

```
1 <#
2 .SYNOPSIS
3     This file is used to create HS domain and Office 365 accounts for users without
4     needing a CSC request form filled out.
5
6 .DESCRIPTION
7     This file is used to create HS domain and Office 365 accounts for users without
8     needing a CSC request form filled out. It does this by comparing the department
9     as given by SailPoint to mapping file for accounts in the NewUsers OU.
10
11     Once a matching user is found, the following steps are performed:
12     1. Search NewUsers OU for any accounts with departments in the
13         DepartmentMap.csv file.
14     1a. Verify if account has been claimed.
15     2. Determine correct OU to move user to & CSC Email Address
16     3. Move user to correct OU (which is specified in mapping file)
17     4. Refresh user object after AD move
18     5. Create & assign rights to home directory.
19     6. Add primary SMTP proxy address: ext15@hsc.wvu.edu (or @wvumedicine.org for
HVI)
20     7. Add remaining proxy addresses:
21         a. samaccountname@hsc.wvu.edu (if not already added in step 6)
22         b. samaccountname@WVUHSC.onmicrosoft.com
23         c. samaccountname@wvumedicine.org (Only for HVI users)
24     8. Add SIP address
25     9. Add user to groups
26     10. Add Yes/No365 to ext7
27     11. Set Name & Display Name
28     12. Enable AD Account
29     13. Add mailNickname field
30     14. Set msexchHiddenFromAddressListsEnabled to $false
31     15. Email CSC
32     16. Copy log file to save directory if user provisioned
33
34 .PARAMETER LogsToSavePath
35     The file runs every 15 minutes. Since most of the time no new accounts are
36     created, this specifies where to copy the session transcript when a new account
37     has been processed.
38
39 .PARAMETER DepartmentMapPath
40     This is the path to the DepartmentMap.csv file. This file maps the SailPoint
41     department names with the destination OUs, groups, and values for ext7.
42
43 .PARAMETER NewUsersOrgUnit
44     The org unit that is to be searched for new users.
45
46 .PARAMETER ADProperties
47     This is the array of additional properties that will be pulled for AD user
objects.
48
49 .PARAMETER Testing
50     This parameter is a flag to run -WhatIf commands instead of making any
51     real changes to an account sitting in NewUsers.
52
53 .NOTES
54     Written by: Jeff Brusoe
55     Last Updated: August 11, 2021
56 #>
57
58 [CmdletBinding()]
```

```
59 param (
60
61     [ValidateNotNullOrEmpty()]
62     [string]$LogsToSavePath = "$PSScriptRoot\AccountCreatedLogs\",
63
64     [ValidateNotNullOrEmpty()]
65     [string]$DepartmentMapPath = (Get-HSCGitHubRepoPath) +
66         "2CommonFiles\MappingFiles\DepartmentMap.csv",
67
68     [ValidateNotNullOrEmpty()]
69     [string]$NewUsersOrgUnit = "OU=NewUsers,DC=HS,DC=wwu-ad,DC=wwu,DC=edu",
70
71     [ValidateNotNullOrEmpty()]
72     [string[]]$ADProperties = @(
73         "extensionAttribute11",
74         "extensionAttribute15",
75         "whenCreated",
76         "PasswordLastSet",
77         "Department"
78     ),
79
80     [switch]$Testing
81 )
82
83 try {
84     Write-Verbose "Initializing Environment"
85
86     [string]$TranscriptLogFile = Set-HSCEnvironment
87
88     if ($null -eq $TranscriptLogFile) {
89         Write-Warning "There was an error configuring the environment. Program is
90 exiting."
