

```
<#
.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","",Justification = "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 following tasks.
        1. Sets strictmode to the latest version
        2. Clear $Error variable
        3. Clear PS window
        4. Sets the PowerShell window title
```

```
51     5. Set location to root of ps1 directory
52     6. Generates transcript log file path
53     7. Start transcript log file
54     8. Removes old .txt log files
55     9. Set $ErrorActionPreference
56
57 .PARAMETER NoSessionTranscript
58     By default, a session transcript is created. This parameter prevents creating the
    file.
59
60 .PARAMETER LogFilePath
61     The path to write any logs files and the session transcript. It defaults to
    $PSScriptRoot\Logs
62
63 .PARAMETER StopOnError
64     Stops program execution if an error is detected
65
66 .PARAMETER DaysToKeepLogFiles
67     Determines how long old log files should be kept for
68
69 .NOTES
70     Written by: Jeff Brusoe
71     Last Updated by: Jeff Brusoe
72     Last Updated: June 23, 2020
73 #>
74
75 [CmdletBinding()]
76
    [Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChanging
    Functions","",Justification = "Doesn't make serious state changes")]
77 [Alias("Set-Environment")]
78 param (
79     [bool]$NoSessionTranscript=$false,
80     [string]$LogFilePath = $($MyInvocation.PSScriptRoot + "\Logs\"),
81     [bool]$StopOnError=$false,
82     [int]$DaysToKeepLogFiles = 5
83 )
84
85 process
86 {
87     #1. Set strict mode
88     Set-StrictMode -Version Latest #Configures for current scope (Probably not needed)
89     Set-PSDebug -Strict #Configures strict mode for global scope
90
91     #2. Clear Error Variable
92     $global:Error.Clear()
93
94     #3. Clear PS Window
95     Clear-Host
96
97     #4. Set Window Title
98     Set-HSCWindowTitle -WindowTitle $MyInvocation.PSCommandPath
99
100    #5. Set location to $PSScriptRoot
101    Set-Location $MyInvocation.PSScriptRoot #Don't use $PSScriptRoot here since that
```

```

    puts it in the common code directory instead of the script root directory.
102
103     #6. Generate transcript log file path
104     #Parse file location to determine program name
105     Write-Output $("PSCommandPath: " + $MyInvocation.PSCommandPath) | Out-Host
106     $ProgramName = $MyInvocation.PSCommandPath
107     $ProgramName = $ProgramName.substring(0,$ProgramName.indexOf("."))
108     $ProgramName = $ProgramName.substring($ProgramName.lastIndexOf("\")+1)
109     Write-Output "Program Name: $ProgramName" | Out-Host
110
111     $TranscriptLogFile = Get-HSCLogFileName -ProgramName $ProgramName
112     Write-Output "Transcript File Path: $TranscriptLogFile" | Out-Host
113
114     #7 & 8. Start transcript and remove old log files
115     if (Test-HSCLogFilePath -LogFilePath $LogFilePath)
116     {
117         if (!$NoSessionTranscript)
118         {
119             Write-Verbose "Starting transcript log file" | Out-Host
120             Start-Transcript $TranscriptLogFile | Out-Host
121         }
122         else
123         {
124             Write-Output "Transcript log file will not be created..." | Out-Host
125         }
126
127         Write-Output "Removing old log files" | Out-Host
128         Remove-HSCOldLogFile -CSV -TXT -Path $LogFilePath -Days $DaysToKeepLogFiles
129     }
130     else
131     {
132         Write-Output "Log file path doesn't exist..." | Out-Host
133     }
134
135     #9. Set $ErrorActionPreference
136     if ($StopOnError)
137     {
138         $global:ErrorActionPreference = "Stop"
139     }
140     else
141     {
142         $global:ErrorActionPreference = "Continue"
143     }
144 }
145 }
146
147 function Set-HSCWindowTitle
148 {
149     <#
150     .SYNOPSIS
151         Sets the PowerShell window title
152
153     .DESCRIPTION
154         The purpose of this function is to change the title in the PowerShell window.

```

```
155     It can do this by either passing in a value or by parsing up the file path.
156
157 .PARAMETER WindowTitle
158     This is a string parameter that specifies the PowerShell window title. If it
159     isn't provided, it will be determined by the $PSCommandPath variable.
160
161 .NOTES
162     Written by: Jeff Brusoe
163     Last Updated by: Jeff Brusoe
164     Last Updated: June 2, 2020
165 #>
166
167 [CmdletBinding()]
168
169 [Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChangingFunctions", "", Justification = "Start-Sleep Doesn't Change System State.")]
