

```
1 <#
2 .SYNOPSIS
3   This module contains some common Active Directory functions that are used by many
   HSC PowerShell files.
4
5 .DESCRIPTION
6   Active Directory functions included in this module:
7
8   1. Get-HSCLoggedInUser
9   2. Get-HSCPrimarySMTPAddress
10  3. Set-HSCPasswordRequired
11
12 .NOTES
13   HSC-ActiveDirectoryModule.psm1
14   Last Modified by: Jeff Brusoe
15   Last Modified: July 16, 2020
16
17   Version: 1.0
18 #>
19
20 [CmdletBinding()]
21 [Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSAvoidTrailingWhiteSpace","",Justification = "Not relevant")]
22 param ()
23
24 Import-Module ActiveDirectory
25
26 Function Get-HSCDirectoryMapping
27 {
28   <#
29   .SYNOPSIS
30     This function takes a user's distinguished name for input and returns
31     the path to the user's home directory.
32
33   .NOTES
34     Last Modified by: Jeff Brusoe
35     Last Modified: July 17, 2020
36   #>
37
38   [CmdletBinding()]
39   param (
40     [Parameter(ValueFromPipeline = $true)]
41     [string]$UserDN = $null,
42     [switch]$DetermineFullPath
43   )
44
45   begin
46   {
47     #Create object to hold directory information
48     $HomeDirectoryInfo = new-object PSObject
49
50     $HomeDirectoryInfo | Add-Member -type NoteProperty -Name DirectoryPath -value
51     $null
52     $HomeDirectoryInfo | Add-Member -type NoteProperty -Name FullPath -Value $false
53     #In cases where the DN still needs to be parsed up later recursively, the
54     FullPath value is set to false.
55     #If the DirectoryPath value is the correct (& final) home directory path, the
56     FullPath value is set to true.
57
58     if ($UserDN.IndexOf("CN=") -ge 0)
```

```

56     {
57         #Need to remove this from the DN
58         $UserDN = $UserDN.substring($UserDN.indexOf(",")+1).Trim()
59         Write-Verbose "Cleaned UserDN: $UserDN"
60     }
61 }
62
63 process
64 {
65     #First check DirectoryMapping file for match
66     [string]$HomeDirectoryPath = $null
67
68     #Step 1: Check against home directory mapping file.
69     #$HomeDirectoryMappings = Import-Csv $($MyInvocation.PSScriptRoot +
70     "\HomeDirectoryMapping.csv")
71     $HomeDirectoryMappings = Import-Csv "C:\Users\microsoft\Documents\GitHub\HSC-
PowerShell-Repository\Create-NewAccount\HomeDirectoryMapping.csv"
72     $HomeDirectoryMappings = Import-Csv "C:\HSCGitHub\HSC-PowerShell-
Repository\Create-NewAccount\HomeDirectoryMapping.csv"
73     [string]$HomeDirectoryPath = ($HomeDirectoryMappings | where {$UserDN -eq
74     $_.UserDN}).DirectoryPath
75
76     if ([string]::IsNullOrEmpty($HomeDirectoryPath))
77     {
78         [string]$HomeDirectoryPath = ($HomeDirectoryMappings | where {$UserDN -match
79         $_.UserDN}).DirectoryPath
80     }
81
82     if ([string]::IsNullOrEmpty($HomeDirectoryPath))
83     {
84         Write-verbose "No match from directory mapping file"
85     }
86     else {
87         $HomeDirectoryInfo.DirectoryPath = $HomeDirectoryPath
88         $HomeDirectoryInfo.FullPath = $true
89     }
90
91     #Step 2: Check against predefined mappings
92     [string]$ParentPath = $null
93
94     if ([string]::IsNullOrEmpty($HomeDirectoryPath))
95     {
96         switch -wildcard ($UserDN)
97         {
98             "*OU=Char_Div*"
99             {
100                 $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\Char_Div\"
101                 $HomeDirectoryInfo.FullPath = $false
102                 $ParentPath = "Char_Div"
103                 break
104             }
105             "*OU=ITS*"
106             {
107                 $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\ITS\"
108                 $HomeDirectoryInfo.FullPath = $false
109                 $ParentPath = "ITS"
110                 break
111             }
112             "*OU=ADMIN*"

