```
#-------------------------------------------------------------------------------------------
#Export-ToSOLE.ps1
#
#Originally Written by: Matt Logue
#
#Modified by: Jeff Brusoe
#
#Last Modified: May 7, 2020
#
#Version: 2.0
#
#Purpose: This file looks at Active directory.  Exports the users, obtains account status from Office365, and Exports it to SOLE SQL server
#
#This file assumes a connection to the HSC Office 365 tenant has been established. If it isn't, then it will
#look for the Connect-ToOffice365-MS3.ps1 file to attempt a connection.
#-------------------------------------------------------------------------------------------

<#
.SYNOPSIS
    This file looks at Active directory.  Exports the user info, obtains account status from Office365, and Exports it to SOLE SQL server

.DESCRIPTION
    Requires
    1. Connection to the HSC tenant (Get-AzureADUser etc)
    2. Connection to Exchange online and PowerShell cmdlets
    3. Misc SQL functions file
    4. HSC common code file

.PARAMETER sqlPasswordPath
    This is the encrypted file that contains the SQL Server password. It can only be decrypted by the user
    account that created the file on the server the file was created on.

.PARAMETER sqlDataSource
    The IP address for the SQL Server

.PARAMETER sqlUsername
    The username to access the DB on a SQL Server

.PARAMETER sqlDatabase

.PARAMETER sqlTable

.PARAMETER MinADUsers
    This is a safety parameter to make sure that enough AD users have been found before clearing out the expor table.


.NOTES
    Original Author: Matt Logue
    Last Updated: May 23, 2018

    Modified by: Jeff Brusoe
    Last Modified: May 7, 2020

    Version History
    * August 5, 2019
        - Change HospitalHIPAA to ext 14 instead of 15
        - Added new common HSC module parameters
    * September 18, 2019
        - Modified to work with GitHub
        - Used HSC SQL Module
```

```
61          - Removed SQL functions from code since they were already in the HSC SQL Module
62          - Applied HSC PowerShell template
63      * September 25, 2019
64          - Minor output cleanup
65          - Added error handling code
66  #>
67
68  [CmdletBinding()]
69  param (
70      #Common HSC module parameters
71      [switch]$NoSessionTranscript,
72      [string]$LogFilePath = "$PSScriptRoot\Logs",
73      [switch]$StopOnError, #$true is used for testing purposes
74      [int]$DaysToKeepLogFiles = 5, #this value used to clean old log files
75
76      #File specific parameters
77
        [string]$sqlPasswordPath="C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-Reposito
        ry\1HSC-PowerShell-Modules\sql3.txt",
78      [bool]$CanDelete = $true, #set to false for testing - used to clear contents of table
79      [string]$sqlDataSource = "sql01.hsc.wvu.edu", #SQL server name
80      [string]$sqlUsername = "itsnetworking", #userid for SQL database
81      [string]$sqlDatabase = "BannerData", #SQL database name
82      #[string]$sqlTable = "HSADExportPS",
83      [string]$sqlTable = "HSADExportTemp", #SQL Table
84      [int]$MinADUsers = 4500 #Ths is just a safety value
85      )
86
87  $Error.Clear()
88  Set-Location $PSScriptRoot
89
90  ##############################
91  #Load HSC PowerShell Modules#
92  ##############################
93
94  #Step 1: Build path to HSC PowerShell Modules
95  $PathToHSCPowerShellModules = $PSScriptRoot
96  $PathToHSCPowerShellModules =
    $PathToHSCPowerShellModules.substring(0,$PathToHSCPowerShellModules.lastIndexOf("\")+1)
97  $PathToHSCPowerShellModules += "1HSC-PowerShell-Modules"
98  Write-Output $PathToHSCPowerShellModules
99
100 #Step 2: Attempt to load common code module
101 $CommonCodeModule = $PathToHSCPowerShellModules + "\HSC-CommonCodeModule.psm1"
102 Write-Output "Path to common code module: $CommonCodeModule"
103 Import-Module $CommonCodeModule -Force -ArgumentList
    $NoSessionTranscript,$LogFilePath,$true,$DaysToKeepLogFiles
104
105 #Step 3: #Attempt to load HSC Office 365 Module
106 $Office365Module = $PathToHSCPowerShellModules + "\HSC-Office365Module-AzureAD.psm1"
107 Write-Output "Path to HSC Office 365 module: $Office365Module"
108 Import-Module $Office365Module -Force
109
110 #Step 4: Attempt to load HSC SQL Module
111 $SQLModule = $PathToHSCPowerShellModules + "\HSC-SQLModule-Ver2a.psm1"
112 Write-Output "Path to HSC SQL Module: $SQLModule"
113 Import-Module $SQLModule -Force
114
115 #Step 5: Attempt to load HSC Active Directory Module
116 $ADModule = $PathToHSCPowerShellModules + "\HSC-ActiveDirectoryModule.psm1"
117 Write-Output "Path to HSC Active Directory Module: $ADModule"
118 Import-Module $ADModule -Force
119
120 if ($Error.Count -gt 0)
121 {
122     #Any errors at this point are from loading modules. Program must stop.