170 [Alias("Set-WindowTitle")]
171 param (
172     [string]$WindowTitle=$MyInvocation.PSCommandPath #Full path with file name
173 )
174
175 process
176 {
177     if (![string]::IsNullOrEmpty($WindowTitle))
178     {
179         try
180         {
181             Write-Verbose "Setting window title" | Out-Host
182             $WindowTitle = $WindowTitle.substring($WindowTitle.lastIndexOf("\")+1)
183             Write-Verbose $WindowTitle | Out-Host
184
185             $Host.UI.RawUI.WindowTitle = $WindowTitle
186         }
187         catch
188         {
189             Write-Warning "Unable to set the window title" | Out-Host
190         }
191     }
192 }
193
194
195 function Get-HSCParameter
196 {
197     <#
198     .SYNOPSIS
199         The purpose of this function is to display any nondefault parameters that were
200         passed to the originating function.
201
202     .PARAMETER ParameterList
203         This parameter comes from the built-in $PSBoundParameters variable.
204         See: https://blogs.msdn.microsoft.com/timid/2014/08/12/psboundparameters-and-commonparameters-whatif-debug-etc/
205
206     .NOTES
```

```
206     Written by: Jeff Brusoe
207     Last Updated by: Jeff Brusoe
208     Last Updated: June 3, 2020
209     #>
210
211     [CmdletBinding()]
212     [Alias("Get-Parameter")]
213     param (
214         [Parameter(Mandatory=$true)][hashtable]$ParameterList
215     )
216
217     process
218     {
219         try
220         {
221             if (($ParameterList.keys | Measure-Object).Count -eq 0)
222             {
223                 Write-Output "All input parameters are set to default values." | Out-Host
224             }
225             else
226             {
227                 Write-Output "The following parameters have nondefault values:" | Out-Host
228
229                 foreach ($key in $ParameterList.keys)
230                 {
231                     $param = Get-Variable -Name $key -ErrorAction SilentlyContinue
232
233                     if($null -ne $param)
234                     {
235                         Write-Output "$($param.name): $($param.value)" | Out-Host
236                     }
237                 }
238             }
239         }
240         catch
241         {
242             Write-Warning "There was an error generating the parameter list." | Out-Host
243         }
244
245         Write-Output "`n" | Out-Host
246     }
247 }
248
249 Function Start-HSCCountdown
250 {
251     <#
252     .SYNOPSIS
253     This function displays a progress bar and message stating the reason for the
254     delay.
255     It is basically a more user friendly version of Start-Sleep which may look like
256     the window
257     has locked up if it is used.
258     .PARAMETER Seconds
259     This is the integer value that tells how long the pause should occur for.
```

```

259
260 .PARAMETER Message
261     The actual message to display in the countdown box
262
263 .NOTES
264     Written by: Jeff Brusoe
265     Last Updated by: Jeff Brusoe
266     Originally Written: October 21, 2016
267     Last Updated; June 23, 2020
268 #>
269
270 [CmdletBinding(PositionalBinding=$false)]
271
272 [Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChanging
Functions", "", Justification = "Start-Sleep Doesn't Change System State.")]
273 [Alias("Start-Countdown")]
274 Param(
275     [Parameter(ValueFromPipeline=$true)][Int32]$Seconds = 10,
276     [Parameter(ValueFromPipeline=$true)][string]$Message = "Pausing for $Seconds
seconds..."
277 )
278 process
279 {
280     for ($Count=1; $Count -le $Seconds; $Count++)
281     {
282         Write-Progress -Id 1 -Activity $Message -Status "Waiting for $Seconds seconds,
$($Seconds - $Count) left" -PercentComplete (($Count / $Seconds) * 100)
283         Start-Sleep -Seconds 1
284     }
285
286     Write-Progress -Id 1 -Activity $Message -Status "Completed" -PercentComplete 100
-Completed
287 }
288 }
289
290 function Test-HSCVerbose
291 {
292     <#
293     .SYNOPSIS
294         This function determines if the verbose common parameter has been used.
295
296     .DESCRIPTION
297         The purpose of this function is to return true/false depending on whether
298         the verbose parameter has been passed to the calling PowerShell file.
299
300     .NOTES
301         Written by: Jeff Brusoe
302         Last Updated by: Jeff Brusoe
303         Last Updated: August 10, 2018
304     #>
305
306 [cmdletbinding()]
307 [Alias("Test-Verbose")]
308 [OutputType([bool])]

```

```
309 param ()
310
311 begin
312 {
313     Write-Output "Test-Verbose: Testing for verbose parameter" | Out-Host
314 }
315
316 process
317 {
318     if ($PSCmdlet.MyInvocation.BoundParameters["Verbose"].IsPresent)
319     {
320         Write-Output "Test-Verbose: Verbose is present" | Out-Host
321         return $true
322     }
323     else
324     {
325         Write-Output "Test-Verbose: Verbose is not present" | Out-Host
326         return $false
327     }
328 }
329 }
330
331 Function Write-HSCColorOutput
332 {
333     <#
334         .SYNOPSIS
335             This function changes the output color and uses Write-Output to log stuff to the
336             session transcript.
