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

.DESCRIPTION
  Active Directory functions included in this module:

  1. Get-HSCLoggedOnUser
  2. Get-HSCPrimarySMTPAddress
  3. Set-HSCPasswordRequired

.NOTES
  HSC-ActiveDirectoryModule.psm1
  Last Modified by: Jeff Brusoe
  Last Modified: July 16, 2020

  Version: 1.0
#>

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

Import-Module ActiveDirectory

Function Get-HSCDirectoryMapping
{
  <#
    .SYNOPSIS
      This function takes a user's distinguished name for input and returns
      the path to the user's home directory.

    .NOTES
      Last Modified by: Jeff Brusoe
      Last Modified: July 17, 2020
  #>

  [CmdletBinding()]
  param (
    [Parameter(ValueFromPipeline = $true)]
    [string]$UserDN = $null,
    [switch]$DetermineFullPath
  )

  begin
  {
    #Create object to hold directory information
    $HomeDirectoryInfo = new-object PSObject

    $HomeDirectoryInfo | Add-Member -type NoteProperty -Name DirectoryPath -value $null
    $HomeDirectoryInfo | Add-Member -type NoteProperty -Name FullPath -Value $false
    #In cases where the DN still needs to be parsed up later recursively, the FullPath value is set to false.
    #If the DirectoryPath value is the correct (& final) home directory path, the FullPath value is set to true.

    if ($UserDN.indexOf("CN=") -ge 0)
```

```powershell
    {
      #Need to remove this from the DN
      $UserDN = $UserDN.substring($UserDN.indexOf(",")+1).Trim()
      Write-Verbose "Cleaned UserDN: $UserDN"
    }
  }

  process
  {
    #First check DirectoryMapping file for match
    [string]$HomeDirectoryPath = $null

    #Step 1: Check against home directory mapping file.
    #$HomeDirectoryMappings = Import-Csv $($MyInvocation.PSScriptRoot + "\HomeDirectoryMapping.csv")
    $HomeDirectoryMappings = Import-Csv "C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-Repository\Create-NewAccount\HomeDirectoryMapping.csv"
    #$HomeDirectoryMappings = Import-Csv "C:\HSCGitHub\HSC-PowerShell-Repository\Create-NewAccount\HomeDirectoryMapping.csv"

    [string]$HomeDirectoryPath = ($HomeDirectoryMappings | where {$UserDN -eq $_.UserDN}).DirectoryPath

    if ([string]::IsNullOrEmpty($HomeDirectoryPath))
    {
      [string]$HomeDirectoryPath = ($HomeDirectoryMappings | where {$UserDN -match $_.UserDN}).DirectoryPath
    }

    if ([string]::IsNullOrEmpty($HomeDirectoryPath))
    {
      Write-verbose "No match from directory mapping file"
    }
    else {
      $HomeDirectoryInfo.DirectoryPath = $HomeDirectoryPath
      $HomeDirectoryInfo.FullPath = $true
    }

    #Step 2: Check against predefined mappings
    [string]$ParentPath = $null

    if ([string]::IsNullOrEmpty($HomeDirectoryPath))
    {
      switch -wildcard ($UserDN)
      {
        "*OU=Char_Div*"
        {
          $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\Char_Div\"
          $HomeDirectoryInfo.FullPath = $false
          $ParentPath = "Char_Div"
          break
        }
        "*OU=ITS*"
        {
          $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\ITS\"
          $HomeDirectoryInfo.FullPath = $false
          $ParentPath = "ITS"
          break
        }
        "*OU=ADMIN*"
    {
     $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\Admin\"
     $HomeDirectoryInfo.FullPath = $false
     $ParentPath = "ADMIN"
     break
    }
    "*OU=BASSCI*"
    {
     $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\bassci\"
     $HomeDirectoryInfo.FullPath = $false
     break
    }
    "*OU=MBRCC*"
    {
     $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\mbrcc\"
     $HomeDirectoryInfo.FullPath = $false
     break
    }
    "*OU=SOM*"
    {
     $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\som\"
     $HomeDirectoryInfo.FullPath = $false
     break
    }
    "*OU=SON*"
    {
     $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\son\"
     $HomeDirectoryInfo.FullPath = $false
     break
    }
    "*OU=SOP*"
    {
     $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\sop\"
     $HomeDirectoryInfo.FullPath = $false
     break
    }
    "*OU=SPH*"
    {
     $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\sph\"
     $HomeDirectoryInfo.FullPath = $false
     break
    }
    default
    {
     $HomeDirectoryInfo.DirectoryPath = "NoHomeDirectory"
     $HomeDirectoryInfo.FullPath = $true
    }
   }
  }