123     Write-Warning "There was an error loading the required modules. Program is ending."
```

```
124         return
125     }
126
127     ###############################################################
128     #End of code block to load HSC PowerShell modules             #
129     #At this point, all modules should be loaded successfully.#
130     ###############################################################
131
132     ##############################
133     #Configure environment block#
134     ##############################
135     Write-Verbose "Getting Parameter Information"
136     Get-Parameter -ParameterList $PSBoundParameters
137
138     Set-Environment
139     Set-WindowTitle
140
141     #See this page to understand what is going on here.
142     #https://www.thecloudjournal.net/2016/07/create-your-own-powershell-module-for-exchange-online/
143     ConnectTo-Office365 #from Office 365 module
144     Import-Module ExchangeOnline -Force #comes from HSC-OFfice365Module.psm1
145
146     #Remove old CSV log files
147     Remove-OldLogFile -CSV -Path $LogFilePath -Days $DaysToKeepLogFiles -Verbose -Delete
148
149     #Decrypt SQL Password
150     $sqlSecureStringPassword = cat $sqlPasswordPath  | convertto-securestring
151     $sqlPassword =
        [System.Runtime.InteropServices.marshal]::PtrToStringAuto([System.Runtime.InteropServices.marshal]::SecureStringToBSTR($sqlSecureStringPassword))
152
153     if ($Error.Count -gt 0)
154     {
155         #Any errors at this point are from enivronment configuration. Program must stop.
156         Write-Warning "There was an error configuring the environment. Program is exiting."
157         return
158     }
159
160     ########################################
161     #End of environment configuration block#
162     ########################################
163
164     ##############################
165     #       MAIN PROGRAM         #
166     ##############################
167
168     $count = 0
169     $i=0
170     $users = @()
171
172     #Generate list of AD users
173     Write-ColorOutput "Getting User Accounts from Active Directory..." -ForegroundColor
        "Cyan"
174     $properties =
        "sAMAccountName","Enabled","DistinguishedName","userAccountControl","LockedOut","mail","givenName","initials","sn","Created","LastLogonDate","modified","extensionAttribute10","extensionAttribute11","extensionAttribute12","extensionAttribute3","proxyAddresses","extensionAttribute6","extensionAttribute14"
175
176     try
177     {
178         $users = Get-ADUser -Filter * -SearchScope Subtree -SearchBase
            "OU=HSC,DC=HS,DC=wvu-ad,DC=wvu,DC=edu" -Properties $properties -ErrorAction Stop |
            Where-Object {$_.extensionAttribute10 -ne "Resource"} | select $properties
179     }
180     catch
```

```
{
    Write-Warning "There was an error searcing AD. Program is exiting."
    Exit-Command
}

Write-Output $("AD User Count: " + $users.count)

if ($users.Count -lt $MinADUsers)
{
    Write-Warning "Too few AD users were returned. Program is exiting."
