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

  1. Set-Environment
  2. Set-WindowTitle
  3. Get-Paraeter
  4. Start-Countdown
  %. Test-Verbose
  6. Write-ColorOutput
  7. Get-LogFileName
  8. Test-LogFilePath
  9. Remove-OldLogFile
  10. Write-LogFileSummaryInformation
  11. Send-Email
  12. Get-PasswordFromSecureStringFile
  13. Get-RandomPassword
  14. Exit-Command
  15. Test-ValidWVUEmail

.NOTES
  HSC-CommonCodeModule.psm1
  Last Modified by: Jeff Brusoe
  Last Modified: Jnue 8, 2020

  Version: 1.5
#>

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

function Set-Environment
{
  <#
  .DESCRIPTION
    This function configures the environment for files to 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

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

  [CmdletBinding()]

[Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChanging
unctions","",Justification = "Doesn't make serious state changes")]
  param (
    [bool]$NoSessionTranscript=$false,
    [string]$LogFilePath = $($MyInvocation.PSScriptRoot + "\Logs\"),
    [bool]$StopOnError,
    [int]$DaysToKeepLogFiles = 5
  )

  #1. Set strict mode
  Set-StrictMode -Version Latest #Configures for current scope (Probably not needed)
  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-WindowTitle -WindowTitle $MyInvocation.PSCommandPath

  #5. Set location to $PSScriptRoot
  Set-Location $MyInvocation.PSScriptRoot #Don't use $PSScriptRoot here since that put
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-LogFileName -ProgramName $ProgramName
  Write-Output "Transcript File Path: $TranscriptLogFile" | Out-Host

  #7 & 8. Start transcript and remove old log files
  if (Test-LogFilePath -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-OldLogFile -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-WindowTitle
{
  <#
  .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("PSUseShouldProcessForStateChanging
unctions","", Justification = "Start-Sleep Doesn't Change System State.")]
  param (
    [string]$WindowTitle=$MyInvocation.PSCommandPath #Full path with file name
  )

  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-Parameter
{
  <#
  .DESCRIPTION
    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()]
  param (
    [Parameter(Mandatory=$true)][hashtable]$ParameterList
  )

  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-Countdown
{
  <#
  .DESCRIPTION
    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 th
window
    has locked up if it is used.

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

  .PARAMETER Messsage

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: October 21, 2016
  #>

[Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChanging
unctions","", Justification = "Start-Sleep Doesn't Change System State.")]
  [CmdletBinding(PositionalBinding=$false)]

  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-Verbose
{
  <#
  .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()]
  [OutputType([bool])]
  param ()

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

  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-ColorOutput
{
  <#
  .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=$false)]
  param (
    [Parameter(Mandatory=$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
    }
    elseif ([Enum]::GetValues([System.ConsoleColor]) -NotContains $ForegroundColor)
    {
      Write-Verbose "An invalid system color was passed into the function. The default
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-LogFileName
{
  <#
  .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
```

```powershell
    Last Updated by: Jeff Brusoe
    Last Updated: June 3, 2020

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

  [Cmdletbinding()]
  [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-LogFilePath
{
  <#
  .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()]
  [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-OldLogFile
{
  <#
  .DESCRIPTION
    This function searches for log files older than three days (or a value specified t
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
  #>

```

```
[Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChanging
unctions","", Justification = "Just needed to remove old log files")]
  [Alias("Remove-OldLogFiles")]
  [Cmdletbinding()]
  [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"

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

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

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

  Write-Verbose "Removing old log files"

  $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."
    $RemoveString += "*.csv"
  }

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

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

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

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

    return $null
  }

  Write-Verbose "RemoveString: $RemoveString"

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

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

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

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

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

Function Write-LogFileSummaryInformation
{
 <#
 .DESCRIPTION
  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()]
 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-Email
{
 <#
 .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 usec
before removing it though.
```

```
  #>

  [CmdletBinding()]
  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-PasswordFromSecureStringFile
{

  <#
  .DESCRIPTION
    The purpose of this function is to decrypt a secure string file to handle user
authentication to
    AD or Office 365.

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: August 27, 2019
  #>

  [CmdletBinding()]
  param (
    [bool]$Prompt=$false,
    [bool]$ChangeSecureStringFile=$false,
    [string]$PWFile = ".\EncryptedPassword.txt" #Mandatory
  )

  [string]$Password=$null

  if ($ChangeSecureStringFile)
  {
    try
    {
      Read-Host "Enter Current Password" -AsSecureString | convertfrom-securestring |
```

```powershell
Out-File $PWFile
   Write-ColorOutput -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)
 {
  $Password = Read-Host "Enter Password" | Out-Host
 }
 else
 {
  if (Test-Path $PasswordFile)
  {
   Write-ColorOutput -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-ColorOutput -ForegroundColor "Green" -Message "Password decrypted successfully." | Out-Host
   }
   catch
   {
    Write-Error "There was an error decrypting the password. Exiting file." | Out-Host
   }
  }
 }

 Return $Password
}

function Get-RandomPassword
{
 <#
 .DESCRIPTION
  The purpose of this function is to generate a random password. 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: April 12, 2018
  #>

  [CmdletBinding()]
  [OutputType([string])]
  param (
    [int]$PasswordLength = 19
  )

  Write-Output "Generating random password for new AD account" | Out-Host


  Write-Output "Generating password for new AD account" | Out-Host

  #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-Output "Password: $Password" | Out-Host

  return $Password
}

function Exit-Command
{
  <#
  .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")]
  param ()

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

  Exit
}

function Test-ValidWVUEmail
{
  <#
  .DESCRIPTION
    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.

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: June 8, 2020
  #>

  [CmdletBinding()]
  [OutputType([bool])]
  param (
    [Parameter(Mandatory=$True)][string]$EmailAddress
  )

  $ValidEmail = $false

  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
  }

  return $ValidEmail
}

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

Export-ModuleMember -Function "Get-*"
Export-ModuleMember -Function "Set-Environment"
Export-ModuleMember -Function "Write-*"
Export-ModuleMember -Function "Send-Email"
Export-ModuleMember -Function "Set-WindowTitle"
Export-ModuleMember -Function "Remove-OldLogFile" -Alias "Remove-OldLogFiles"
Export-ModuleMember -Function "Start-*"
Export-ModuleMember -Function "Test-*"
Export-ModuleMember -Function "Exit-Command" -Alias "Exit-Commands"
```