```
<#
  .SYNOPSIS
    This file is used to create HS domain and Office 365 accounts for users without
    needing a CSC request form filled out.

  .DESCRIPTION
    This file is used to create HS domain and Office 365 accounts for users without
    needing a CSC request form filled out. It does this by comparing the department
    as given by SailPoint to mapping file for accounts in the NewUsers OU.

    Currently, only four departments are in this mapping file: Family Medicine,
Opthamology,
    Eastern Division, and Anesthesiology.

    Once a matching user is found, the following steps are performed:
    1. Search NewUsersOU for any accounts with departments in the
       DepartmentMap.csv file.
    2. Determine correct OU to move user to
    3. Move user to correct OU (which is specified in mapping file)
    4. Refresh user object after AD move
    5. Create & assign rights to home directory.
    6. Add primary SMTP proxy address: ext15@hsc.wvu.edu (or @wvumedicine.org for
HVI)
    7. Add remaining proxy addresses:
      a. samaccountname@hsc.wvu.edu (if not already added in step 6)
      b. samaccountname@WVUHSC.onmicrosoft.com
      c. samaccountname@wvumedicine.org (Only for HVI users)
    8. Add SIP address
    9. Add user to groups
    10. Add Yes/No365 to ext7
    11. Set Name & Display Name
    12. Enable AD Account
    13. Email CSC

  .PARAMETER Testing
    This parameter is a flag to run -WhatIf commands instead of making any
    real changes to an account sitting in NewUsers.

  .PARAMETER LogsToSavePath
    The file runs every 15 minutes. Since most of the time no new accounts are
    created, this specifies where to copy the session transcript when a new account
    has been processed.

  .PARAMETER DepartmentMapPath
    This is the path to the DepartmentMap.csv file. This file maps the SailPoint
    department names with the destination OUs, groups, and values for ext7.

  .PARAMETER NewUsersOrgUnit
    The org unit that is to be searched for new users.

  .PARAMETER ADProperties
    This is the array of additional properties that will be pulled for AD user
objects.

  .NOTES
    Written by: Jeff Brusoe
    Last Updated: January 11, 2021

    To Be Done:
    1. Add code to create home directories for other departments.
    2. Add code for date offset
#>

[CmdletBinding()]
param (
  [switch]$Testing,

  [ValidateNotNullOrEmpty()]
  [string]$LogsToSavePath = "$PSScriptRoot\AccountCreatedLogs\",

  [ValidateNotNullOrEmpty()]
  [string]$DepartmentMapPath = "$PSScriptRoot\MappingFiles\DepartmentMap.csv",

  [ValidateNotNullOrEmpty()]
  [string]$NewUsersOrgUnit = "OU=NewUsers,DC=HS,DC=wvu-ad,DC=wvu,DC=edu",

  [ValidateNotNullOrEmpty()]
  [string[]]$ADProperties = @(
        "extensionAttribute11",
        "extensionAttribute15",
        "whenCreated",
        "PasswordLastSet",
        "Department"
      ),

  [ValidateNotNullOrEmpty()]
  [datetime]$StartDate = (Get-Date "1/21/2021 16:00")
)

#Initialize environment block
try {
  [string]$TranscriptLogFile = Set-HSCEnvironment

  if ($null -eq $TranscriptLogFile)
  {
    Write-Warning "There was an error configuring the environment. Program is
exiting."