    Exit-Command
}
else
{
    Write-Output "Beginning to process AD users"
}

$MailboxEnabledFile = "$PSScriptRoot\Logs\" + (Get-Date -format yyyy-MM-dd-HH-mm) + "-MailboxesEnabled.csv"
New-Item -type file $MailboxEnabledFile -Force

$HaveMailboxesFile = "$PSScriptRoot\Logs\" + (Get-Date -format yyyy-MM-dd-HH-mm) + "-HaveMailboxes.csv"
New-Item -type file $HaveMailboxesFile -Force

#Calls function in HSC-CommonCodeModule to get Exchange/Office365 Info
Write-ColorOutput "Getting Office365Information..." -ForegroundColor "Cyan"
$Office365Info = Get-O365MailboxStatus -ExportFile $MailboxEnabledFile -Verbose

Write-ColorOutput "Getting Active Mailboxes" -ForegroundColor "Yellow"
$MailboxInfo = Get-Mailbox -ResultSize Unlimited | Where-Object {($_.MaxReceiveSize -like "*MB*") -AND ($_.PrimarySMTPAddress -notlike "*rni.*" -AND $_.PrimarySMTPAddress -notlike "*wvurni*")} | Select-Object UserPrincipalName,MaxReceiveSize
$MailboxInfo | Export-csv $HaveMailboxesFile
Write-Output "Active Mailbox Count:  $(($MailboxInfo.UserPrincipalName | Measure-Object).Count)"

foreach ($user in $users)
{
    Write-Output $("Current User: " + $user.SamAccountName)

    Write-ColorOutput "Editing User Information" -ForegroundColor "Cyan"

    #Add User OU
    Write-Output "Before Get-ADUserParentContainer"
    $UserOU = Get-ADUserParentContainer -User $user.SamAccountName
    Write-Output "User OU: $UserOU"
    $user | Add-Member -MemberType NoteProperty -Name "OU" -Value $UserOU

    #Adds values needed to each entry for SQL query
    $user | Add-Member -MemberType NoteProperty -Name "resource" -Value $false
    $user | Add-Member -MemberType NoteProperty -Name "unsure" -Value $false
    $user | Add-Member -MemberType NoteProperty -Name "ruby" -Value $false
    $user | Add-Member -MemberType NoteProperty -Name "clinic" -Value $false
    $user | Add-Member -MemberType NoteProperty -Name "student" -Value $false
    $user | Add-Member -MemberType NoteProperty -Name "ExchangeEnabled" -Value $false
    $user | Add-Member -MemberType NoteProperty -Name "HasMailbox" -Value $false

    #Modifies blank values to no entry
    if ([string]::IsNullOrEmpty($user.extensionAttribute10))
    {
           $user.extensionAttribute10 = "No Entry"
    }

    if ([string]::IsNullOrEmpty($user.extensionAttribute11))
    {
            $user.extensionAttribute11 = "No Entry"
```

```
243         }
244
245         if ([string]::IsNullOrEmpty($user.mail))
246         {
247                 $user.mail = "Not Found"
248         }
249
250         if ([string]::IsNullOrEmpty($user.extensionAttribute3))
251         {
252                 $user.extensionAttribute3 = "Not Found"
253         }
254
255         if ([string]::IsNullOrEmpty($user.extensionAttribute6))
256         {
257                 $user.extensionAttribute3 = "Not Found"
258         }
259
260         if ([string]::IsNullOrEmpty($user.extensionAttribute14))
261         {
262             $user.extensionAttribute14 = "0" #HSCHIPAA
263         }
264         elseif ($user.extensionAttribute14 -eq "HospitalHIPAA")
265         {
266             $user.extensionAttribute14 = "1"
267         }
268         else
269         {
270             $user.extensionAttribute14 = "0"
271         }
272
273         if ([string]::IsNullOrEmpty($user.givenName))
274         {
275                 $user.givenname = "Not Found"
276                 $user | Add-Member -MemberType NoteProperty -Name "FirstName" -Value "Not
                    Found"
277         }
278         else
279         {
280             $user | Add-Member -MemberType NoteProperty -Name "FirstName" -Value
                    $user.givenName
281         }
282
283         if ([string]::IsNullOrEmpty($user.sn))
284         {
285                 $user.sn = "Not Found"
286                 $user | Add-Member -MemberType NoteProperty -Name "LastName" -Value "Not
                    Found"
287         }
288         else
289         {
290             $user | Add-Member -MemberType NoteProperty -Name "LastName" -Value $user.sn
291         }
292
293         $user | Add-Member -MemberType NoteProperty -Name "LogonEnabled" -Value $false
294
295         if ($user.Enabled) {
296
297             $user.LogonEnabled = $true
298         }
299
300         #Looks for resource account (extensionAttribute10) value or student value
            (extensionAttribute6)
301         if ($user.extensionAttribute10.ToLower() -eq "resource")
302         {
303             $user.resource = $true
304         }
305         elseif ($user.extensionAttribute10.ToLower() -eq "unsure")
```

```
306         {
307             $user.unsure = $true
308         }
309         elseif ($user.extensionAttribute10.ToLower() -eq "ruby")
310         {
311             $user.ruby = $true
312         }
313         elseif ($user.extensionAttribute10.ToLower() -eq "clinic")
314         {
315             $user.clinic = $true
316         }
317 
318         if ($user.extensionAttribute6 -eq "STUDENT")
319         {
320             $user.student = $true
321         }
322 
323         #Looks for Office365 Enabled
324         $proxies = $user.proxyaddresses
325         $proxies = $proxies | where {$_ -like "*@hsc.wvu.edu"}
326         $proxies = $proxies -replace "smtp:"
327         $proxies = $proxies -replace "sip:"
328 
329         if ($Office365Info.O365EmailAddress -contains ($user.mail))
330         {
331             $user.ExchangeEnabled = $true
332         }
333         else
334         {
335             foreach ($proxy in $proxies)
336             {
337                 if ($Office365Info.O365EmailAddress -contains ($proxy))
338                 {
339                    $user.ExchangeEnabled = $true
340                 }
341             }
342         }
343 
344         #looks for mailbox based on username
345         if ($MailboxInfo.UserPrincipalName -contains "$($user.samaccountname)@hsc.wvu.edu")
346         {
347             $user.HasMailbox = $true
348         }
349 
350         Write-Output "********************"
351     } #end foreach to modify user info and adding items
352 
353         ############################
354         #         SQL Upload       #
355         ############################
356 
357         #SQL parameters defined at begining of script
358 
359         [string]$ConnectionString = $null
360         try
361         {
362             $ConnectionString = Get-ConnectionString -SQLPassword $sqlPassword -ErrorAction