  #Step 3: Determine full path if switch not used
  if ($DetermineFullPath -AND !$HomeDirectoryInfo.FullPath -AND ($HomeDirectoryInfo -ne "NoHomeDirectory"))
  {
   $ParsedDN = $UserDN -split ","
   $ParsedDNCount = $ParsedDN.Length
   $AddToPath = $false

   for ($i=$ParsedDNCount-1; $i -ge 0; $i--)
   {
```

```powershell
        $ParsedDN[$i] = $ParsedDN[$i] -replace "OU=",""

        if ($ParsedDN[$i] -eq $ParentPath)
        {
          $AddToPath = $true
        }
        elseif ($AddToPath)
        {
          $HomeDirectoryInfo.DirectoryPath = $HomeDirectoryInfo.DirectoryPath + "\" + $ParsedDN[$i]+ "\"
        }
      }
      $HomeDirectoryInfo.FullPath = $true
    }
  }

  end
  {
    return $HomeDirectoryInfo
  }
}

function Get-HSCLoggedOnUser
{
  <#
  .SYNOPSIS
    This function returns the currently logged on user.

  .OUTPUTS
    Returns a PSCustomObject that has two properties: Logged on username and domain.

  .NOTES
    Tested with the following version of PowerShell:
    1. 5.1.18362.752
    2. 7.0.2

    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: June 24, 2020
  #>

  [CmdletBinding()]
  [OutputType([PSCustomObject])]
  param()

  try
  {
    $LoggedOnUser = [PSCustomObject]@{
      UserName = $((Get-ChildItem Env:\USERNAME).Value)
      Domain = $((Get-ChildItem Env:\USERDOMAIN).Value)
    }

    return $LoggedOnUser
  }
  catch
  {
    Write-Warning "Error getting logged on user" | Out-Null

    return $null
  }
}

function Get-HSCPrimarySMTPAddress
{
  <#
  .SYNOPSIS
    This function retrieves the primary SMTP address for AD users.

  .INPUTS
    This function can take a string(array) or ADUser object(array) and will
    get the primary SMTP address for those users.

  .PARAMETER UserNames
    This parameter takes a string array of users names as input. It will attempt to
get
    the primary SMTP address after finding the users.

  .PARAMETER ADUsers
    Similar to UserNames, but this paramter takes an array of ADUsers
(Microsoft.ActiveDirectory.Management.ADAccount)
    and attempts to get their primary SMTP address.


  .EXAMPLE
    PS C:\Windows\system32> "jbrusoe","krussell" | Get-HSCPrimarySMTPAddress

    SamAccountName PrimarySMTPAddress
    -------------- ------------------
    jbrusoe        jbrusoe@hsc.wvu.edu
    krussell       krussell@hsc.wvu.edu

  .EXAMPLE
    PS C:\Windows\system32> $Jeff = Get-ADUser jbrusoe -Properties proxyAddresses
    PS C:\Windows\system32> $Kevin = Get-ADUser krussell -Properties proxyAddresses
    PS C:\Windows\system32> $Jeff,$Kevin | Get-HSCPrimarySMTPAddress

    SamAccountName PrimarySMTPAddress
    -------------- ------------------
    jbrusoe        jbrusoe@hsc.wvu.edu
    krussell       krussell@hsc.wvu.edu

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

  [CmdletBinding()]
  [OutputType([PSObject])]

  param (
    [Parameter(ValueFromPipeline=$true,
      ParameterSetName="ADUserArray",
      Mandatory=$true,
      Position=0)]
    [Microsoft.ActiveDirectory.Management.ADAccount[]]$ADUsers,

    [Parameter(ValueFromPipeline=$true,
      ParameterSetName="UserNameArray",
      Mandatory=$true,
      Position=0)]
    [string[]]$UserNames,

    [switch]$NoOutput
  )

  begin
  {
    [psobject[]]$PrimarySMTPAddresses = @()
  }

  process
  {
    Write-Debug $("In process block - Parameter Set Name: " + 
$PSCmdlet.ParameterSetName)

    #Get array of ADUsers if a string array is passed in
    if ($PSCmdlet.ParameterSetName -eq "UserNameArray")
    {
      $ADusers = $null
      foreach ($UserName in $UserNames)
      {
        try
        {
          $ADUsers += Get-ADUser $UserName -Properties proxyAddresses -ErrorAction 
Stop
          Write-Verbose "Found User: $UserName"
        }
        catch
        {
          Write-Warning "Unable to find user name"

          $ADUserObject = New-Object -TypeName PSObject
          $ADUserObject | Add-Member -MemberType NoteProperty -Name 
"SamAccountName" -Value $UserName
          $ADUserObject | Add-Member -MemberType NoteProperty -Name 
"PrimarySMTPAddress" -Value "UserNotFound"

          $PrimarySMTPAddresses += $ADUserObject
        }
      }
    }

    if ($null -ne $ADUsers)
    {
      foreach ($ADUser in $ADUsers)
      {
        $ADUserObject = New-Object -TypeName PSObject
        $ADUserObject | Add-Member -MemberType NoteProperty -Name "SamAccountName" 
-Value $ADUser.SamAccountName


        [string[]]$ProxyAddresses = $ADUser.proxyAddresses

        [string]$PrimarySMTPAddress = $ProxyAddresses -cmatch "SMTP:"

        if ([string]::IsNullOrEmpty($PrimarySMTPAddress))
        {
          Write-Verbose "Primary SMTP Address isn't defined"
```