    Invoke-HSCExitCommand -ErrorCount $Error.Count
  }

  Write-Verbose "Transcript Log file: $TranscriptLogFile"
  Write-Output "Start Date: $StartDate"

  $UserProcessed = $false
}
catch {
  Write-Warning "Error configuing environment"
  Invoke-HSCExitCommand -ErrorCount $Error.Count
}

#Read & process department mapping file
try {
  Write-Output "Opening Deparment Map File"
  $DepartmentMap = Import-Csv $DepartmentMapPath -ErrorAction Stop
}
catch {
  Write-Warning "Unable to open department map. Program is exiting"
  Invoke-HSCExitCommand -ErrorCount $Error.Count
}

```

```powershell
#Step 1: Search NewUsers for matching department names
try {
  Write-Output "`nStep1: Getting SailPoint deparmtents from department map file"

  $SPDepartments = $DepartmentMap.SailPointDepartmentName
  Write-Output "SailPoint Departments:"
  Write-Output $SPDepartments

  Write-Output "Searching AD NewUsers for matching department names"

  $GetADUserParams = @{
    SearchBase = $NewUsersOrgUnit
    Properties = $ADProperties
    Filter = "*"
    ErrorAction = "Stop"
  }
  $ADUsers = Get-ADUser @GetADUserParams |
    Where-Object {($SPDepartments -contains $_.Department) -AND
            ([datetime]$_.whenCreated -gt $StartDate)}
}
catch {
  Write-Warning "Unable to generate AD user list. Program is exiting."
  Invoke-HSCExitCommand -ErrorCount $Error.Count
}

if ($null -eq $ADUsers)
{
  Write-Warning "No matching departments were found"
  Invoke-HSCExitCommand -ErrorCount $Error.Count
}

Write-Output "`n**********************`n"

foreach ($ADUser in $ADUsers)
{
  $UserProcessed = $true
  Write-Output "User found in NewUsers Org Unit"
  Write-Output $("SamAccountName: " + $ADUser.SamAccountName)
  Write-Output $("Password Last Set: " + $ADUser.PasswordLastSet)
  Write-Output $("When Created: " + $ADUser.whenCreated)
  Write-Output "Distinguished Name:"
  Write-Output $ADUser.DistinguishedName

  #Step 2: Find OU to move user to
  Write-Output "`nStep2: Determine destination OU"

  $DepartmentOrgUnit = ($DepartmentMap |
    Where-Object {$_.SailPointDepartmentName -eq $ADUser.Department}).OUPath

  Write-Output "Destination OU From File: $DepartmentOrgUnit"

  try {
    Write-Output "Searching for Org Unit"
    $TargetOU = Get-ADOrganizationalUnit -Filter * -ErrorAction Stop |
      Where-Object {$_.DistinguishedName -like "*$DepartmentOrgUnit*"}

    Write-Output "AD Target OU:"
    Write-Output $TargetOU.DistinguishedName
  }
  catch {
    Write-Warning "Unable to find target OU in AD"
  }

  if ($null -eq $TargetOU)
  {
    Write-Warning "Unable to find destination OU in AD."
    Write-Output "`n***********************`n"
    continue
  }
  elseif (($TargetOU | Measure-Object).Count -gt 1)
  {
    Write-Warning "Multiple Org Units were found"
    Write-Output "`n***********************`n"
    continue
  }
  elseif (($TargetOU | Measure-Object).Count -eq 0)
  {
    Write-Warning "Unable to find the user's destination org unit"
    Write-Output "`n***********************`n"
    continue
  }
  else {
    Write-Output "One unique destination OU was found. User is being moved there."
  }

  #Step 3: Attempt user move
  try {
    Write-Output "Step 3: Attempting to move user to destination OU"

    $MoveADObjectParams = @{
      Identity = $ADUser.DistinguishedName
      TargetPath = $TargetOU
      ErrorAction = "Stop"
      WhatIf = $true
    }

    if ($Testing) {
      Move-ADObject @MoveADObjectParams
    }
    else {
      $MoveADObjectParams["WhatIf"] = $false

      Move-ADObject @MoveADObjectParams
    }

    Write-Output "Successfully moved user"
  }
  catch {
    Write-Warning "Unable to move user to target OU"
    Write-Output "`n***********************`n"
    continue
  }

  Start-HSCCountdown -Message "Delay to allow time for user move to sync" -Seconds
10

  #Step 4: Refresh AD user object after move
  try {
    Write-Output "`nStep 4: Generating new user object after move"

    $NewADUser = Get-ADUser -Filter * -Properties $Properties -ErrorAction Stop |
      Where-Object {$_.UserPrincipalName -eq $NewADUser.UserPrincipalName}
  }
  catch {
    Write-Output "Unable to find user object after move."