337
338         .DESCRIPTION
339             This function allows color output in combination with Write-Output.
340             It's needed since Write-Output doesn't support this feature found in Write-Host
341             Write-Output is used due to some issues writing log files.
342
343             In this code, ForegroundColor refers to the color of the text.
344
345         .NOTES
346             Written by: Jeff Brusoe
347             Last Updated: June 5, 2020
348     #>
349     [CmdletBinding(PositionalBinding=$false)]
350     [Alias("Write-ColorOutput")]
351     param (
352         [Parameter(Mandatory=$true, ValueFromPipeline=$true)][string[]]$Message,
353         [string]$ForegroundColor = "Green"
354     )
355
356     begin
357     {
358         if ([string]::IsNullOrEmpty($Message))
359         {
360             Write-Warning "A null message value was passed into the function." | Out-Host
361             return $null
362         }
```

```
363     elseif ([Enum]::GetValues([System.ConsoleColor]) -NotContains $ForegroundColor)
364     {
365         Write-Verbose "An invalid system color was passed into the function. The default
value of green is being used." | Out-Host
366         $ForegroundColor = "Green"
367     }
368
369     $CurrentColor = [Console]::ForegroundColor
370     $BackgroundColor = [Console]::BackgroundColor
371
372     if ($CurrentColor -eq $ForegroundColor)
373     {
374         Write-Verbose "Current color matches input foreground color." | Out-Host
375     }
376
377     if ($BackgroundColor -eq "ForegroundColor")
378     {
379         Write-Verbose "Foreground color matches background color and will not be
changed." | Out-Host
380     }
381 }
382
383 process
384 {
385     [Console]::ForegroundColor = $ForegroundColor
386
387     foreach ($m in $Message)
388     {
389         Write-Output $m
390     }
391 }
392
393 end
394 {
395     [Console]::ForegroundColor = $CurrentColor
396 }
397 }
398
399 function Get-HSCLogFileName
400 {
401     <#
402     .SYNOPSIS
403         This function generates the names of the various log files.
404
405     .DESCRIPTION
406         The purpose of this function is to generate the names of log files used by the
calling file. It is used
407         with the Start-Transcript cmdlet. The path used is the one supplied by the
$LogFilePath parameter
408         which is being passed into the function. The date format used is Year-Month-Day(2
digit)-Hour(24 hour time)-Minute(2 digit).
409
410     .PARAMETER ProgramName
411         ProgramName is the user provided name of the program. It is used to help
412         build the session transcript log name. If it is null, then its use is omitted.
```



```
413
414 .NOTES
415     Written by: Jeff Brusoe
416     Last Updated by: Jeff Brusoe
417     Last Updated: June 3, 2020
418
419     See this link for information about PowerShell's return values and why Out-Null is
used here.
420     https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about\_return?view=powershell-6
421     #>
422
423 [Cmdletbinding()]
424 [Alias("Get-LogFileName")]
425 [OutputType([string])]
426 Param(
427     [string]$ProgramName=$null,
428     [ValidateSet("SessionTranscript", "Error", "Output", "Other")]
[string]$LogFileType="SessionTranscript",
429     [ValidateSet("txt", "csv", "log")] [string]$FileExtension="txt"
430 )
431
432 Write-Output "Generating $LogFileType log file name..." | Out-Null
433
434 [string]$LogFile=$null
435
436 if ([string]::IsNullOrEmpty($ProgramName))
437 {
438     $LogFile = $LogFilePath + "\" + (Get-Date -format yyyy-MM-dd-HH-mm) +
"$LogFileType.$FileExtension"
439 }
440 else
441 {
442     $LogFile = $LogFilePath + "\" + (Get-Date -format yyyy-MM-dd-HH-mm) +
"$ProgramName-$LogFileType.$FileExtension"
