out-Host
    return $false
  }
}

####################
# Export functions #
####################

#Exit Modules
Export-ModuleMember -Function "Exit-HSCCommand" -Alias "Exit-Commands","Exit-Command"

#Get Modules
Export-ModuleMember -Function "Get-HSCServerName" -Alias "Get-ServerName"
Export-ModuleMember -Function "Get-HSCEncryptedFilePath" -Alias "Get-EncryptedFilePath"
Export-ModuleMember -Function "Get-HSCPasswordFromSecureStringFile" -Alias "Get-PasswordFromSecureStringFile"
Export-ModuleMember -Function "Get-HSCParameter" -Alias "Get-Parameter"
Export-ModuleMember -Function "Get-HSCLogFileName" -Alias "Get-LogFileName"
Export-ModuleMember -Function "Get-HSCRandomPassword" -Alias "Get-RandomPassword"

#New-Modules
Export-ModuleMember -Function "New-HSCSecureStringFile"

#Remove Modules
Export-ModuleMember -Function "Remove-HSCOldLogFile" -Alias "Remove-OldLogFiles"

#Send Modules
```

```powershell
Export-ModuleMember -Function "Send-HSCEmail" -Alias "Send-Email"

#Set Modules
Export-ModuleMember -Function "Set-HSCEnvironment" -Alias "Set-Environment"
Export-ModuleMember -Function "Set-HSCWindowTitle" -Alias "Set-WindowTitle"

#Start Modules
Export-ModuleMember -Function "Start-HSCCountdown" -Alias "Start-Countdown"

#Test Modules
Export-ModuleMember -Function "Test-HSCValidWVUEmail" -Alias "Test-ValidWVUEmail"
Export-ModuleMember -Function "Test-HSCVerbose" -Alias "Test-Verbose"
Export-ModuleMember -Function "Test-HSCLogFilePath" -Alias "Test-LogFilePath"

#Write Modules
Export-ModuleMember -Function "Write-HSCColorOutput" -Alias "Write-ColorOutput"
Export-ModuleMember -Function "Write-HSCLogFileSummaryInformation" -Alias "Write-
LogFileSummaryInformation"
```