    continue
  }

  #Step 5: Create & add permissions for home directory
  #To Do: Add code here for other departments
  Write-Output "`nStep5: Creating home directory"

  $HomeDirectoryPath  = ($DepartmentMap |
              Where-Object {$_.SailPointDepartmentName -eq
$NewADUser.Department}).HomeDirectoryPath

  Write-Output "Home Directory Path: $HomeDirectoryPath"

  if ($HomeDirectoryPath -eq "NoHomeDirectory")
  {
    Write-Output $("Home Directory Path: " + $HomeDirectoryInfo.DirectoryPath)
    Write-Output "Not creating a home directory"
  }
  else {
    #Create home directory
    $UserHomeDirectory = $HomeDirectoryPath + "\" + $NewADUser.SamAccountName
    Write-Output "User Home Directory: $UserHomeDirectory"

    if ($Testing) {
      Write-Output "Home directory not being created because of -Testing parameter"
      New-Item -ItemType Directory -Path $UserHomeDirectory -WhatIf
    }
    else {
      try {
        New-Item -ItemType Directory -Path $UserHomeDirectory -ErrorActdion Stop
      }
      catch {
        Write-Warning "Error creating home directory: $UserHomeDirctory"
      }
    }

    Start-HSCCountdown -Message "Home Directory Created. Delay before adding
permissions." -Seconds 10

    #Now add file system permissions
    try {
      $UserName = "HS\" + $NewADUser.SamAccountName
      $Acl = (Get-Item $UserHomeDirectory -ErrorAction
Stop).GetAccessControl('Access')

      $Ar = New-Object
System.Security.AccessControl.FileSystemAccessRule($Username,
                                        'FullControl',
                                        'ContainerInherit,ObjectInherit',
                                        'None',
                                        'Allow'
                                        )

      $Acl.SetAccessRule($Ar)
      Set-Acl -Path $UserHomeDirectory -AclObject $Acl -ErrorAction Stop
```

```powershell
    }
    catch {
      Write-Warning "Unable to set ACL on home directory: $UserHomeDirectory"
    }
  }

  #Add Proxy addresses
  Write-Output "`nAdding ProxyAddresses"

  #Step 6: Set Primary SMTP Address
  Write-Output "`nStep 6: Adding Primary SMTP Address"

  if ($NewADUser.DistinguishedName.indexOf("HVI") -lt 0) {
    $PrimarySMTPAddress = $NewADUser.extensionAttribute15 + "@hsc.wvu.edu"
  }
  else {
    $PrimarySMTPAddress = $NewADUser.extensionAttribute15 + "@wvumedicine.org"
  }

  try {
    Write-Output "Setting Primary SMTP Address: $PrimarySMTPAddress"

    $PrimarySMTPAddress = "SMTP:" + $PrimarySMTPAddress

    if ($Testing) {
      $NewADUser |
        Set-ADUser -Add @{ProxyAddresses=$PrimarySMTPAddress} -ErrorAction Stop -WhatIf
    }
    else {
      $NewADUser |
        Set-ADUser -Add @{ProxyAddresses=$PrimarySMTPAddress} -ErrorAction Stop
    }

    Write-Output "Successfully set primary smtp address"

    $PrimarySMTPAddress = $PrimarySMTPAddress -replace "SMTP:",""
  }
  catch {
    #An error here will not allow other proxy addresses to be added.