91         Invoke-HSCExitCommand -ErrorCount $Error.Count
92     }
93
94     Write-Verbose "Transcript Log file: $TranscriptLogFile"
95
96     $NewADUserCount = 0
97     $UserProcessed = $false
98 }
99 catch {
100     Write-Warning "Error configuring environment"
101     Invoke-HSCExitCommand -ErrorCount $Error.Count
102 }
103
104 try {
105     Write-Output "Opening Department Map File"
106     Write-Output "Department Mapping File: $DepartmentMapPath"
107
108     $DepartmentMap = Import-Csv $DepartmentMapPath -ErrorAction Stop
109 }
110 catch {
111     Write-Warning "Unable to open department map. Program is exiting"
112     Invoke-HSCExitCommand -ErrorCount $Error.Count
113 }
114
115 try {
116     Write-Output "`n`nStep 1: Search NewUsers for matching department names"
117     Write-Output "`n`nGetting SailPoint departments from department map file"
```

```

118 $SPDepartments = $DepartmentMap.SailPointDepartmentName
119 Write-Output "SailPoint Departments:"
120 Write-Output $SPDepartments
121
122 Write-Output "`n`nSearching AD NewUsers for matching department names"
123
124 $GetADUserParams = @{
125     SearchBase = $NewUsersOrgUnit
126     Properties = $ADProperties
127     Filter = "*"
128     SearchScope = "OneLevel"
129     ErrorAction = "Stop"
130 }
131
132 $ADUsers = Get-ADUser @GetADUserParams |
133     Where-Object { $SPDepartments -contains $_.Department }
134
135 if ($null -eq $ADUsers) {
136     Write-Warning "No matching departments were found"
137     Invoke-HSCExitCommand -ErrorCount $Error.Count
138 }
139 }
140 catch {
141     Write-Warning "Unable to generate AD user list. Program is exiting."
142     Invoke-HSCExitCommand -ErrorCount $Error.Count
143 }
144
145 Write-Output "`n*****`n"
146
147 foreach ($ADUser in $ADUsers)
148 {
149     $UserProcessed = $true
150     Write-Output "User found in NewUsers Org Unit"
151     Write-Output $("SamAccountName: " + $ADUser.SamAccountName)
152     Write-Output $("Password Last Set: " + $ADUser.PasswordLastSet)
153     Write-Output $("When Created: " + $ADUser.whenCreated)
154     Write-Output "Distinguished Name:"
155     Write-Output $ADUser.DistinguishedName
156
157     $NewADUserCount++
158     Write-Output "New AD User Count: $NewADUserCount"
159
160     #Step 1a: Verify user is claimed
161     Write-Output "User was found in NewUsers org unit"
162     Write-Output $("Password Last Set: " + $ADUser.PasswordLastSet)
163     Write-Output $("When Created: " + $ADUser.whenCreated)
164
165     #Test time difference between password last set and when created
166     [datetime]$PasswordLastSet = $ADUser.PasswordLastSet
167     [datetime]$WhenCreated = $ADUser.whenCreated
168
169     if ($PasswordLastSet.AddMinutes(-2) -le $WhenCreated) {
170         #Unclaimed account
171         Write-Output "Account is unclaimed. CSC request will not be processed"
172         Write-Output "*****"
173
174         continue
175     }
176     else {
177         #Claimed Account

```

```
178     Write-Output "Account has been claimed. CSC request will be processed"
179 }
180
181 #Step 2: Find OU to move user to
182 Write-Output "`nStep2: Determine destination OU & CSC Email Address"
183
184 $DepartmentOrgUnit = ($DepartmentMap |
185     Where-Object {$_.SailPointDepartmentName -eq $ADUser.Department}).OUPath.Trim()
186
187 $CSCEmail = ($DepartmentMap |
188     Where-Object {$_.SailPointDepartmentName -eq $ADUser.Department}).CSCEmail
189
190 Write-Output "Destination OU From File: $DepartmentOrgUnit"
191
192 try {
193     Write-Output "Searching for Org Unit"
194     $TargetOU = Get-ADOrganizationalUnit -Identity $DepartmentOrgUnit -ErrorAction