443 }
444
445 Write-Verbose "Log File: $LogFile" | Out-Host
446
447 return $LogFile
448 }
449
450 function Test-HSCLogFilePath
451 {
452     <#
453     .DESCRIPTION
454         This function verifies that the log file path exists.
455         An option exists to create the path if it doesn't exist.
456
457     .NOTES
458         Written by: Jeff Brusoe
459         Last Updated by: Jeff Brusoe
460         Last Updated: April 10, 2018
461     #>
462
```

```
463 [Cmdletbinding()]
464 [Alias("Test-LogFilePath")]
465 [OutputType([bool])]
466 Param(
467     [string]$LogFilePath,
468     [switch]$CreatePath
469 )
470
471 if ([string]::IsNullOrEmpty($LogFilePath))
472 {
473     Write-Warning "Log file path is empty." | Out-Null
474
475     return $false
476 }
477
478 if (!(Test-Path -Path $LogFilePath))
479 {
480     if ($CreatePath)
481     {
482         Write-Output "Log file path doesn't exist and is being created..." | Out-Null
483
484         try
485         {
486             New-Item -Path $LogFilePath -ItemType "Directory" -ErrorAction "Stop"
487             return $true
488         }
489         catch
490         {
491             Write-Error "Unable to create log file path directory" | Out-Null
492             return $false
493         }
494     }
495 }
496 else
497 {
498     return $true
499 }
500 }
501
502 Function Remove-HSCOldLogFile
503 {
504     <#
505     .DESCRIPTION
506         This function searches for log files older than three days (or a value specified
507         by the user)
508         and removes (or copies) the files from a specified directory.
509     .NOTES
510         Written by Kevin Russell
511         Last updated by: Jeff Brusoe
512         Last Updated: August 26, 2019
513
514         Function Status: Working, but making changes to improve functionality.
515
516         To Do
```

```
517     1. Add ability to copy files instead of delete
518     2. Loop around if/then statements for file paths until a valid path is entered.
519     3. Add ability to accept custom file extensions
520     #>
521
522     [Cmdletbinding()]
523
524     [Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSUseShouldProcessForStateChangingFunctions", "", Justification = "Just needed to remove old log files")]
525     [Alias("Remove-OldLogFiles")]
526     [Alias("Remove-OldLogFile")]
527     [OutputType([string])]
528
529     Param(
530         [string]$path = $($MyInvocation.PSScriptRoot + "\Logs\"),
531         [switch]$CSV,
532         [switch]$TXT,
533         [switch]$LOG,
534         [switch]$LBB, #Generated from SAN encryption key backup
535         [switch]$Delete,
536         #[string]$CopyPath = $null - Needs to be implemented
537         [int]$Days = 3
538     )
539
540     Write-Verbose "Days to keep log files: $Days" | Out-Host
541
542     if ($Days -gt 0)
543     {
544         $Days = -1*$Days
545     }
546
547     if ($Delete)
548     {
549         Write-Output "Files will be deleted." | Out-Host
550     }
551     else
552     {
553         Write-Output "Files will not be deleted." | Out-Host
554     }
555
556     $time = (Get-Date).AddDays($Days)
557
558     Write-Verbose "Removing old log files" | Out-Host
559
560     $ValidPath = $false
561
562     while(!$ValidPath)
563     {
564         if ([string]::IsNullOrEmpty($path))
565         {
566             $path = Read-Host "Please enter the directory path"
567         }
568         elseif (!(Test-Path $path))
569         {
570             $path = Read-Host "Please enter a valid directory path"
```

```
570     }
571     else
572     {
573         $ValidPath = $true
574     }
575 }
576
577 $RemoveString = @() #Array of file extensions to remove
578
579 if ($CSV)
580 {
581     Write-Verbose "Adding csv files to remove string." | Out-Host
582     $RemoveString += "/*.csv"
583 }
584
585 if ($TXT)
586 {
587     Write-Verbose "Adding txt files to remove string" | Out-Host
588     $RemoveString += "/*.txt"
589 }
590
591 if ($LOG)
592 {
593     $RemoveString += "/*.log"
594 }
595
596 if ($LBB)
597 {
598     $RemoveString += "/*.lbb"
599 }
600
601 if (($RemoveString | Measure-Object).Count -eq 0)
602 {
603     Write-Output "No files to remove" | Out-Host
604
605     return $null
606 }
607
608 Write-Verbose "RemoveString: $RemoveString" | Out-Host
609
610 if ([string]::IsNullOrEmpty($RemoveString))
611 {
612     Write-Output "Unable to remove any files." | Out-Host
613 }
614 else
615 {
616     Write-Output "Path: $path" | Out-Host
617     $files = Get-ChildItem -path $path\* -Include $RemoveString
618
619     if ($null -eq $files)
620     {
621         #Nothing is found
622         Write-Verbose "No files in directory" | Out-Host
623     }
624     else
```

```
625     {
626         Write-Verbose $("File Count: " + ($files | Measure-Object).Count) | Out-Host
627
628         foreach ($file in $files)
629         {
630             #Write-Output $file.FullName
631
632             if($file.LastWriteTime -lt $time)
633             {
634                 if (!$Delete)
635                 {
636                     Write-Output $("Potential Delete: " + $file.FullName) | Out-Host
637                 }
638                 else
639                 {
640                     Write-Verbose $("Removing: " + $file.FullName) | Out-Host
641
642                     Remove-Item -Path $file.fullname -Force
643                 }
644             }
645         }
646     }
647 }
648 }
649
650 Function Write-HSCLogFileSummaryInformation
651 {
652     <#
653     .SYNOPSIS
654         This function writes common information to log files used for Active Directory
655         and Exchange PowerShell files.
