

```
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     Currently, only four departments are in this mapping file: Family Medicine,
12     Opthamology,
13     Eastern Division, and Anesthesiology.
14
15     Once a matching user is found, the following steps are performed:
16     1. Search NewUsersOU for any accounts with departments in the
17         DepartmentMap.csv file.
18     2. Determine correct OU to move user to
19     3. Move user to correct OU (which is specified in mapping file)
20     4. Refresh user object after AD move
21     5. Create & assign rights to home directory.
22     6. Add primary SMTP proxy address: ext15@hsc.wvu.edu (or @wvumedicine.org for
23         HVI)
24     7. Add remaining proxy addresses:
25         a. samaccountname@hsc.wvu.edu (if not already added in step 6)
26         b. samaccountname@WVUHSC.onmicrosoft.com
27         c. samaccountname@wvumedicine.org (Only for HVI users)
28     8. Add SIP address
29     9. Add user to groups
30     10. Add Yes/No365 to ext7
31     11. Set Name & Display Name
32     12. Enable AD Account
33     13. Email CSC
34
35 .PARAMETER Testing
36     This parameter is a flag to run -WhatIf commands instead of making any
37     real changes to an account sitting in NewUsers.
38
39 .PARAMETER LogsToSavePath
40     The file runs every 15 minutes. Since most of the time no new accounts are
41     created, this specifies where to copy the session transcript when a new account
42     has been processed.
43
44 .PARAMETER DepartmentMapPath
45     This is the path to the DepartmentMap.csv file. This file maps the SailPoint
46     department names with the destination OUs, groups, and values for ext7.
47
48 .PARAMETER NewUsersOrgUnit
49     The org unit that is to be searched for new users.
50
51 .PARAMETER ADProperties
52     This is the array of additional properties that will be pulled for AD user
53     objects.
54
55 .NOTES
56     Written by: Jeff Brusoe
57     Last Updated: January 11, 2021
58
59     To Be Done:
60     1. Add code to create home directories for other departments.
```

```

58     2. Add code for date offset
59 #>
60
61 [CmdletBinding()]
62 param (
63     [switch]$Testing,
64
65     [ValidateNotNullOrEmpty()]
66     [string]$LogsToSavePath = "$PSScriptRoot\AccountCreatedLogs\",
67
68     [ValidateNotNullOrEmpty()]
69     [string]$DepartmentMapPath = "$PSScriptRoot\MappingFiles\DepartmentMap.csv",
70
71     [ValidateNotNullOrEmpty()]
72     [string]$NewUsersOrgUnit = "OU=NewUsers,DC=HS,DC=wwu-ad,DC=wwu,DC=edu",
73
74     [ValidateNotNullOrEmpty()]
75     [string[]]$ADProperties = @(
76         "extensionAttribute11",
77         "extensionAttribute15",
78         "whenCreated",
79         "PasswordLastSet",
80         "Department"
81     ),
82
83     [ValidateNotNullOrEmpty()]
84     [datetime]$StartDate = (Get-Date "1/21/2021 16:00")
85 )
86
87 #Initialize environment block
88 try {
89     [string]$TranscriptLogFile = Set-HSCEnvironment
90
91     if ($null -eq $TranscriptLogFile)
92     {
93         Write-Warning "There was an error configuring the environment. Program is
94         exiting."
95         Invoke-HSCExitCommand -ErrorCount $Error.Count
96     }
97
98     Write-Verbose "Transcript Log file: $TranscriptLogFile"
99     Write-Output "Start Date: $StartDate"
100
101     $UserProcessed = $false
102 }
103 catch {
104     Write-Warning "Error configuing environment"
105     Invoke-HSCExitCommand -ErrorCount $Error.Count
106 }
107
108 #Read & process department mapping file
109 try {
110     Write-Output "Opening Department Map File"
111     $DepartmentMap = Import-Csv $DepartmentMapPath -ErrorAction Stop
112 }
113 catch {
114     Write-Warning "Unable to open department map. Program is exiting"
115     Invoke-HSCExitCommand -ErrorCount $Error.Count
116 }

```

```

