```powershell
<#
.SYNOPSIS
  This module contains some common Active Directory functions that are used by many HSC PowerShell files.

.DESCRIPTION
  Active Directory functions included in this module:

  1. Get-HSCDirectoryMapping
  2. Get-HSCLoggedOnUser
  3. Get-HSCPrimarySMTPAddress
  4. Send-HSCNewAccountEmail
  4. Set-HSCGroupMembership
  5. Set-HSCPasswordRequired

.NOTES
  HSC-ActiveDirectoryModule.psm1
  Last Modified by: Jeff Brusoe
  Last Modified: August 4, 2020

  Version: 1.0
#>

[CmdletBinding()]
[Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSAvoidTrailingWhiteSpace","",Justification = "Not relevant")]
[Diagnostics.CodeAnalysis.SuppressMessageAttribute("PSAvoidUsingCmdletAliases","",Justification = "Only MS Provided Aliases are Used")]
param ()

Import-Module ActiveDirectory

Function Get-HSCDirectoryMapping
{
  <#
    .SYNOPSIS
      This function takes a user's distinguished name for input and returns
      the path to the user's home directory.

    .DESCRIPTION
      During the HSC new account creation process, a homedirectory is created for
      all new users. This function determines the correct path to the users homedirectory
      based on a mapping file.

    .OUTPUTS
      This function currently returns a PS Object that contains the path to the
      users's homedirectory and whether it's a complete path. The complete path is being
      deprecated. When that happens, this function will just return a string for the path.

    .EXAMPLE
      PS C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-Repository> $ADUser = Get-ADUser jbrusoe
      PS C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-Repository> Get-HSCDirectoryMapping -UserDN $ADUser.DistinguishedName

      DirectoryPath                                                FullPath
      -------------                                                --------
```

```
      \\hs.wvu-ad.wvu.edu\public\ITS\Network and Voice Services\     True


      PS C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-Repository> $ADUser = Get-ADUser krussell
      PS C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-Repository> Get-HSCDirectoryMapping -UserDN $ADUser.DistinguishedName

      DirectoryPath                                                  FullPath
      -------------                                                  --------
      \\hs.wvu-ad.wvu.edu\public\ITS\support?services\desktop?support\     True

    .PARAMETER UserDN
      This is the distinguished name of the user to determine the home directory
      mapping of.

    .PARAMETER DetermineFullPath
      This parameter is being deprecated and shouldn't be used.

    .NOTES
      Last Modified by: Jeff Brusoe
      Last Modified: August 4, 2020
  #>

  [CmdletBinding()]
  param (
    [Parameter(ValueFromPipeline = $true,
      Mandatory=$true)]
    [string]$UserDN,
    [switch]$DetermineFullPath
  )

  begin
  {
    #Create object to hold directory information
    $HomeDirectoryInfo = New-Object PSObject

    $HomeDirectoryInfo | Add-Member -type NoteProperty -Name DirectoryPath -value $null
    $HomeDirectoryInfo | Add-Member -type NoteProperty -Name FullPath -Value $false
    #In cases where the DN still needs to be parsed up later recursively, the FullPath value is set to false.
    #If the DirectoryPath value is the correct (& final) home directory path, the FullPath value is set to true.

    if ($UserDN.indexOf("CN=") -ge 0)
    {
      #Need to remove this from the DN
      $UserDN = $UserDN.substring($UserDN.indexOf(",")+1).Trim()
      Write-Verbose "Cleaned UserDN: $UserDN"
    }
  }

  process
  {
    #First check DirectoryMapping file for match
    [string]$HomeDirectoryPath = $null

    #Step 1: Check against home directory mapping file.