                Stop
363         }
364         catch
365         {
366             Write-Warning "Unable to generate connection string. Program is exiting."
367             Exit-Command
368         }
369 
370 
371         if ($CanDelete)
```

```
    {
        $DeleteQuery = "DELETE from $sqlTable"
        Invoke-SqlCmd -Query $DeleteQuery -ConnectionString $ConnectionString
    }

foreach ($user in $users)
{
    #fixes apostrophe in LDAP string and last name for SQL query
    $LastName = $user.LastName
    $OU = $user.OU
    $FirstName = $user.FirstName
    $MiddleName = $user.Initials

    if (($user.OU -like "*'*"))
    {
        $OU = $OU -replace "'","''"
        $LastName = $LastName -replace "'","''"
        $FirstName = $FirstName -replace "'","''"
    }

    if ($user.LastName -like "*'*")
    {
        $LastName = $LastName -replace "'","''"
        #$OU = $OU -replace "'","''"
    }
    if ($user.FirstName -like "*'*")
    {
        $FirstName = $FirstName -replace "'","''"
        #$OU = $OU -replace "'","''"
    }
    if ($user.Initials -like "*'*")
    {
        $MiddleName = $MiddleName -replace "'","''"
    }

    $UpdateQuery = "INSERT INTO $sqlTable
    (samAccountName,OU,Enabled,ExchangeEnabled,FirstName,LastName,Email,AccountCreationDT
    M,LastLoginDTM,LastModifiedDTM,ResourceAccount,UID,Unsure,IsStudent,RubyAccount,Healt
    hCareProvider,MiddleName,HasMailbox,WvumHipaaTraining) VALUES
    ('$($user.samaccountname)','$OU','$($user.Enabled)','$($user.ExchangeEnabled)','$Firs
    tName','$LastName','$($user.mail)','$($user.Created)','$($user.LastLogonDate)','$($us
    er.modified)','$($user.resource)','$($user.ExtensionAttribute11)','$($user.unsure)','
    $($user.student)','$($user.ruby)','$($user.clinic)','$MiddleName','$($user.HasMailbox
    )','$($user.extensionAttribute14)')"

    $sqlQuery = Invoke-SqlCmd -Query $UpdateQuery -ConnectionString $ConnectionString

    Write-Output "`n************************"
    Write-Output "Inserted into SQL Database:`nFirst Name: $FirstName `nLast Name:
    $LastName `nUsername: $($user.samAccountName)`n Email: $($user.mail)`n Exchanged
    Enabled: $($user.HasMailbox)"
    Write-Output "************************`n"
}

Write-Output "User data uploaded to temp database"

try
{
    $SQLConn = Connect-SQL -SQLPassword $sqlPassword
    Write-Output "Executing Stored Procedure"
    $sqlspcmd = New-Object System.Data.SqlClient.SqlCommand
    $sqlspcmd.CommandText = "sp_RebuildHSADExport"
    $sqlspcmd.Connection = $SQLConn
    $SqlAdapter = New-Object System.Data.SqlClient.SqlDataAdapter
    $SqlAdapter.SelectCommand = $sqlspcmd
    $DataSet = New-Object System.Data.DataSet
    $SqlAdapter.Fill($DataSet)
```

```
}
catch
{
    Write-Warning "Error running sp_RebuildHSADExport"
}

try
{
    Write-Verbose "Attempting to close SQL Connection"
    $SQLConn.close()
    Write-Verbose "SQL Connection has been closed"
}
catch
{
    Write-Warning "Error closing SQL connection"
}

Exit
```