```powershell
<#
  .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.

    Once a matching user is found, the following steps are performed:
    1. Search NewUsers OU for any accounts with departments in the
       DepartmentMap.csv file.
    1a. Verify if account has been claimed.
    2. Determine correct OU to move user to & CSC Email Address
    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. Add mailNickname field
    14. Set msexchHiddenFromAddressListsEnabled to $false
    15. 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: June 4, 2021
#>

[CmdletBinding()]
param (
```

```powershell
  [switch]$Testing,

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

  [ValidateNotNullOrEmpty()]
  [string]$DepartmentMapPath = (Get-HSCGitHubRepoPath) +
                  "2CommonFiles\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"
      )
)

#Initialize environment block
try {
  Write-Verbose "Initializing Environment"

  [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"

  $NewADUserCount = 0
  $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"
  Write-Output "Department Mapping File: $DepartmentMapPath"

  $DepartmentMap = Import-Csv $DepartmentMapPath -ErrorAction Stop
}
catch {
  Write-Warning "Unable to open department map. Program is exiting"
  Invoke-HSCExitCommand -ErrorCount $Error.Count
}

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

```

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

  Write-Output "`n`nSearching AD NewUsers for matching department names"

  $GetADUserParams = @{
    SearchBase = $NewUsersOrgUnit
    Properties = $ADProperties
    Filter = "*"
    SearchScope = "OneLevel"
    ErrorAction = "Stop"
  }

  #To do: Ensure that account is claimed
  $ADUsers = Get-ADUser @GetADUserParams |
    Where-Object { $SPDepartments -contains $_.Department }
}
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

  $NewADUserCount++
  Write-Output "New AD User Count: $NewADUserCount"

  #Step 1a: Verify user is claimed
  Write-Output "User was found in NewUsers org unit"
  Write-Output $("Password Last Set: " + $ADUser.PasswordLastSet)
  Write-Output $("When Created: " + $ADUser.whenCreated)

  #Test time diffrence between password last set and when created
  [datetime]$PasswordLastSet = $ADUser.PasswordLastSet
  [datetime]$WhenCreated = $ADUser.whenCreated

  if ($PasswordLastSet.AddMinutes(-2) -le $WhenCreated) {
    #Unclaimed account
    Write-Output "Account is unclaimed. CSC request will not be processed"
    Write-Output "************************"

    continue
  }
  else {
```

```powershell
    #Claimed Account
    Write-Output "Account has been claimed. CSC request will be processed"
  }

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

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

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

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

  try {
    Write-Output "Searching for Org Unit"
    $TargetOU = Get-ADOrganizationalUnit -Identity $DepartmentOrgUnit -ErrorAction
Stop

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

    continue
  }

  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"
```

```powershell
      WhatIf = $true
    }

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

      Move-ADObject @MoveADObjectParams
    }
    Start-HSCCountdown -Message "Delay to allow time for user move to sync" -Seconds 30

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

    continue
  }

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

    $LDAPFilter = "(sAMAccountName=" + $ADUser.SamAccountName + ")"
    Write-Verbose "LDAPFilter: $LDAPFilter"

    $GetADUserParams = @{
      LDAPFIlter = $LDAPFilter
      Properties = $ADProperties
      ErrorAction = "Stop"
    }

    $NewADUser = Get-ADUser @GetADUserParams
  }
  catch {
    Write-Warning "Unable to find user object after move."
    Write-Output "`n**********************`n"

    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 $ADUser.Department}).HomeDirectoryPath

  Write-Output "Home Directory Path: $HomeDirectoryPath"

  if ($HomeDirectoryPath -eq "NoHomeDirectory") {
    Write-Output $("Home Directory Path: " + $HomeDirectoryPath)
    Write-Output "Not creating a home directory"
  }
  else {
```