```

```

111     {
112         $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\Admin\"
113         $HomeDirectoryInfo.FullPath = $false
114         $ParentPath = "ADMIN"
115         break
116     }
117     "*OU=BASSCI*"
118     {
119         $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\bassci\"
120         $HomeDirectoryInfo.FullPath = $false
121         break
122     }
123     "*OU=MBRCC*"
124     {
125         $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\mbrcc\"
126         $HomeDirectoryInfo.FullPath = $false
127         break
128     }
129     "*OU=SOM*"
130     {
131         $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\som\"
132         $HomeDirectoryInfo.FullPath = $false
133         break
134     }
135     "*OU=SON*"
136     {
137         $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\son\"
138         $HomeDirectoryInfo.FullPath = $false
139         break
140     }
141     "*OU=SOP*"
142     {
143         $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\sop\"
144         $HomeDirectoryInfo.FullPath = $false
145         break
146     }
147     "*OU=SPH*"
148     {
149         $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\sph\"
150         $HomeDirectoryInfo.FullPath = $false
151         break
152     }
153     default
154     {
155         $HomeDirectoryInfo.DirectoryPath = "NoHomeDirectory"
156         $HomeDirectoryInfo.FullPath = $true
157     }
158 }
159 }
160
161 #Step 3: Determine full path if switch not used
162 if ($DetermineFullPath -AND !$HomeDirectoryInfo.FullPath -AND
($HomeDirectoryInfo -ne "NoHomeDirectory"))
163 {
164     $ParsedDN = $UserDN -split ","
165     $ParsedDNCount = $ParsedDN.Length
166     $AddToPath = $false
167
168     for ($i=$ParsedDNCount-1; $i -ge 0; $i--)
169     {

```

```

170     $ParsedDN[$i] = $ParsedDN[$i] -replace "OU=", ""
171
172     if ($ParsedDN[$i] -eq $ParentPath)
173     {
174         $AddToPath = $true
175     }
176     elseif ($AddToPath)
177     {
178         $HomeDirectoryInfo.DirectoryPath = $HomeDirectoryInfo.DirectoryPath + "\"
179     + $ParsedDN[$i]+ "\"
180     }
181     $HomeDirectoryInfo.FullPath = $true
182 }
183 }
184
185 end
186 {
187     return $HomeDirectoryInfo
188 }
189 }
190
191 function Get-HSCLoggedOnUser
192 {
193     <#
194     .SYNOPSIS
195         This function returns the currently logged on user.
196
197     .OUTPUTS
198         Returns a PSCustomObject that has two properties: Logged on username and
199         domain.
200
201     .NOTES
202         Tested with the following version of PowerShell:
203         1. 5.1.18362.752
204         2. 7.0.2
205
206         Written by: Jeff Brusoe
207         Last Updated by: Jeff Brusoe
208         Last Updated: June 24, 2020
209     #>
210     [CmdletBinding()]
211     [OutputType([PSCustomObject])]
212     param()
213
214     try
215     {
216         $LoggedOnUser = [PSCustomObject]@{
217             UserName = $((Get-ChildItem Env:\USERNAME).Value)
218             Domain = $((Get-ChildItem Env:\USERDOMAIN).Value)
219         }
220
221         return $LoggedOnUser
222     }
223     catch
224     {
225         Write-Warning "Error getting logged on user" | Out-Null
226
227         return $null