117 #Step 1: Search NewUsers for matching department names
118 try {
119     Write-Output "`nStep1: Getting SailPoint deparmtents from department map file"
120
121     $SPDepartments = $DepartmentMap.SailPointDepartmentName
122     Write-Output "SailPoint Departments:"
123     Write-Output $SPDepartments
124
125     Write-Output "Searching AD NewUsers for matching department names"
126
127     $GetADUserParams = @{
128         SearchBase = $NewUsersOrgUnit
129         Properties = $ADProperties
130         Filter = "*"
131         ErrorAction = "Stop"
132     }
133     $ADUsers = Get-ADUser @GetADUserParams |
134         Where-Object {($SPDepartments -contains $_.Department) -AND
135             ([datetime]$_whenCreated -gt $StartDate)}
136 }
137 catch {
138     Write-Warning "Unable to generate AD user list. Program is exiting."
139     Invoke-HSCEExitCommand -ErrorCount $Error.Count
140 }
141
142 if ($null -eq $ADUsers)
143 {
144     Write-Warning "No matching departments were found"
145     Invoke-HSCEExitCommand -ErrorCount $Error.Count
146 }
147
148 Write-Output "`n*****`n"
149
150 foreach ($ADUser in $ADUsers)
151 {
152     $UserProcessed = $true
153     Write-Output "User found in NewUsers Org Unit"
154     Write-Output $("SamAccountName: " + $ADUser.SamAccountName)
155     Write-Output $("Password Last Set: " + $ADUser.PasswordLastSet)
156     Write-Output $("When Created: " + $ADUser.whenCreated)
157     Write-Output "Distinguished Name:"
158     Write-Output $ADUser.DistinguishedName
159
160     #Step 2: Find OU to move user to
161     Write-Output "`nStep2: Determine destination OU"
162
163     $DepartmentOrgUnit = ($DepartmentMap |
164         Where-Object {$_SailPointDepartmentName -eq $ADUser.Department}).OUPath
165
166     Write-Output "Destination OU From File: $DepartmentOrgUnit"
167
168     try {
169         Write-Output "Searching for Org Unit"
170         $TargetOU = Get-ADOrganizationalUnit -Filter * -ErrorAction Stop |
171             Where-Object {$_DistinguishedName -like "*$DepartmentOrgUnit*"}
172
173         Write-Output "AD Target OU:"
174         Write-Output $TargetOU.DistinguishedName
175     }
176     catch {

```

```

177     Write-Warning "Unable to find target OU in AD"
178 }
179
180 if ($null -eq $TargetOU)
181 {
182     Write-Warning "Unable to find destination OU in AD."
183     Write-Output "`n*****`n"
184     continue
185 }
186 elseif (($TargetOU | Measure-Object).Count -gt 1)
187 {
188     Write-Warning "Multiple Org Units were found"
189     Write-Output "`n*****`n"
190     continue
191 }
192 elseif (($TargetOU | Measure-Object).Count -eq 0)
193 {
194     Write-Warning "Unable to find the user's destination org unit"
195     Write-Output "`n*****`n"
196     continue
197 }
198 else {
199     Write-Output "One unique destination OU was found. User is being moved there."
200 }
201
202 #Step 3: Attempt user move
203 try {
204     Write-Output "Step 3: Attempting to move user to destination OU"
205
206     $MoveADObjectParams = @{
207         Identity = $ADUser.DistinguishedName
208         TargetPath = $TargetOU
209         ErrorAction = "Stop"
210         WhatIf = $true
211     }
212
213     if ($Testing) {
214         Move-ADObject @MoveADObjectParams
215     }
216     else {
217         $MoveADObjectParams["WhatIf"] = $false
218
219         Move-ADObject @MoveADObjectParams
220     }
221
222     Write-Output "Successfully moved user"
223 }
224 catch {
225     Write-Warning "Unable to move user to target OU"
226     Write-Output "`n*****`n"
227     continue
228 }
229
230 Start-HSCCountdown -Message "Delay to allow time for user move to sync" -Seconds
10
231
232 #Step 4: Refresh AD user object after move
233 try {
234     Write-Output "`nStep 4: Generating new user object after move"
235

```

```

236     $NewADUser = Get-ADUser -Filter * -Properties $Properties -ErrorAction Stop |