```
295     #Create home directory
296     $UserHomeDirectory = $HomeDirectoryPath + "\" + $NewADUser.SamAccountName
297     Write-Output "User Home Directory: $UserHomeDirectory"
298
299     if ($Testing) {
300       Write-Output "Home directory not being created because of -Testing parameter"
301       New-Item -ItemType Directory -Path $UserHomeDirectory -WhatIf
302     }
303     else {
304       try {
305         New-Item -ItemType Directory -Path $UserHomeDirectory -ErrorAction Stop
306       }
307       catch {
308         Write-Warning "Error creating home directory: $UserHomeDirctory"
309       }
310     }
311
312     Start-HSCCountdown -Message "Home Directory Created. Delay before adding
permissions." -Seconds 10
313
314     #Now add file system permissions
315     try {
316       $UserName = "HS\" + $NewADUser.SamAccountName
317       $Acl = (Get-Item $UserHomeDirectory -ErrorAction
Stop).GetAccessControl('Access')
318
319       $Ar = New-Object
System.Security.AccessControl.FileSystemAccessRule($Username,
320                                           'FullControl',
321                                           'ContainerInherit,ObjectInherit',
322                                           'None',
323                                           'Allow'
324                                           )
325
326       $Acl.SetAccessRule($Ar)
327       Set-Acl -Path $UserHomeDirectory -AclObject $Acl -ErrorAction Stop
328     }
329     catch {
330       Write-Warning "Unable to set ACL on home directory: $UserHomeDirectory"
331     }
332   }
333
334   #Add Proxy addresses
335   Write-Output "`nAdding ProxyAddresses"
336
337   #Step 6: Set Primary SMTP Address
338   Write-Output "`nStep 6: Adding Primary SMTP Address"
339
340   if ($NewADUser.DistinguishedName.indexOf("HVI") -lt 0) {
341     $PrimarySMTPAddress = $NewADUser.extensionAttribute15 + "@hsc.wvu.edu"
342   }
343   else {
344     $PrimarySMTPAddress = $NewADUser.extensionAttribute15 + "@wvumedicine.org"
345   }
346
347   try {
348     Write-Output "Setting Primary SMTP Address: $PrimarySMTPAddress"
349
350     $PrimarySMTPAddress = "SMTP:" + $PrimarySMTPAddress
351
```

```
352     if ($Testing) {
353       $NewADUser |
354         Set-ADUser -Add @{ProxyAddresses=$PrimarySMTPAddress} -ErrorAction Stop -
    WhatIf
355     }
356     else {
357       $NewADUser |
358         Set-ADUser -Add @{ProxyAddresses=$PrimarySMTPAddress} -ErrorAction Stop
359     }
360
361     Write-Output "Successfully set primary smtp address"
362
363     $PrimarySMTPAddress = $PrimarySMTPAddress -replace "SMTP:",""
364   }
365   catch {
366     #An error here will not allow other proxy addresses to be added.
367     Write-Warning "Error setting primary smtp address"
368   }
369
370   Start-HSCCountdown -Message "Delay after adding primary SMTP address" -Seconds 5
371
372   #Step 7a: Verify samaccountname@hsc.wvu.edu is a proxy address
373   $SAMProxyAddress = $NewADUser.SamAccountName + "@hsc.wvu.edu"
374
375   if ($SAMProxyAddress -ne $PrimarySMTPAddress)
376   {
377     Write-Output "`nStep 7a: Adding Proxy Address: $SAMProxyAddress"
378     $SAMProxyAddress = "smtp:" + $SAMProxyAddress
379
380     try {
381       if ($Testing) {
382         $NewADUser |
383           Set-ADUser -Add @{ProxyAddresses = $SAMProxyAddress} -ErrorAction Stop -
    WhatIf
384       }
385       else {
386         $NewADUser |
387           Set-ADUser -Add @{ProxyAddresses = $SAMProxyAddress} -ErrorAction Stop
388       }
389
390       Write-Output "Succesfully set samaccountname@hsc proxy address"
391     }
392     catch {
393       Write-Warning "Error setting samaccountname@hsc proxy address"
394     }
395
396     $SAMProxyAddress = $NewADUser.SamAccountName + "@hsc.wvu.edu"
397   }
398   else {
399     Write-Output "samaccountname@hsc.wvu.edu is already a proxy address"
400   }
401
402   #Step 7b: Add samaccountname@wvuhsc.onmicrosoft.com
403   try {
404     $OnMicrosoftProxy = $NewADUser.samaccountname + "@WVUHSC.onmicrosoft.com"
405     Write-Output "`nStep 7b: Adding onmicrosoft.com proxy: $OnMicrosoftProxy"
406
407     $OnMicrosoftProxy = "smtp:" + $OnMicrosoftProxy
408
409     if ($Testing) {
```