Stop
195
196     Write-Output "AD Target OU:"
197     Write-Output $TargetOU.DistinguishedName
198 }
199 catch {
200     Write-Warning "Unable to find target OU in AD"
201     Write-Output "`n*****`n"
202
203     continue
204 }
205
206 if ($null -eq $TargetOU) {
207     Write-Warning "Unable to find destination OU in AD."
208     Write-Output "`n*****`n"
209
210     continue
211 }
212 elseif (($TargetOU | Measure-Object).Count -gt 1) {
213     Write-Warning "Multiple Org Units were found"
214     Write-Output "`n*****`n"
215
216     continue
217 }
218 elseif (($TargetOU | Measure-Object).Count -eq 0) {
219     Write-Warning "Unable to find the user's destination org unit"
220     Write-Output "`n*****`n"
221
222     continue
223 }
224 else {
225     Write-Output "One unique destination OU was found. User is being moved there."
226 }
227
228 #Step 3: Attempt user move
229 try {
230     Write-Output "`n`nStep 3: Attempting to move user to destination OU"
231
232     $MoveADObjectParams = @{
233         Identity = $ADUser.DistinguishedName
234         TargetPath = $TargetOU
235         ErrorAction = "Stop"
236         WhatIf = $true
```

```
237     }
238
239     if ($Testing) {
240         Move-ADObject @MoveADObjectParams
241     }
242     else {
243         $MoveADObjectParams["WhatIf"] = $false
244
245         Move-ADObject @MoveADObjectParams
246
247         Write-Output "User move complete"
248     }
249     Start-HSCCountdown -Message "Delay to allow time for user move to sync" -
Seconds 30
250
251     Write-Output "Successfully moved user"
252 }
253 catch {
254     Write-Warning "Unable to move user to target OU"
255     Write-Output "`n*****`n"
256
257     continue
258 }
259
260 #Step 4: Refresh AD user object after move
261 try {
262     Write-Output "`nStep 4: Refreshing user object after move"
263
264     $LDAPFilter = "(sAMAccountName=" + $ADUser.SamAccountName + ")"
265     Write-Verbose "LDAPFilter: $LDAPFilter"
266
267     $GetADUserParams = @{
268         LDAPFilter = $LDAPFilter
269         Properties = $ADProperties
270         ErrorAction = "Stop"
271     }
272
273     $NewADUser = Get-ADUser @GetADUserParams
274 }
275 catch {
276     Write-Warning "Unable to find user object after move."
277     Write-Output "`n*****`n"
278
279     continue
280 }
281
282 #Step 5: Create & add permissions for home directory
283 #To Do: Add code here for other departments
284 Write-Output "`nStep5: Creating home directory"
285
286 $HomeDirectoryPath = ($DepartmentMap |
287     Where-Object {$_.SailPointDepartmentName -eq
$ADUser.Department}).HomeDirectoryPath
288
289 Write-Output "Home Directory Path: $HomeDirectoryPath"
290
291 if ($HomeDirectoryPath -eq "NoHomeDirectory") {
292     Write-Output $("Home Directory Path: " + $HomeDirectoryPath)
293     Write-Output "Not creating a home directory"
294 }
```

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

```

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

```

```

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

```



```
465
466     if ($Testing) {
467         $NewADUser |
468         Set-ADUser -Add @{ProxyAddresses = $SIPAddress} -ErrorAction Stop -WhatIf
469     }
470     else {
471         $NewADUser |
472         Set-ADUser -Add @{ProxyAddresses = $SIPAddress} -ErrorAction Stop
473     }
474 }
475 catch {
476     Write-Warning "Unable to add SIP address"
477 }
478
479 Start-HSCCountdown -Message "Delay after adding SIP address" -Seconds 5
480
481 #Step 9: Add newly created user to groups
482 try {
483     Write-Output "`nStep 9: Adding user to groups"
484
485     $UserDN = $NewADUser.DistinguishedName
486
487     Write-Output "User DN: "
488     Write-Output $UserDN
489
490     if ($Testing) {
491         Set-HSCGroupMembership -UserDN $UserDN -WhatIf -ErrorAction Stop
492     }
493     else {
494         Set-HSCGroupMembership -UserDN $UserDN -ErrorAction Stop
495     }
496 }
497 catch {
498     Write-Warning "Unable to add user to groups"
499 }
500
501 #Step 10: Add Yes/No365 to ext7