```

```
107     #$HomeDirectoryMappings = Import-Csv $($MyInvocation.PSScriptRoot +
"\HomeDirectoryMapping.csv")
108     $HomeDirectoryMappings = Import-Csv "C:\Users\microsoft\Documents\GitHub\HSC-
PowerShell-Repository\Create-NewAccount\HomeDirectoryMapping.csv"
109     #$HomeDirectoryMappings = Import-Csv "C:\HSCGitHub\HSC-PowerShell-
Repository\Create-NewAccount\HomeDirectoryMapping.csv"
110
111     [string]$HomeDirectoryPath = ($HomeDirectoryMappings | where {$UserDN -eq
$_.UserDN}).DirectoryPath
112
113     if ([string]::IsNullOrEmpty($HomeDirectoryPath))
114     {
115       [string]$HomeDirectoryPath = ($HomeDirectoryMappings | where {$UserDN -match
$_.UserDN}).DirectoryPath
116       $HomeDirectoryInfo.DirectoryPath = $HomeDirectoryPath
117       $HomeDirectoryInfo.FullPath = $true
118     }
119
120     if ([string]::IsNullOrEmpty($HomeDirectoryPath))
121     {
122       Write-verbose "No match from directory mapping file"
123     }
124     else {
125       $HomeDirectoryInfo.DirectoryPath = $HomeDirectoryPath
126       $HomeDirectoryInfo.FullPath = $true
127     }
128
129     #Step 2: Check against predefined mappings
130     [string]$ParentPath = $null
131
132     if ([string]::IsNullOrEmpty($HomeDirectoryPath))
133     {
134       switch -wildcard ($UserDN)
135       {
136         "*OU=Char_Div*"
137         {
138           $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\Char_Div\"
139           $HomeDirectoryInfo.FullPath = $false
140           $ParentPath = "Char_Div"
141           break
142         }
143         "*OU=ITS*"
144         {
145           $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\ITS\"
146           $HomeDirectoryInfo.FullPath = $false
147           $ParentPath = "ITS"
148           break
149         }
150         "*OU=ADMIN*"
151         {
152           $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\Admin\"
153           $HomeDirectoryInfo.FullPath = $false
154           $ParentPath = "ADMIN"
155           break
156         }
157         "*OU=BASSCI*"
158         {
159           $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\bassci\"
160           $HomeDirectoryInfo.FullPath = $false
161           break
```

```
        }
        "*OU=MBRCC*"
        {
          $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\mbrcc\"
          $HomeDirectoryInfo.FullPath = $false
          break
        }
        "*OU=SOM*"
        {
          $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\som\"
          $HomeDirectoryInfo.FullPath = $false
          break
        }
        "*OU=SON*"
        {
          $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\son\"
          $HomeDirectoryInfo.FullPath = $false
          break
        }
        "*OU=SOP*"
        {
          $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\sop\"
          $HomeDirectoryInfo.FullPath = $false
          break
        }
        "*OU=SPH*"
        {
          $HomeDirectoryInfo.DirectoryPath = "\\hs.wvu-ad.wvu.edu\public\sph\"
          $HomeDirectoryInfo.FullPath = $false
          break
        }
        default
        {
          $HomeDirectoryInfo.DirectoryPath = "NoHomeDirectory"
          $HomeDirectoryInfo.FullPath = $true
        }
      }
    }

    #Step 3: Determine full path if switch not used
    if ($DetermineFullPath -AND !$HomeDirectoryInfo.FullPath -AND
($HomeDirectoryInfo -ne "NoHomeDirectory"))
    {
      $ParsedDN = $UserDN -split ","
      $ParsedDNCount = $ParsedDN.Length
      $AddToPath = $false

      for ($i=$ParsedDNCount-1; $i -ge 0; $i--)
      {
        $ParsedDN[$i] = $ParsedDN[$i] -replace "OU=",""

        if ($ParsedDN[$i] -eq $ParentPath)
        {
          $AddToPath = $true
        }
        elseif ($AddToPath)
        {
          $HomeDirectoryInfo.DirectoryPath = $HomeDirectoryInfo.DirectoryPath + "\"
+ $ParsedDN[$i]+ "\"
        }
```

```
220       }
221       $HomeDirectoryInfo.FullPath = $true
222     }
223   }
224
225   end
226   {
227     return $HomeDirectoryInfo
228   }
229 }
230
231 function Get-HSCLoggedOnUser
232 {
233   <#
234   .SYNOPSIS
235     This function returns the currently logged on user.
236
237   .OUTPUTS
238     Returns a PSCustomObject that has two properties: Logged on username and
    domain.