656
657     .NOTES
658         Written By: Matt Logue
659         Last Updated: November 13, 2016
660     #>
661
662     [cmdletbinding()]
663     [Alias("Write-LogFileSummaryInformation")]
664     Param(
665         [string]$FilePath = $null, #A null path will just put this information on the
screen
666         [switch]$ComputerName, #true = include computer name in log file
667         [switch]$ExcludedUsers, #true = display list of users excluded from processing
668         [string]$Summary = $null #if not null output summary
669     )
670
671     $dateTime = Get-Date -Format G
672
673     if ([string]::IsNullOrEmpty($FilePath) -or ((Test-Path -Path $FilePath) -eq
>false))
674     {
675         Write-Verbose $("*----- "+$dateTime+"-----*") | Out-Host
676         Write-ColorOutput -Message "File Path is Empty" -ForegroundColor "Green"
-Verbose | Out-Host

```

```
677
678     if ($ComputerName -eq $true)
679     {
680         Write-Verbose $("Computer Name: " + $env:computername) | Out-Host
681     }
682
683     if ($ExcludedUsers -eq $true)
684     {
685         Write-Verbose $("Excluded Users: ") | Out-Host
686     }
687
688     if (![string]::IsNullOrEmpty($Summary))
689     {
690         Write-Verbose $("Summary: " + $Summary) | Out-Host
691     }
692 }
693
694 else
695 {
696
697     Write-Verbose ("*----- $dateTime -----*`n`r") | Out-Host
698     Add-Content -Value ("*----- $dateTime -----*`n`r") -Path
$FilePath
699
700     Write-Verbose $("File: " + $FilePath) | Out-Host
701     Add-Content -Value "File: $FilePath`n`r" -Path $FilePath
702
703     if ($ComputerName -eq $true)
704     {
705         Write-Verbose $("Computer Name: " + $env:computername) | Out-Host
706         Add-Content -Value "`r`nComputerName: $env:computername`n`r" -Path $FilePath
707     }
708
709     if ($ExcludedUsers -eq $true)
710     {
711         Write-Verbose $("Excluded Users: ") | Out-Host
712         Add-Content -Value "`r`nExcluded Users: `r`n" -Path $FilePath
713     }
714
715     if (![string]::IsNullOrEmpty($Summary))
716     {
717         Write-Verbose $("Summary: " + $Summary) | Out-Host
718         Add-Content -Value "Summary:`r`n$Summary" -Path $FilePath
719     }
720 }
721 }
722
723 Function Send-HSCEmail
724 {
725     <#
726     .DESCRIPTION
727     The purpose of this function is to serve as a wrapper for the Send-MailMessage
728     cmdlet. This is done to handle
729     decrypting the encrypted password file which is needed to relay mail with Send-
730     MailMessage.
```

```
729
730 .NOTES
731     Written by: Jeff Brusoe
732     Last Updated by: Jeff Brusoe
733     Last Updated: April 16, 2018
734
735     This function probably isn't needed anymore. Need to see if it is still being use
before removing it though.
736     #>
737
738     [CmdletBinding()]
739     [Alias("Send-Email")]
740     Param (
741         [string[]]$To,
742         [string]$From,
743         [string]$Subject,
744         [string]$MessageBody,
745         [string[]]$Attachments,
746         [string]$SMTPServer = "Hssmtp.hsc.wvu.edu"
747     )
748
749     Write-Verbose "Preparing to send email..." | Out-Host
750     $Error.Clear()
751
752     try
753     {
754         Send-MailMessage -to $To -From $From -SMTPServer $SMTPServer -Subject $Subject
-UseSSL -port 587 -Attachments $Attachments -Body $MessageBody -ErrorAction Stop
755     }
756     catch
757     {
758         Write-Warning "Unable to send email message" | Out-Host
759     }
760 }
761
762 function Get-HSCPasswordFromSecureStringFile
763 {
764     <#
765         .SYNOPSIS
766         The purpose of this function is to decrypt a secure string file to handle user
authentication to Office 365
767         or other HSC protected environments.
768
769         .DESCRIPTION
770         This function decrypts a secure string file in order to use that for
authentication. In order to decrypt it,
771         the file must have been encrypted on the same machine with the same logged in
user used as the one being used
772         for decryption. There are also options to change the secure string file as wel
as prompt the user for credentials.
773
774         .PARAMETER Prompt
775         Causes the function to prompt the user for credentials instead of reading them
from a file.