```powershell
          $ADUserObject | Add-Member -MemberType NoteProperty -Name "PrimarySMTPAddress" -Value $null
        }
        else {
          Write-Verbose $("Current User" + $ADUser.SamAccountName)
          Write-Verbose "Primary SMTP Address: $PrimarySMTPAddress"

          $PrimarySMTPAddress = ($PrimarySMTPAddress -replace "SMTP:","").Trim()

          $ADUserObject | Add-Member -MemberType NoteProperty -Name "PrimarySMTPAddress" -Value $PrimarySMTPAddress
        }

        $PrimarySMTPAddresses += $ADUserObject
      }
    }
    else
    {
      Write-Warning "ADUser object is null"
    }
  }

  end
  {
    return $PrimarySMTPAddresses
  }
}

Function Set-HSCPasswordRequired
{
  <#
  .SYNOPSIS
    This function sets the password required for a user

  .INPUTS
    This function can take a string(array) or ADUser object(array) and will
    set the password required attribute for those users.

  .PARAMETER UserNames
    This parameter takes a string array of users names as input. It will attempt to
    set the password required attribute on all of these users.

  .PARAMETER ADUsers
    Similar to UserNames, but this paramter takes an array of ADUsers (Microsoft.ActiveDirectory.Management.ADAccount)
    and attempts to set the password required field on them.

  .PARAMETER NoOutput
    This is a switch parameter that prevents displaying function output.

  .EXAMPLE
    PS C:\Windows\system32> "jbrusoe","krussell" | Set-HSCPasswordRequired
    Current user: jbrusoe
    Password Not Required: False
    *****************************
    Current user: krussell
    Password Not Required: False
    *****************************

  .EXAMPLE
```

```
    PS C:\Windows\system32>  $Jeff = Get-ADUser jbrusoe -Properties PasswordNotRequired
    PS C:\Windows\system32> $Kevin = Get-ADUser krussell -Properties PasswordNotRequired
    PS C:\Windows\system32> @($Jeff,$Kevin) | Set-HSCPasswordRequired
    Current user: jbrusoe
    Password Not Required: False
    *****************************
    Current user: krussell
    Password Not Required: False
    *****************************

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: July 13, 2020
  #>

  [CmdletBinding(SupportsShouldProcess=$true,
    ConfirmImpact="Medium")]

  param (
    [Parameter(ValueFromPipeline=$true,
      ParameterSetName="ADUserArray",
      Mandatory=$true,
      Position=0)]
    [Microsoft.ActiveDirectory.Management.ADAccount[]]$ADUsers,

    [Parameter(ValueFromPipeline=$true,
      ParameterSetName="UserNameArray",
      Mandatory=$true,
      Position=0)]
    [string[]]$UserNames,

    [switch]$NoOutput
  )

  begin
  {
    Write-Verbose "Beginning to set password required"

    $Error.Clear()

    if ($null -eq (Get-Module ActiveDirectory))
    {
      Write-Verbose "Importing Active Diretory Module"
    }
  }

  process
  {
    Write-Debug $("In process block - Parameter Set Name: " + $PSCmdlet.ParameterSetName)

    #Get array of ADUsers if a string array is passed in
    if ($PSCmdlet.ParameterSetName -eq "UserNameArray")
    {
      $ADusers = $null

      Write-Debug "Process Block - If Statement"
      foreach ($UserName in $UserNames)
      {
        try
        {
          $ADUsers += Get-ADUser $UserName -Properties PasswordNotRequired -
ErrorAction Stop
        }
        catch
        {
          Write-Warning "Unable to find user name"
        }
      }
    }

    foreach ($ADUser in $ADUsers)
    {
      if (!$NoOutput)
      {
        Write-Output $("Current user: " + $ADUser.SamAccountName) | Out-Host
        Write-Output $("Password Not Required: " + $ADUser.PasswordNotRequired) |
Out-Host
      }

      try
      {
        if ($PSCmdlet.ShouldProcess("Setting password required for " +
$ADUser.SamAccountName))
        {
          $ADUser | Set-ADUser -PasswordNotRequired $false -ErrorAction Stop
        }
      }
      catch
      {
          Write-Warning "There was an error setting the password not required
field"
      }

      if (!$NoOutput) {
        Write-Output "*****************************" | Out-Host
      }
    }
  }

  end
  {
    if ($Error.Count -gt 0)
    {
      $Error | FL
    }
    else
    {
      Write-Verbose "Password required has been set."
    }
  }
}


####################
# Export Functions #
```

```
####################

#Get Functions
Export-ModuleMember -Function "Get-HSCDirectoryMapping"
Export-ModuleMember -Function "Get-HSCLoggedOnUser"
Export-ModuleMember -Function "Get-HSCPrimarySMTPAddress"

#Set Functions
Export-ModuleMember -Function "Set-HSCPasswordRequired"
```