239
240   .NOTES
241     Tested with the following version of PowerShell:
242     1. 5.1.18362.752
243     2. 7.0.2
244
245     Written by: Jeff Brusoe
246     Last Updated by: Jeff Brusoe
247     Last Updated: June 24, 2020
248   #>
249
250   [CmdletBinding()]
251   [OutputType([PSCustomObject])]
252   param()
253
254   try
255   {
256     $LoggedOnUser = [PSCustomObject]@{
257       UserName = $((Get-ChildItem Env:\USERNAME).Value)
258       Domain = $((Get-ChildItem Env:\USERDOMAIN).Value)
259     }
260
261     return $LoggedOnUser
262   }
263   catch
264   {
265     Write-Warning "Error getting logged on user" | Out-Null
266
267     return $null
268   }
269 }
270
271 function Get-HSCPrimarySMTPAddress
272 {
273   <#
274   .SYNOPSIS
275     This function retrieves the primary SMTP address for AD users.
276
277   .INPUTS
278     This function can take a string(array) or ADUser object(array) and will
```

```
279     get the primary SMTP address for those users.
280
281   .PARAMETER UserNames
282     This parameter takes a string array of users names as input. It will attempt to
    get
283     the primary SMTP address after finding the users.
284
285   .PARAMETER ADUsers
286     Similar to UserNames, but this paramter takes an array of ADUsers
    (Microsoft.ActiveDirectory.Management.ADAccount)
287     and attempts to get their primary SMTP address.
288
289
290   .EXAMPLE
291     PS C:\Windows\system32> "jbrusoe","krussell" | Get-HSCPrimarySMTPAddress
292
293     SamAccountName PrimarySMTPAddress
294     -------------- ------------------
295     jbrusoe        jbrusoe@hsc.wvu.edu
296     krussell       krussell@hsc.wvu.edu
297
298   .EXAMPLE
299     PS C:\Windows\system32> $Jeff = Get-ADUser jbrusoe -Properties proxyAddresses
300     PS C:\Windows\system32> $Kevin = Get-ADUser krussell -Properties proxyAddresses
301     PS C:\Windows\system32> $Jeff,$Kevin | Get-HSCPrimarySMTPAddress
302
303     SamAccountName PrimarySMTPAddress
304     -------------- ------------------
305     jbrusoe        jbrusoe@hsc.wvu.edu
306     krussell       krussell@hsc.wvu.edu
307
308   .NOTES
309     Written by: Jeff Brusoe
310     Last Updated by: Jeff Brusoe
311     Last Updated: July 16, 2020
312   #>
313
314   [CmdletBinding()]
315   [OutputType([PSObject])]
316
317   param (
318     [Parameter(ValueFromPipeline=$true,
319       ParameterSetName="ADUserArray",
320       Mandatory=$true,
321       Position=0)]
322     [Microsoft.ActiveDirectory.Management.ADAccount[]]$ADUsers,
323
324     [Parameter(ValueFromPipeline=$true,
325       ParameterSetName="UserNameArray",
326       Mandatory=$true,
327       Position=0)]
328     [string[]]$UserNames,
329
330     [switch]$NoOutput
331   )
332
333   begin
334   {
335     [psobject[]]$PrimarySMTPAddresses = @()
336   }
```

```
337
338   process
339   {
340     Write-Debug $("In process block - Parameter Set Name: " +
$PSCmdlet.ParameterSetName)
341
342     #Get array of ADUsers if a string array is passed in
343     if ($PSCmdlet.ParameterSetName -eq "UserNameArray")
344     {
345       $ADusers = $null
346       foreach ($UserName in $UserNames)
347       {
348         try
349         {
350           $ADUsers += Get-ADUser $UserName -Properties proxyAddresses -ErrorAction
Stop
351           Write-Verbose "Found User: $UserName"
352         }
353         catch
354         {
355           Write-Warning "Unable to find user name"
356
357           $ADUserObject = New-Object -TypeName PSObject
358           $ADUserObject | Add-Member -MemberType NoteProperty -Name
"SamAccountName" -Value $UserName
359           $ADUserObject | Add-Member -MemberType NoteProperty -Name
"PrimarySMTPAddress" -Value "UserNotFound"
360