237         Where-Object {$_.UserPrincipalName -eq $NewADUser.UserPrincipalName}
238     }
239     catch {
240         Write-Output "Unable to find user object after move."
241         continue
242     }
243
244     #Step 5: Create & add permissions for home directory
245     #To Do: Add code here for other departments
246     Write-Output "`nStep5: Creating home directory"
247
248     $HomeDirectoryPath = ($DepartmentMap |
249         Where-Object {$_.SailPointDepartmentName -eq
$NewADUser.Department}).HomeDirectoryPath
250
251     Write-Output "Home Directory Path: $HomeDirectoryPath"
252
253     if ($HomeDirectoryPath -eq "NoHomeDirectory")
254     {
255         Write-Output $("Home Directory Path: " + $HomeDirectoryInfo.DirectoryPath)
256         Write-Output "Not creating a home directory"
257     }
258     else {
259         #Create home directory
260         $UserHomeDirectory = $HomeDirectoryPath + "\" + $NewADUser.SamAccountName
261         Write-Output "User Home Directory: $UserHomeDirectory"
262
263         if ($Testing) {
264             Write-Output "Home directory not being created because of -Testing parameter"
265             New-Item -ItemType Directory -Path $UserHomeDirectory -WhatIf
266         }
267         else {
268             try {
269                 New-Item -ItemType Directory -Path $UserHomeDirectory -ErrorAction Stop
270             }
271             catch {
272                 Write-Warning "Error creating home directory: $UserHomeDirectory"
273             }
274         }
275
276         Start-HSCCountdown -Message "Home Directory Created. Delay before adding
permissions." -Seconds 10
277
278         #Now add file system permissions
279         try {
280             $UserName = "HS\" + $NewADUser.SamAccountName
281             $Acl = (Get-Item $UserHomeDirectory -ErrorAction
Stop).GetAccessControl('Access')
282
283             $Ar = New-Object
System.Security.AccessControl.FileSystemAccessRule($Username,
284                 'FullControl',
285                 'ContainerInherit,ObjectInherit',
286                 'None',
287                 'Allow'
288             )
289
290             $Acl.SetAccessRule($Ar)
291             Set-Acl -Path $UserHomeDirectory -AclObject $Acl -ErrorAction Stop

```

```

292     }
293     catch {
294         Write-Warning "Unable to set ACL on home directory: $UserHomeDirectory"
295     }
296 }
297
298 #Add Proxy addresses
299 Write-Output "`nAdding ProxyAddresses"
300
301 #Step 6: Set Primary SMTP Address
302 Write-Output "`nStep 6: Adding Primary SMTP Address"
303
304 if ($NewADUser.DistinguishedName.IndexOf("HVI") -lt 0) {
305     $PrimarySMTPAddress = $NewADUser.extensionAttribute15 + "@hsc.wvu.edu"
306 }
307 else {
308     $PrimarySMTPAddress = $NewADUser.extensionAttribute15 + "@wvumedicine.org"
309 }
310
311 try {
312     Write-Output "Setting Primary SMTP Address: $PrimarySMTPAddress"
313
314     $PrimarySMTPAddress = "SMTP:" + $PrimarySMTPAddress
315
316     if ($Testing) {
317         $NewADUser |
318         Set-ADUser -Add @{ProxyAddresses=$PrimarySMTPAddress} -ErrorAction Stop -
WhatIf
319     }
320     else {
321         $NewADUser |
322         Set-ADUser -Add @{ProxyAddresses=$PrimarySMTPAddress} -ErrorAction Stop
323     }
324
325     Write-Output "Successfully set primary smtp address"
326
327     $PrimarySMTPAddress = $PrimarySMTPAddress -replace "SMTP:", ""
328 }
329 catch {
330     #An error here will not allow other proxy addresses to be added.