776
```

```
777 .PARAMETER ChangeSecureStringFile
778     Writes a new secure string file
779
780 .PARAMETER EncryptedFileDirectory
781     This parameter is the path to the encrypted files directory. It defaults to:
782     C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-Repository\HSCCustomModules
\EncryptedFiles\.
783
784 .PARAMETER PWFile
785     The actual name of the password file to be decrypted.
786
787 .NOTES
788     Written by: Jeff Brusoe
789     Last Updated by: Jeff Brusoe
790     Last Updated: June 23, 2020
791 #>
792
793 [CmdletBinding()]
794 [Alias("Get-PasswordFromSecureStringFile")]
795 param (
796     [bool]$Prompt=$false,
797     [bool]$ChangeSecureStringFile=$false,
798     [string]$EncryptedFileDirectory = "C:\Users\microsoft\Documents\GitHub\HSC-
PowerShell-Repository\HSCCustomModules\EncryptedFiles\",
799     [string]$PWFile = "normal2.txt" #Mandatory
800 )
801
802 begin
803 {
804     [string]$Password=$null
805     $PWFile = $EncryptedFileDirectory + $PWFile
806
807     Write-Verbose "Password File Path: $PWFile" | Out-Host
808 }
809
810 process
811 {
812     if ($ChangeSecureStringFile)
813     {
814         try
815         {
816             Read-Host "Enter Current Password" -AsSecureString | ConvertFrom-SecureString
| Out-File $PWFile
817             Write-HSCColorOutput -foregroundcolor "Green" -Message "Successfully updated
secure string file.`n" | Out-Host
818         }
819         catch
820         {
821             $Prompt = $false
822             Write-Error "There was an error generating the secure string file.`n" | Out-
Host
823         }
824     }
825
826     if ($Prompt)
```



```
827     {
828         $Password = Read-Host "Enter Password" | Out-Host
829     }
830     else
831     {
832         if (Test-Path $PasswordFile)
833         {
834             Write-HSCColorOutput -ForegroundColor "Green" -Message "Decrypting
Password..." | Out-Host
835
836             try
837             {
838                 $securestring = convertto-securestring -string (get-content $PWFile)
839                 $bstr =
[System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($securestring)
840                 $Password = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr)
841
842                 Write-HSCColorOutput -ForegroundColor "Green" -Message "Password decrypted
successfully." | Out-Host
843             }
844             catch
845             {
846                 Write-Error "There was an error decrypting the password. Exiting file." |
Out-Host
847             }
848         }
849     }
850 } #End process block
851
852 end
853 {
854     return $Password
855 }
856 }
857
858 function Get-HSCRandomPassword
859 {
860     <#
861     .SYNOPSIS
862         The purpose of this function is to generate a random password.
863
864     .DESCRIPTION
865         The password generated meets WVU password complexity requirements:
866         1. Must be between 8 and 20 characters in length.
867         2. Must contain characters from at least three of the following four categories:
868             a. Uppercase letters: A-Z
869             b. Lowercase letters: a-z
870             c. Numbers: 0-9
871             d. Only these special characters: ! ^ ? : . ~ - _
872
873     .NOTES
874         Written by: Jeff Brusoe
875         Last Updated by: Jeff Brusoe
876         Last Updated: June 23, 2020
877     #>
```

```
878
879 [CmdletBinding()]
880 [Alias("Get-RandomPassword")]
881 [OutputType([string])]
882 param (
883     [int]$PasswordLength = 19
884 )
885
886 begin
887 {
888     Write-Verbose "Generating random password..." | Out-Host
889 }
890
891 process
892 {
893     try
894     {
895         #https://blogs.technet.microsoft.com/undocumentedfeatures/2016/09/20/powershell
random-password-generator/
896         [string]$Password = ([char[]]([char]33..[char]95) + ([char[]]([char]97..
[char]126)) + 0..9 | Sort-Object {Get-Random})[0..$PasswordLength] -join ''
897
898         Write-Verbose "Password: $Password" | Out-Host
899     }
900     catch
901     {
902         Write-Warning "Error generating random password" | Out-Host
903         [string]$Password = $null
904     }
905     finally
906     {
907         Write-Verbose "Done generating random password" | Out-Host
908     }
909 }
910
911 end
912 {
913     return $Password
914 }
915 }
916
917 function Exit-HSCCommand
918 {
919     <#
920     .DESCRIPTION
921         This function is called to handle error conditions where a PS file should exit.
922
923     .NOTES
924         Written by: Jeff Brusoe
925         Last Updated by: Jeff Brusoe
926         Last Updated: June 4, 2020
927     #>
928
929     [CmdletBinding()]
930     [Alias("Exit-Commands")]
```

```
931 [Alias("Exit-Command")]
932 param ()
933
934 #To do: Display way program is stopping (Complete, error & location, etc.)