    Write-Warning "Error setting primary smtp address"
  }

  Start-HSCCountdown -Message "Delay after adding primary SMTP address" -Seconds 5

  #Step 7a: Verify samaccountname@hsc.wvu.edu is a proxy address
  $SAMProxyAddress = $NewADUser.SamAccountName + "@hsc.wvu.edu"

  if ($SAMProxyAddress -ne $PrimarySMTPAddress)
  {
    Write-Output "`nStep 7a: Adding Proxy Address: $SAMProxyAddress"
    $SAMProxyAddress = "smtp:" + $SAMProxyAddress

    try {
      if ($Testing) {
        $NewADUser |
          Set-ADUser -Add @{ProxyAddresses = $SAMProxyAddress} -ErrorAction Stop -WhatIf
      }
      else {
        $NewADUser |
          Set-ADUser -Add @{ProxyAddresses = $SAMProxyAddress} -ErrorAction Stop
      }

      Write-Output "Succesfully set samaccountname@hsc proxy address"
    }
    catch {
      Write-Warning "Error setting samaccountname@hsc proxy address"
    }

    $SAMProxyAddress = $NewADUser.SamAccountName + "@hsc.wvu.edu"
  }
  else {
    Write-Output "samaccountname@hsc.wvu.edu is already a proxy address"
  }

  #Step 7b: Add samaccountname@wvuhsc.onmicrosoft.com
  try {
    $OnMicrosoftProxy = $NewADUser.samaccountname + "@WVUHSC.onmicrosoft.com"
    Write-Output "`nStep 7b: Adding onmicrosoft.com proxy: $OnMicrosoftProxy"

    $OnMicrosoftProxy = "smtp:" + $OnMicrosoftProxy

    if ($Testing) {
      $NewADUser |
        Set-ADUser -Add @{ProxyAddresses=$OnMicrosoftProxy} -ErrorAction Stop -
WhatIf
    }
    else {
      $NewADUser |
        Set-ADUser -Add @{ProxyAddresses=$OnMicrosoftProxy} -ErrorAction Stop
    }

    Write-Output "Successfully set onmicrosoft.com proxy address"

    $OnMicrosoftProxy = $OnMicrosoftProxy -replace "smtp:",""
  }
  catch {
    Write-Warning "Error setting onmicrosoft.com proxy address"
  }

  #Step 7c: Add samaccountname@hsc.wvu.edu for HVI users
  Write-Output "`nStep 7c: Settin samaccountname@hsc.wvu.edu for HVI users"

  if ($PrimarySMTPAddress.indexOf("wvumedicine.org") -gt 0)
  {
    $HVISAMProxyAddress = $SAMProxyAddress -replace "hsc.wvu.edu","wvumedicine.org"

    if ($HVISAMProxyAddress -ne $PrimarySMTPAddress) {
      Write-Output "Adding Proxy Address for HVI User: $HVISAMProxyAddress"
      $HVISAMProxyAddress = "smtp:" + $HVISAMProxyAddress

      try {
        if ($Testing) {
          $NewADUser |
            Set-ADUser -Add @{ProxyAddresses = $HVISAMProxyAddress} -ErrorAction
Stop -WhatIf
        }
        else {
          $NewADUser |
            Set-ADUser -Add @{ProxyAddresses = $HVISAMProxyAddress} -ErrorAction Stop
        }

        Write-Output "Successfully set HVI samaccountname@hsc.wvu.edu email address"
      }
      catch {
        Write-Warning "Unable to set HVI samaccountname@hsc.wvu.edu email address"
      }
    }
  }

  Start-HSCCountdown -Message "Delay to allow proxyaddresses to sync before adding SIP address" -Seconds 5

  #Step 8: Add SIP address
  try {
    $SIPAddress = "SIP:" + $NewADUser.SamAccountName + "@hsc.wvu.edu"
    Write-Output "`nStep 8: Adding SIP address: $SIPAddress"

    if ($Testing) {
      $NewADUser |
        Set-ADUser -Add @{ProxyAddresses = $SIPAddress} -ErrorAction Stop -WhatIf
    }
    else {
      $NewADUser |