361           $PrimarySMTPAddresses += $ADUserObject
362         }
363       }
364     }
365
366     if ($null -ne $ADUsers)
367     {
368       foreach ($ADUser in $ADUsers)
369       {
370         $ADUserObject = New-Object -TypeName PSObject
371         $ADUserObject | Add-Member -MemberType NoteProperty -Name "SamAccountName"
-Value $ADUser.SamAccountName
372
373
374         [string[]]$ProxyAddresses = $ADUser.proxyAddresses
375
376         [string]$PrimarySMTPAddress = $ProxyAddresses -cmatch "SMTP:"
377
378         if ([string]::IsNullOrEmpty($PrimarySMTPAddress))
379         {
380           Write-Verbose "Primary SMTP Address isn't defined"
381           $ADUserObject | Add-Member -MemberType NoteProperty -Name
"PrimarySMTPAddress" -Value $null
382         }
383         else {
384           Write-Verbose $("Current User" + $ADUser.SamAccountName)
385           Write-Verbose "Primary SMTP Address: $PrimarySMTPAddress"
386
387           $PrimarySMTPAddress = ($PrimarySMTPAddress -replace "SMTP:","").Trim()
388
389           $ADUserObject | Add-Member -MemberType NoteProperty -Name
"PrimarySMTPAddress" -Value $PrimarySMTPAddress
```

```
390         }
391
392         $PrimarySMTPAddresses += $ADUserObject
393       }
394     }
395     else
396     {
397       Write-Warning "ADUser object is null"
398     }
399   }
400
401   end
402   {
403     return $PrimarySMTPAddresses
404   }
405 }
406
407 function Get-HSCADUserFromString
408 {
409   <#
410     .SYNOPSIS
411       This function returns an AD User array from a passed in string array
412
413     .OUTPUTS
414       [Microsoft.ActiveDirectory.Management.ADAccount[]]
415
416     .PARAMETER UserNames
417       This is the string array of usernames to get the corresponding AD user
    objects for.
418
419     .EXAMPLE
420       PS C:\Windows\system32> "jbrusoe","krussell" | Get-HSCADUserFromString
421
422       DistinguishedName : CN=Jeff Brusoe,OU=Network Systems,OU=Network and Voice
423                 Services,OU=ITS,OU=ADMIN,OU=HSC,DC=HS,DC=wvu-ad,DC=wvu,DC=edu
424       Enabled           : True
425       GivenName         : Jeff
426       Name              : Jeff Brusoe
427       ObjectClass       : user
428       ObjectGUID        : 04d2e77c-b50e-40ce-9e94-c819e2ed85e0
429       SamAccountName    : jbrusoe
430       SID               : S-1-5-21-865322659-4255640127-3857865232-2111
431       Surname           : Brusoe
432       UserPrincipalName : jbrusoe@hsc.wvu.edu
433
434       DistinguishedName : CN=Kevin Russell,OU=Desktop Support,OU=Support
435                 Services,OU=ITS,OU=ADMIN,OU=HSC,DC=HS,DC=wvu-ad,DC=wvu,DC=edu
436       Enabled           : True
437       GivenName         : Kevin
438       Name              : Kevin Russell
439       ObjectClass       : user
440       ObjectGUID        : f1db51f3-2a85-49d5-9ec6-19951d2257ae
441       SamAccountName    : krussell
442       SID               : S-1-5-21-865322659-4255640127-3857865232-2123
443       Surname           : Russell
444       UserPrincipalName : krussell@hsc.wvu.edu
445
446     .NOTES
447       Last updated by: Jeff Brusoe
448       Last Updated: July 27, 2020
```

```
449   #>
450
451   [CmdletBinding()]
452   param (
453     [Parameter(Mandatory=$true,
454       ValueFromPipeline = $true)]
455     [string[]]$UserNames
456   )
457
458   begin
459   {
460     Write-Verbose "Beginning to Search for AD User Objects"
461
462     [Microsoft.ActiveDirectory.Management.ADAccount[]]$ADUsers = $null
463   }
464
465   process
466   {
467     foreach ($UserName in $UserNames)
468     {
469       try {
470         Write-Verbose "Attempting to find user: $UserName"
471         $ADUser = Get-ADUser $UserName -ErrorAction Stop
472
473         if ($null -ne $ADUser)