502 try {
503     Write-Output "`nStep10: Setting extensionAttribute7"
504
505     #Determine value of ext7
506     $Ext7 = ($DepartmentMap |
507         Where-Object {$_.SailPointDepartmentName -eq
508 $ADUser.Department}).CreateEmail
509
510     if ($Ext7 -eq "Yes") {
511         if ($Testing) {
512             Set-HSCADUserExt7 $NewADUser.SamAccountName -WhatIf
513         }
514         else {
515             Set-HSCADUserExt7 $NewADUser.SamAccountName
516         }
517     }
518     else {
519         if ($Testing) {
520             Set-HSCADUserExt7 $NewADUser.SamAccountName -WhatIf -No365
521         }
522         else {
523             Set-HSCADUserExt7 $NewADUser.SamAccountName -No365
524         }
525     }
526 }
```

```
524     }
525 }
526 catch {
527     Write-Warning "Unable to set ext7"
528 }
529
530 Start-HSCCountdown -Message "Synchronization Delay" -seconds 2
531
532 #Step 11: Set name and display name
533 try {
534     Write-Output "`nStep 11: Setting name and display name"
535
536     $DisplayName = $NewADUser.Surname + ", " + $NewADUser.GivenName
537     Write-Output "Display Name: $DisplayName`n"
538
539     if ($Testing) {
540         $NewADUser |
541         Set-ADUser -DisplayName $DisplayName -ErrorAction Stop -WhatIf
542     }
543     else {
544         $NewADUser |
545         Set-ADUser -DisplayName $DisplayName -ErrorAction Stop
546     }
547 }
548 catch {
549     Write-Warning "Error setting display name"
550 }
551
552 #Step 12: Enable AD Account
553 try {
554     Write-Output $("`nStep 12: Enabling AD Account: " + $NewADUser.SamAccountName)
555
556     if ($Testing) {
557         $NewADUser |
558         Enable-ADAccount -ErrorAction Stop -WhatIf
559     }
560     else {
561         $NewADUser |
562         Enable-ADAccount -ErrorAction Stop
563     }
564
565     Write-Output "Successfully enabled account"
566 }
567 catch {
568     Write-Warning "Unable to enable account"
569 }
570
571 #Step 13 & 14: Add mailNickname and Unhide account from address list
572 Write-Output "`n`nStep 13: Att mailNickname and"
573 Write-Output "Step 14: Unhide user from address book"
574
575 try {
576     $AddressBookVisibilityParams = @{
577         SamAccountName = $NewADUser.SamAccountName
578         ErrorAction = "Stop"
579         WhatIf = $true
580     }
581
582     if ($Testing) {
583         Set-HSCADUserAddressBookVisibility @AddressBookVisibilityParams
```

```
584     }
585     else {
586         $AddressBookVisibilityParams["WhatIf"] = $false
587
588         Set-HSCADUserAddressBookVisibility @AddressBookVisibilityParams
589     }
590 }
591 catch {
592     Write-Warning "Unable to add mail nickname and unhide user account"
593 }
594
595 # Step 15: Now send CSC email
596 try {
597     Write-Output "`nStep 15: Send CSC Confirmation Email"
598
599     $SendNewAccountEmailParams = @{
600         CSCEmail = $CSCEmail
601         SamAccountName = $NewADUser.SamAccountName
602         PrimarySMTPAddress = $PrimarySMTPAddress
603         ErrorAction = "Stop"
604     }
605
606     if ($Testing) {
607         Send-HSCNewAccountEmail @SendNewAccountEmailParams
608     }
609     else {
610         $SendNewAccountEmailParams["CSCEmail"] = $CSCEmail
611         Send-HSCNewAccountEmail @SendNewAccountEmailParams
612     }
613 }
614 catch {
615     Write-Warning "Unable to send new account email"
616 }
617
618 Write-Output "*****"
619 }
620
621 #Step 16: Copy log file if a user was provisioned
622 if ($UserProcessed) {
623     Write-Output "`nStep 16: Copying session transcript to logs to save directory"
624
625     try {
626         $TranscriptLogFile = Get-HSCLastFile -DirectoryPath "$PSScriptRoot\Logs\"
627         Copy-Item -Path $TranscriptLogFile -Destination $LogsToSavePath -ErrorAction
628 Stop
629         Write-Output "Successfully copied log file"
630     }
631     catch {
632         Write-Warning "Error copying files to backup directory"
633     }
634 }
635
636 Invoke-HSCExitCommand -ErrorCount $Error.Count
```