```
410       $NewADUser |
411         Set-ADUser -Add @{ProxyAddresses=$OnMicrosoftProxy} -ErrorAction Stop -
    WhatIf
412     }
413     else {
414       $NewADUser |
415         Set-ADUser -Add @{ProxyAddresses=$OnMicrosoftProxy} -ErrorAction Stop
416     }
417
418     Write-Output "Successfully set onmicrosoft.com proxy address"
419
420     $OnMicrosoftProxy = $OnMicrosoftProxy -replace "smtp:",""
421   }
422   catch {
423     Write-Warning "Error setting onmicrosoft.com proxy address"
424   }
425
426   #Step 7c: Add samaccountname@hsc.wvu.edu for HVI users
427   if ($PrimarySMTPAddress.indexOf("wvumedicine.org") -gt 0)
428   {
429     Write-Output "`nStep 7c: Setting samaccountname@hsc.wvu.edu for HVI users"
430
431     $HVISAMProxyAddress = $SAMProxyAddress -replace "hsc.wvu.edu","wvumedicine.org"
432
433     if ($HVISAMProxyAddress -ne $PrimarySMTPAddress) {
434       Write-Output "Adding Proxy Address for HVI User: $HVISAMProxyAddress"
435       $HVISAMProxyAddress = "smtp:" + $HVISAMProxyAddress
436
437       try {
438         if ($Testing) {
439           $NewADUser |
440             Set-ADUser -Add @{ProxyAddresses = $HVISAMProxyAddress} -ErrorAction
    Stop -WhatIf
441         }
442         else {
443           $NewADUser |
444             Set-ADUser -Add @{ProxyAddresses = $HVISAMProxyAddress} -ErrorAction
    Stop
445         }
446
447         Write-Output "Successfully set HVI samaccountname@hsc.wvu.edu email
    address"
448       }
449       catch {
450         Write-Warning "Unable to set HVI samaccountname@hsc.wvu.edu email address"
451       }
452     }
453   }
454
455   Start-HSCCountdown -Message "Delay to allow proxyaddresses to sync before adding
    SIP address" -Seconds 5
456
457   #Step 8: Add SIP address
458   try {
459     $SIPAddress = "SIP:" + $NewADUser.SamAccountName + "@hsc.wvu.edu"
460     Write-Output "`nStep 8: Adding SIP address: $SIPAddress"
461
462     if ($Testing) {
463       $NewADUser |
464         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 -ErrorAction Stop
    }
    else {
      Set-HSCGroupMembership -UserDN $UserDN -ErrorAction Stop
    }
  }
  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 ($Ext7 -eq "Yes") {
      if ($Testing) {
        Set-HSCADUserExt7 $NewADUser.SamAccountName -WhatIf
      }
      else {
        Set-HSCADUserExt7 $NewADUser.SamAccountName
      }
    }
    else {
      if ($Testing) {
        Set-HSCADUserExt7 $NewADUser.SamAccountName -WhatIf -No365
      }
      else {
        Set-HSCADUserExt7 $NewADUser.SamAccountName -No365
      }
    }
  }
  catch {
    Write-Warning "Unable to set ext7"
```

```
524   }
525
526   Start-HSCCountdown -Message "Synchronization Delay" -seconds 2
527
528   #Step 11: Set name and display name
529   try {
530     Write-Output "`nStep 11: Setting name and display name"
531
532     $DisplayName = $NewADUser.Surname + ", " + $NewADUser.GivenName
533     Write-Output "Display Name: $DisplayName`n"
534
535     if ($Testing) {
536       $NewADUser |
537         Set-ADUser -DisplayName $DisplayName -ErrorAction Stop -WhatIf
538     }
539     else {
540       $NewADUser |
541         Set-ADUser -DisplayName $DisplayName -ErrorAction Stop
542     }
543   }
544   catch {
545     Write-Warning "Error setting display name"
546   }
547
548   #Step 12: Enable AD Account
549   try {
550     Write-Output $("`nStep 12: Enabling AD Account: " + $NewADUser.SamAccountName)
551
552     if ($Testing) {
553       $NewADUser |
554         Enable-ADAccount -ErrorAction Stop -WhatIf
555     }
556     else {
557       $NewADUser |
558         Enable-ADAccount -ErrorAction Stop
559     }
560
561     Write-Output "Successfully enabled account"
562   }
563   catch {
564     Write-Warning "Unable to enable account"
565   }
566
567   #Step 13 & 14: Add mailNickname and Unhide account from address list
568   if ($Testing) {
569     Set-HSCADUserAddressBookVisibility -SAMAccountName $NewADUser.SAMAccountName -
    WhatIF
570   }
571   else {
572     Set-HSCADUserAddressBookVisibility -SAMAccountName $NewADUser.SAMAccountName
573   }
574
575
576   # Step 15: Now send CSC email
577   try {
578     Write-Output "`nStep 13: Send CSC Confirmation Email"
579
580     $SendNewAccountEmailParams = @{
581       CSCEmail = $CSCEmail
582       SamAccountName = $NewADUser.SamAccountName
```

```
583       PrimarySMTPAddress = $PrimarySMTPAddress
584       ErrorAction = "Stop"
585     }
586
587     if ($Testing) {
588       Send-HSCNewAccountEmail @SendNewAccountEmailParams
589     }
590     else {
591       $SendNewAccountEmailParams["CSCEmail"] = $CSCEmail
592       Send-HSCNewAccountEmail @SendNewAccountEmailParams
593     }
594   }
595   catch {
596     Write-Warning "Unable to send new account email"
597   }
598
599   Write-Output "************************"
600 }
601
602 if ($UserProcessed) {
603   Write-Output "Copying session transcript to logs to save directory"
604
605   try {
606     $TranscriptLogFile = Get-HSCLastFile -DirectoryPath "$PSScriptRoot\Logs\"
607     Copy-Item -Path $TranscriptLogFile -Destination $LogsToSavePath -ErrorAction
    Stop
608   }
609   catch {
610     Write-Warning "Error copying files to backup directory"
611   }
612 }
613
614 Invoke-HSCExitCommand -ErrorCount $Error.Count
```