331     Write-Warning "Error setting primary smtp address"
332 }
333
334 Start-HSCCountdown -Message "Delay after adding primary SMTP address" -Seconds 5
335
336 #Step 7a: Verify samaccountname@hsc.wvu.edu is a proxy address
337 $SAMPProxyAddress = $NewADUser.SamAccountName + "@hsc.wvu.edu"
338
339 if ($SAMPProxyAddress -ne $PrimarySMTPAddress)
340 {
341     Write-Output "`nStep 7a: Adding Proxy Address: $SAMPProxyAddress"
342     $SAMPProxyAddress = "smtp:" + $SAMPProxyAddress
343
344     try {
345         if ($Testing) {
346             $NewADUser |
347             Set-ADUser -Add @{ProxyAddresses = $SAMPProxyAddress} -ErrorAction Stop -
WhatIf
348         }
349         else {

```

```

350     $NewADUser |
351     Set-ADUser -Add @{ProxyAddresses = $SAMProxyAddress} -ErrorAction Stop
352 }
353
354     Write-Output "Succesfully set samaccountname@hsc proxy address"
355 }
356 catch {
357     Write-Warning "Error setting samaccountname@hsc proxy address"
358 }
359
360     $SAMProxyAddress = $NewADUser.SamAccountName + "@hsc.wvu.edu"
361 }
362 else {
363     Write-Output "samaccountname@hsc.wvu.edu is already a proxy address"
364 }
365
366 #Step 7b: Add samaccountname@wvuhsc.onmicrosoft.com
367 try {
368     $OnMicrosoftProxy = $NewADUser.samaccountname + "@WVUHSC.onmicrosoft.com"
369     Write-Output "`nStep 7b: Adding onmicrosoft.com proxy: $OnMicrosoftProxy"
370
371     $OnMicrosoftProxy = "smtp:" + $OnMicrosoftProxy
372
373     if ($Testing) {
374         $NewADUser |
375         Set-ADUser -Add @{ProxyAddresses=$OnMicrosoftProxy} -ErrorAction Stop -
WhatIf
376     }
377     else {
378         $NewADUser |
379         Set-ADUser -Add @{ProxyAddresses=$OnMicrosoftProxy} -ErrorAction Stop
380     }
381
382     Write-Output "Successfully set onmicrosoft.com proxy address"
383
384     $OnMicrosoftProxy = $OnMicrosoftProxy -replace "smtp:", ""
385 }
386 catch {
387     Write-Warning "Error setting onmicrosoft.com proxy address"
388 }
389
390 #Step 7c: Add samaccountname@hsc.wvu.edu for HVI users
391 Write-Output "`nStep 7c: Settin samaccountname@hsc.wvu.edu for HVI users"
392
393 if ($PrimarySMTPAddress.IndexOf("wvumedicine.org") -gt 0)
394 {
395     $HVISAMProxyAddress = $SAMProxyAddress -replace "hsc.wvu.edu","wvumedicine.org"
396
397     if ($HVISAMProxyAddress -ne $PrimarySMTPAddress) {
398         Write-Output "Adding Proxy Address for HVI User: $HVISAMProxyAddress"
399         $HVISAMProxyAddress = "smtp:" + $HVISAMProxyAddress
400
401         try {
402             if ($Testing) {
403                 $NewADUser |
404                 Set-ADUser -Add @{ProxyAddresses = $HVISAMProxyAddress} -ErrorAction
Stop -WhatIf
405             }
406             else {
407                 $NewADUser |

```

```
408         Set-ADUser -Add @{ProxyAddresses = $HVISAMProxyAddress} -ErrorAction
Stop
409     }
410
411     Write-Output "Successfully set HVI samaccountname@hsc.wvu.edu email
address"
412 }
413 catch {
414     Write-Warning "Unable to set HVI samaccountname@hsc.wvu.edu email address"
415 }
416 }
417 }
418
419 Start-HSCCountdown -Message "Delay to allow proxyaddresses to sync before adding
SIP address" -Seconds 5
420
421 #Step 8: Add SIP address
422 try {
423     $SIPAddress = "SIP:" + $NewADUser.SamAccountName + "@hsc.wvu.edu"
424     Write-Output "`nStep 8: Adding SIP address: $SIPAddress"
425
426     if ($Testing) {
427         $NewADUser |
428         Set-ADUser -Add @{ProxyAddresses = $SIPAddress} -ErrorAction Stop -WhatIf
429     }
430     else {
431         $NewADUser |
432         Set-ADUser -Add @{ProxyAddresses = $SIPAddress} -ErrorAction Stop