935 process
936 {
937     try
938     {
939         Write-Verbose "Stopping Transcript" | Out-Host
940         Stop-Transcript -ErrorAction Stop
941     }
942     catch
943     {
944         Write-Verbose "Unable to stop transcript" | Out-Host
945     }
946     finally
947     {
948         Write-Verbose "Exiting file" | Out-Host
949     }
950 }
951
952 end
953 {
954     exit
955 }
956 }
957
958 function Test-HSCValidWVUEmail
959 {
960     <#
961     .SYNOPSIS
962         This function tests whether an email is a valid WVU email address. It only checks
963         if
964         it's possible, but not that the account actually exists.
965     .DESCRIPTION
966         There are currently only two tests to determine if the email address is valid.
967         1. Does the email address contain wvu.edu?
968         2. Is there an @ symbol in the email address string
969     .PARAMETER EmailAddress
970         This email address is what will be tested by the logic of hte code.
971     .OUTPUTS
972         Returns a boolean value to indicate whether the email address is valid
973     .NOTES
974         Written by: Jeff Brusoe
975         Last Updated by: Jeff Brusoe
976         Last Updated: June 23, 2020
977     #>
978
979 [CmdletBinding()]
980 [Alias("Test-ValidWVUEmail")]
981 [OutputType([bool])]
```

```
985 param (
986     [Parameter(Mandatory=$True)][string]$EmailAddress
987 )
988
989 begin
990 {
991     $ValidEmail = $false
992 }
993
994 process
995 {
996     Write-Verbose "Attempting to verify: $EmailAddress" | Out-Host
997
998     if (($EmailAddress.IndexOf("wvu.edu") -gt 0) -AND ($EmailAddress.IndexOf("@")
999 -gt0))
1000     {
1001         Write-Verbose "The email is valid" | Out-Host
1002         $ValidEmail = $true
1003     }
1004     else
1005     {
1006         Write-Verbose "The email is invalid" | Out-Host
1007     }
1008 }
1009
1010 end
1011 {
1012     return $ValidEmail
1013 }
1014
1015 function Get-HSCServerName
1016 {
1017     <#
1018     .SYNOPSIS
1019         This function returns the name of the server currently running the ps1 file.
1020
1021     .PARAMETER MandatoryServerNames
1022         This paramter tells the function to only return the server name if the name is
1023         included
1024         in the $AllowedServerNames array. Currently, this array contains the three
1025         sysscript
1026         servers (sysscript2, sysscript3, and sysscript4).
1027
1028     .OUTPUTS
1029         Returns the server name as a string. If the server name can't be determined, ther
1030         it returns $null.
1031
1032     .EXAMPLE
1033         Get-HSCServerName
1034         <return server name>
1035
1036     .EXAMPLE
1037         Get-HSCServeName -MandatoryServerNames
1038         - sysscript2 (if on sysscript2)
```

```
1037     - $null if not on sysscript2, 3, or 4
1038
1039 .NOTES
1040     Written by: Jeff Brusoe
1041     Last Updated by: Jeff Brusoe
1042     Last Updated: June 23, 2020
1043 #>
1044
1045 [CmdletBinding()]
1046 [Alias("Get-ServerName")]
1047 [OutputType([string])]
1048 param(
1049     [switch]$MandatoryServerNames
1050 )
1051
1052 begin
1053 {
1054     $AllowedServerNames = @("sysscript2","sysscript3","sysscript4")
1055 }
1056
1057 process
1058 {
1059     try
1060     {
1061         [string]$ServerName = (Get-ChildItem env:computername).Value
1062         Write-Verbose "Server Name: $ServerName" | Out-Host
1063
1064         if ($MandatoryServerNames -AND $AllowedServerNames -contains $ServerName)
1065         {
1066             return $ServerName
1067         }
1068         elseif ($MandatoryServerNames -AND $AllowedServerNames -notcontains $ServerName)
1069         {
1070             Write-Warning "Server name is not in server name array" | Out-Host
1071             return $null
1072         }
1073         elseif ([string]::IsNullOrEmpty($ServerName))
1074         {
1075             Write-Warning "Error retrieving server name" | Out-Host
1076             return $null
1077         }
1078         else
1079         {
1080             return $ServerName
1081         }
1082     }
1083     catch
1084     {
1085         Write-Warning "Error retrieving server name" | Out-Host
1086         return $null
1087     }
1088 } #end process
1089 }
1090
1091 function Get-HSCEncryptedFilePath
```

```
1092 {
1093     <#
1094     .SYNOPSIS
1095         Returns the path to the encrypted files used to establish 0365 tenant connectic
1096
1097     .NOTES
1098         Written by: Jeff Brusoe
1099         Last Updated: June 16, 2020
1100     #>
1101
1102     [CmdletBinding()]
1103     [Alias("Get-EncryptedFilePath")]
1104     [OutputType([String])]
1105     param (
1106         [ValidateSet("sysscript2", "sysscript3", "sysscript4")]
1107         [string]$ServerName = (Get-HSCServerName -MandatoryServerName)
1108     )
1109
1110     if ([string]::IsNullOrEmpty($ServerName))
1111     {
1112         Write-Warning "Unable to get encrypted file path" | Out-Host
1113         return $null
1114     }
1115
1116     try
1117     {
1118         Write-Output "`n`nServer Name: $ServerName" | Out-Host
1119
1120         $ServerNumber = $ServerName.substring($ServerName.Length - 1)
1121         Write-Verbose "ServerNumber: $ServerNumber" | Out-Host
1122
1123         $EncryptedFilePath = "C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-
Repository\1HSCCustomModules\EncryptedFiles\normal$ServerNumber.txt"