```

```

228 }
229 }
230
231 function Get-HSCPrimarySMTPAddress
232 {
233     <#
234     .SYNOPSIS
235         This function retrieves the primary SMTP address for AD users.
236
237     .INPUTS
238         This function can take a string(array) or ADUser object(array) and will
239         get the primary SMTP address for those users.
240
241     .PARAMETER UserNames
242         This parameter takes a string array of users names as input. It will attempt to
243         get
244         the primary SMTP address after finding the users.
245
246     .PARAMETER ADUsers
247         Similar to UserNames, but this parameter takes an array of ADUsers
248         (Microsoft.ActiveDirectory.Management.ADAccount)
249         and attempts to get their primary SMTP address.
250
251     .EXAMPLE
252         PS C:\Windows\system32> "jbrusoe","krussell" | Get-HSCPrimarySMTPAddress
253
254         SamAccountName PrimarySMTPAddress
255         -----
256         jbrusoe          jbrusoe@hsc.wvu.edu
257         krussell         krussell@hsc.wvu.edu
258
259     .EXAMPLE
260         PS C:\Windows\system32> $Jeff = Get-ADUser jbrusoe -Properties proxyAddresses
261         PS C:\Windows\system32> $Kevin = Get-ADUser krussell -Properties proxyAddresses
262         PS C:\Windows\system32> $Jeff,$Kevin | Get-HSCPrimarySMTPAddress
263
264         SamAccountName PrimarySMTPAddress
265         -----
266         jbrusoe          jbrusoe@hsc.wvu.edu
267         krussell         krussell@hsc.wvu.edu
268
269     .NOTES
270         Written by: Jeff Brusoe
271         Last Updated by: Jeff Brusoe
272         Last Updated: July 16, 2020
273     #>
274
275     [CmdletBinding()]
276     [OutputType([PSObject])]
277
278     param (
279         [Parameter(ValueFromPipeline=$true,
280             ParameterSetName="ADUserArray",
281             Mandatory=$true,
282             Position=0)]
283         [Microsoft.ActiveDirectory.Management.ADAccount[]]$ADUsers,
284
285         [Parameter(ValueFromPipeline=$true,
286             ParameterSetName="UserNameArray",

```

```

286     Mandatory=$true,
287     Position=0)]
288     [string[]]$UserNames,
289
290     [switch]$NoOutput
291 )
292
293 begin
294 {
295     [psobject[]]$PrimarySMTPAddresses = @()
296 }
297
298 process
299 {
300     Write-Debug $("In process block - Parameter Set Name: " +
$PSCmdlet.ParameterSetName)
301
302     #Get array of ADUsers if a string array is passed in
303     if ($PSCmdlet.ParameterSetName -eq "UserNameArray")
304     {
305         $ADUsers = $null
306         foreach ($UserName in $UserNames)
307         {
308             try
309             {
310                 $ADUsers += Get-ADUser $UserName -Properties proxyAddresses -ErrorAction
Stop
311                 Write-Verbose "Found User: $UserName"
312             }
313             catch
314             {
315                 Write-Warning "Unable to find user name"
316
317                 $ADUserObject = New-Object -TypeName PSObject
318                 $ADUserObject | Add-Member -MemberType NoteProperty -Name
"SamAccountName" -Value $UserName
319                 $ADUserObject | Add-Member -MemberType NoteProperty -Name
"PrimarySMTPAddress" -Value "UserNotFound"
320
321                 $PrimarySMTPAddresses += $ADUserObject
322             }
323         }
324     }
325
326     if ($null -ne $ADUsers)
327     {
328         foreach ($ADUser in $ADUsers)
329         {
330             $ADUserObject = New-Object -TypeName PSObject
331             $ADUserObject | Add-Member -MemberType NoteProperty -Name "SamAccountName"
-Value $ADUser.SamAccountName
332
333
334             [string[]]$ProxyAddresses = $ADUser.proxyAddresses
335
336             [string]$PrimarySMTPAddress = $ProxyAddresses -cmatch "SMTP:"
337
338             if ([string]::IsNullOrEmpty($PrimarySMTPAddress))
339             {
340                 Write-Verbose "Primary SMTP Address isn't defined"

```

```

341     $ADUserObject | Add-Member -MemberType NoteProperty -Name
"PrimarySMTPAddress" -Value $null
342 }
343 else {
344     Write-Verbose $("Current User" + $ADUser.SamAccountName)
345     Write-Verbose "Primary SMTP Address: $PrimarySMTPAddress"
346
347     $PrimarySMTPAddress = ($PrimarySMTPAddress -replace "SMTP:", "").Trim()
348
349     $ADUserObject | Add-Member -MemberType NoteProperty -Name
"PrimarySMTPAddress" -Value $PrimarySMTPAddress
350 }
351
352     $PrimarySMTPAddresses += $ADUserObject
353 }
354 }
355 else
356 {
357     Write-Warning "ADUser object is null"
358 }
359 }
360
361 end
362 {
363     return $PrimarySMTPAddresses
364 }
365 }
366
367 Function Set-HSCPasswordRequired
368 {
369     <#
370     .SYNOPSIS
371         This function sets the password required for a user
372
373     .INPUTS
374         This function can take a string(array) or ADUser object(array) and will
375         set the password required attribute for those users.
376
377     .PARAMETER UserNames
378         This parameter takes a string array of users names as input. It will attempt to
379         set the password required attribute on all of these users.
380
381     .PARAMETER ADUsers
382         Similar to UserNames, but this paramter takes an array of ADUsers
383         (Microsoft.ActiveDirectory.Management.ADAccount)
384         and attempts to set the password required field on them.
385
386     .PARAMETER NoOutput
387         This is a switch parameter that prevents displaying function output.
388
389     .EXAMPLE
390     PS C:\Windows\system32> "jbrusoe","krussell" | Set-HSCPasswordRequired
391     Current user: jbrusoe
392     Password Not Required: False
393     *****
394     Current user: krussell
395     Password Not Required: False
396     *****
397     .EXAMPLE