        Set-ADUser -Add @{ProxyAddresses = $SIPAddress} -ErrorAction Stop
    }
  }
  catch {
    Write-Warning "Unable to add SIP address"
  }

  Start-HSCCountdown -Message "Delay after adding SIP address" -Seconds 5

  #Step 9: Add newly created user to groups
  try {
    Write-Output "`nStep 9: Adding user to groups"

    $UserDN = $NewADUser.DistinguishedName

    Write-Output "User DN: "
    Write-Output $UserDN

    if ($Testing) {
      Set-HSCGroupMembership -UserDN $UserDN -WhatIf
    }
    else {
      Set-HSCGroupMembership -UserDN $UserDN
    }
  }
  catch {
    Write-Warning "Unable to add user to groups"
  }

  #Step 10: Add Yes/No365 to ext7
  try {
    Write-Output "`nStep10: Setting extensionAttribute7"

    #Determine value of ext7
    $Ext7 = ($DepartmentMap |
          Where-Object {$_.SailPointDepartmentName -eq
$ADUser.Department}).CreateEmail

    if ($Ext7 -eq "Yes") {
      $Ext7 = "Yes365"
    }
    else {
      $Ext7 = "No365"
    }

    Write-Output "Setting ext7 to $Ext7"

    if ($Testing) {
      $NewADUser |
        Set-ADUser -Add @{extensionAttribute7=$Ext7} -ErrorAction Stop -WhatIf
    }
    else {
      $NewADUser |
        Set-ADUser -Add @{extensionAttribute7=$Ext7} -ErrorAction Stop
    }

  }
  catch {
    Write-Warning "Unable to set ext7"
  }

  Start-HSCCountdown -Message "Synchronization Delay" -seconds 2

  #Step 11: Set name and display name
  try {
    Write-Output "`nStep 11: Setting name and display name"

    $DisplayName = $NewADUser.Surname + ", " + $NewADUser.GivenName
    Write-Output "Display Name: $DisplayName`n"

    if ($Testing) {
      $NewADUser |
        Set-ADUser -DisplayName $DisplayName -ErrorAction Stop -WhatIf
    }
    else {
      $NewADUser |
        Set-ADUser -DisplayName $DisplayName -ErrorAction Stop
    }
  }
  catch {
    Write-Warning "Error setting display name"
  }

  #Step 12: Enable AD Account
  try {
    Write-Output $("`nStep 12: Enabling AD Account" + $NewADUser.SamAccountName)

    if ($Testing) {
      $NewADUser |
        Enable-ADAccount -ErrorAction Stop -WhatIf
    }
    else {
      $NewADUser |
        Enable-ADAccount -ErrorAction Stop
    }

    Write-Output "Successfully enabled account"
  }
  catch {
    Write-Warning "Unable to enable account"
  }

  # Step 13: Now send CSC email
  try {
    Write-Output "`nStep 13: Send CSC Confirmation Email"

    $SendNewAccountEmailParams = @{
      CSCEmail = $CSCEmail
      SamAccountName = $NewADUser.SamAccountName
      PrimarySMTPAddress = $PrimarySMTPAddress
      ErrorAction = "Stop"
    }

    if ($Testing) {
      Send-HSCNewAccountEmail @SendNewAccountEmailParams
    }
    else {
      $SendNewAccountEmailParams["CSCEmail"] = $CSCEmail
      Send-HSCNewAccountEmail @SendNewAccountEmailParams
    }
  }
  catch {
    Write-Warning "Unable to send new account email"
  }

  Write-Output "*************************"
}

if ($UserProcessed) {
  Write-Output "Copying session transcript to logs to save directory"

  try {
    Copy-Item -Path $TranscriptLogFile -Destination $LogsToSavePath -ErrorAction
Stop
  }
  catch {
    Write-Warning "Error copying files to backup directory"
  }
}

Invoke-HSCExitCommand -ErrorCount $Error.Count
```