1124
1125         Write-Output "Encrypted File Path: $EncryptedFilePath" | Out-Host
1126
1127         return $EncryptedFilePath
1128     }
1129     catch
1130     {
1131         Write-Warning "Unable to get encrypted file path" | Out-Host
1132         return $null
1133     }
1134 }
1135
1136 function New-HSCSecureStringFile
1137 {
1138     <#
1139     .SYNOPSIS
1140         Creates a new secure string file.
1141
1142     .NOTES
1143         Written by: Jeff Brusoe
1144         Last Updated: June 18, 2020
1145     #>
```

```
1146
1147 [CmdletBinding()]
1148 [OutputType([bool])]
1149 param(
1150     [string]$OutputFilePath = "C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-
Repository\1HSCCustomModules\EncryptedFiles\",
1151     [parameter(Mandatory=$true)]
1152     [string]$UserName
1153 )
1154
1155 if (Test-Path $OutputFilePath)
1156 {
1157     try
1158     {
1159         Read-Host "Enter Current Password" -assecurestring | ConvertFrom-SecureString |
Out-File "$OutputFilePath\$UserName.txt"
1160         Write-Verbose "Successfully updated secure string file." | Out-Host
1161     }
1162     catch
1163     {
1164
1165         Write-Warning "There was an error generating the secure string file." | Out-Hos
1166     }
1167 }
1168 else
1169 {
1170     Write-Warning "File output path doesn't exist" | Out-Host
1171     return $false
1172 }
1173 }
1174
1175 #####
1176 # Export functions #
1177 #####
1178
1179 #Exit Modules
1180 Export-ModuleMember -Function "Exit-HSCCommand" -Alias "Exit-Commands","Exit-Command"
1181
1182 #Get Modules
1183 Export-ModuleMember -Function "Get-HSCServerName" -Alias "Get-ServerName"
1184 Export-ModuleMember -Function "Get-HSCEncryptedFilePath" -Alias "Get-
EncryptedFilePath"
1185 Export-ModuleMember -Function "Get-HSCPasswordFromSecureStringFile" -Alias "Get-
PasswordFromSecureStringFile"
1186 Export-ModuleMember -Function "Get-HSCParameter" -Alias "Get-Parameter"
1187 Export-ModuleMember -Function "Get-HSCLogFileName" -Alias "Get-LogFileName"
1188 Export-ModuleMember -Function "Get-HSCRandomPassword" -Alias "Get-RandomPassword"
1189
1190 #New-Modules
1191 Export-ModuleMember -Function "New-HSCSecureStringFile"
1192
1193 #Remove Modules
1194 Export-ModuleMember -Function "Remove-HSCOldLogFile" -Alias "Remove-OldLogFiles"
1195
1196 #Send Modules
```

```
1197 Export-ModuleMember -Function "Send-HSCEmail" -Alias "Send-Email"
1198
1199 #Set Modules
1200 Export-ModuleMember -Function "Set-HSCEnvironment" -Alias "Set-Environment"
1201 Export-ModuleMember -Function "Set-HSCWindowTitle" -Alias "Set-WindowTitle"
1202
1203 #Start Modules
1204 Export-ModuleMember -Function "Start-HSCCountdown" -Alias "Start-Countdown"
1205
1206 #Test Modules
1207 Export-ModuleMember -Function "Test-HSCValidWVUEmail" -Alias "Test-ValidWVUEmail"
1208 Export-ModuleMember -Function "Test-HSCVerbose" -Alias "Test-Verbose"
1209 Export-ModuleMember -Function "Test-HSCLogFilePath" -Alias "Test-LogFilePath"
1210
1211 #Write Modules
1212 Export-ModuleMember -Function "Write-HSCColorOutput" -Alias "Write-ColorOutput"
1213 Export-ModuleMember -Function "Write-HSCLogFileSummaryInformation" -Alias "Write-
    LogFileSummaryInformation"
```