```

```

398     PS C:\Windows\system32> $Jeff = Get-ADUser jbrusoe -Properties
PasswordNotRequired
399     PS C:\Windows\system32> $Kevin = Get-ADUser krussell -Properties
PasswordNotRequired
400     PS C:\Windows\system32> @($Jeff,$Kevin) | Set-HSCPasswordRequired
401     Current user: jbrusoe
402     Password Not Required: False
403     *****
404     Current user: krussell
405     Password Not Required: False
406     *****
407
408 .NOTES
409     Written by: Jeff Brusoe
410     Last Updated by: Jeff Brusoe
411     Last Updated: July 13, 2020
412 #>
413
414 [CmdletBinding(SupportsShouldProcess=$true,
415     ConfirmImpact="Medium")]
416
417 param (
418     [Parameter(ValueFromPipeline=$true,
419         ParameterSetName="ADUserArray",
420         Mandatory=$true,
421         Position=0)]
422     [Microsoft.ActiveDirectory.Management.ADAccount[]]$ADUsers,
423
424     [Parameter(ValueFromPipeline=$true,
425         ParameterSetName="UserNameArray",
426         Mandatory=$true,
427         Position=0)]
428     [string[]]$UserNames,
429
430     [switch]$NoOutput
431 )
432
433 begin
434 {
435     Write-Verbose "Beginning to set password required"
436
437     $Error.Clear()
438
439     if ($null -eq (Get-Module ActiveDirectory))
440     {
441         Write-Verbose "Importing Active Directory Module"
442     }
443 }
444
445 process
446 {
447     Write-Debug $("In process block - Parameter Set Name: " +
$PSCmdlet.ParameterSetName)
448
449     #Get array of ADUsers if a string array is passed in
450     if ($PSCmdlet.ParameterSetName -eq "UserNameArray")
451     {
452         $ADUsers = $null
453
454         Write-Debug "Process Block - If Statement"

```



```

455
456     foreach ($UserName in $UserNames)
457     {
458         try
459         {
460             $ADUsers += Get-ADUser $UserName -Properties PasswordNotRequired -
ErrorAction Stop
461         }
462         catch
463         {
464             Write-Warning "Unable to find user name"
465         }
466     }
467 }
468
469 foreach ($ADUser in $ADUsers)
470 {
471     if (!$NoOutput)
472     {
473         Write-Output $("Current user: " + $ADUser.SamAccountName) | Out-Host
474         Write-Output $("Password Not Required: " + $ADUser.PasswordNotRequired) |
Out-Host
475     }
476
477     try
478     {
479         if ($PSCmdlet.ShouldProcess("Setting password required for " +
$ADUser.SamAccountName))
480         {
481             $ADUser | Set-ADUser -PasswordNotRequired $false -ErrorAction Stop
482         }
483     }
484     catch
485     {
486         Write-Warning "There was an error setting the password not required
field"
487     }
488
489     if (!$NoOutput) {
490         Write-Output "*****" | Out-Host
491     }
492 }
493 }
494
495 end
496 {
497     if ($Error.Count -gt 0)
498     {
499         $Error | FL
500     }
501     else
502     {
503         Write-Verbose "Password required has been set."
504     }
505 }
506 }
507
508
509 #####
510 # Export Functions #

```

```
511 #####
512
513 #Get Functions
514 Export-ModuleMember -Function "Get-HSCDirectoryMapping"
515 Export-ModuleMember -Function "Get-HSCLoggedInUser"
516 Export-ModuleMember -Function "Get-HSCPrimarySMTPAddress"
517
518 #Set Functions
519 Export-ModuleMember -Function "Set-HSCPasswordRequired"
```