433     }
434 }
435 catch {
436     Write-Warning "Unable to add SIP address"
437 }
438
439 Start-HSCCountdown -Message "Delay after adding SIP address" -Seconds 5
440
441 #Step 9: Add newly created user to groups
442 try {
443     Write-Output "`nStep 9: Adding user to groups"
444
445     $UserDN = $NewADUser.DistinguishedName
446
447     Write-Output "User DN: "
448     Write-Output $UserDN
449
450     if ($Testing) {
451         Set-HSCGroupMembership -UserDN $UserDN -WhatIf
452     }
453     else {
454         Set-HSCGroupMembership -UserDN $UserDN
455     }
456 }
457 catch {
458     Write-Warning "Unable to add user to groups"
459 }
460
461 #Step 10: Add Yes/No365 to ext7
462 try {
463     Write-Output "`nStep10: Setting extensionAttribute7"
464 }
```



```
465 #Determine value of ext7
466 $Ext7 = ($DepartmentMap |
467     Where-Object {$_.SailPointDepartmentName -eq
$ADUser.Department}).CreateEmail
468
469 if ($Ext7 -eq "Yes") {
470     $Ext7 = "Yes365"
471 }
472 else {
473     $Ext7 = "No365"
474 }
475
476 Write-Output "Setting ext7 to $Ext7"
477
478 if ($Testing) {
479     $NewADUser |
480     Set-ADUser -Add @{extensionAttribute7=$Ext7} -ErrorAction Stop -WhatIf
481 }
482 else {
483     $NewADUser |
484     Set-ADUser -Add @{extensionAttribute7=$Ext7} -ErrorAction Stop
485 }
486
487 }
488 catch {
489     Write-Warning "Unable to set ext7"
490 }
491
492 Start-HSCCountdown -Message "Synchronization Delay" -seconds 2
493
494 #Step 11: Set name and display name
495 try {
496     Write-Output "`nStep 11: Setting name and display name"
497
498     $DisplayName = $NewADUser.Surname + ", " + $NewADUser.GivenName
499     Write-Output "Display Name: $DisplayName`n"
500
501     if ($Testing) {
502         $NewADUser |
503         Set-ADUser -DisplayName $DisplayName -ErrorAction Stop -WhatIf
504     }
505     else {
506         $NewADUser |
507         Set-ADUser -DisplayName $DisplayName -ErrorAction Stop
508     }
509 }
510 catch {
511     Write-Warning "Error setting display name"
512 }
513
514 #Step 12: Enable AD Account
515 try {
516     Write-Output $("`nStep 12: Enabling AD Account" + $NewADUser.SamAccountName)
517
518     if ($Testing) {
519         $NewADUser |
520         Enable-ADAccount -ErrorAction Stop -WhatIf
521     }
522     else {
523         $NewADUser |
```

```
524     Enable-ADAccount -ErrorAction Stop
525 }
526
527 Write-Output "Successfully enabled account"
528 }
529 catch {
530     Write-Warning "Unable to enable account"
531 }
532
533 # Step 13: Now send CSC email
534 try {
535     Write-Output "`nStep 13: Send CSC Confirmation Email"
536
537     $SendNewAccountEmailParams = @{
538         CSCEmail = $CSCEmail
539         SamAccountName = $NewADUser.SamAccountName
540         PrimarySMTPAddress = $PrimarySMTPAddress
541         ErrorAction = "Stop"
542     }
543
544     if ($Testing) {
545         Send-HSCNewAccountEmail @SendNewAccountEmailParams
546     }
547     else {
548         $SendNewAccountEmailParams["CSCEmail"] = $CSCEmail
549         Send-HSCNewAccountEmail @SendNewAccountEmailParams
550     }
551 }
552 catch {
553     Write-Warning "Unable to send new account email"
554 }
555
556 Write-Output "*****"
557 }
558
559 if ($UserProcessed) {
560     Write-Output "Copying session transcript to logs to save directory"
561
562     try {
563         Copy-Item -Path $TranscriptLogFile -Destination $LogsToSavePath -ErrorAction
Stop
564     }
565     catch {
566         Write-Warning "Error copying files to backup directory"
567     }
568 }
569
570 Invoke-HSCExitCommand -ErrorCount $Error.Count
```