474         {
475           $ADUsers += $ADUser
476         }
477       }
478       catch {
479         Write-Warning "Unable to find user"
480       }
481     }
482   }
483
484   end
485   {
486     return $ADUsers
487   }
488 }
489
490 function Send-HSCNewAccountEmail
491 {
492   <#
493     .SYNOPSIS
494       This function sends the CSC a new account creation email.
495
496     .DESCRIPTION
497       This function sends a confirmation email to the department's CSC
498       after successfully creating a new account. It is not intended to run
499       in a stand-alone mode.
500
501     .OUTPUTS
502       The output of this function is an email such as the following:
503
504       Subject: New Account Created: emily.battle@hsc.wvu.edu
505
506       Message Body:
507       The following account has been created in the HS domain and email system:
508       Username: ehb10007
```

```
509       Email Address: emily.battle@hsc.wvu.edu
510
511     .PARAMETER CSCEmail
512       The CSC email address who will be receiving the email.
513
514     .PARAMETER SamAccountName
515       This is the SamAccountName f the newly created account.
516
517     .PARAMETER PrimarySMTPAddress
518       This is the PrimarySMTPAddress of the newly created account.
519
520     .PARAMETER SMTPServer
521       The SMTP server to relay mail through.
522
523     .PARAMETER From
524       The email address sending the message
525
526     .NOTES
527       Writen by: Jeff Brusoe
528       Last Updated: August 4, 2020
529   #>
530
531   [CmdletBinding()]
532   param (
533     [Parameter(Mandatory=$true)]
534     [string]$CSCEmail,
535     [Parameter(Mandatory=$true)]
536     [string]$SamAccountName,
537     [Parameter(Mandatory=$true)]
538     [string]$PrimarySMTPAddress,
539     [string]$SMTPServer = "Hssmtp.hsc.wvu.edu",
540     [string]$From = "microsoft@hsc.wvu.edu"
541   )
542
543   process
544   {
545     $MsgSubject = "New Account Created: $PrimarySMTPAddress"
546     $MsgBody = "The following account has been created in the HS domain and email
system:`nUsername: $SamAccountName`nEmail Address: $PrimarySMTPAddress"
547
548     Write-Output "Message Subject: $MsgSubject" | Out-Host
549     Write-Output "Message Body: $MsgBody" | Out-Host
550
551     $Recipients = @($CSCEmail,"microsoft@hsc.wvu.edu","jbrusoe@hsc.wvu.edu")
552
553     try {
554       Send-MailMessage -to $Recipients -From $From -SMTPServer $SMTPServer -Subject
$MsgSubject -Body $MsgBody -ErrorAction Stop #UseSSL -port 587
555     }
556     catch {
557       Write-Warning "Unable to email message confirming account createion"
558     }
559   }
560 }
561
562 function Set-HSCGroupMembership
563 {
564   <#
565     .SYNOPSIS
566       The purpsoe of this function is to add a new HSC AD user to the
```

```
567       correct AD groups.
568
569     .PARAMETER UserDN
570       This parameter is the users's distinguished name. It is parsed up
571       to determine the OU that it is in and to figure out the group mapping.
572
573     .DESCRIPTION
574       Steps to search for groups
575       1. Determine user OU parent DN.
576       2. Add users to the DUO MFA and O365 license groups
577       3. Add groups based on group mapping file
578
579     .NOTES
580       Last updated by: Jeff Brusoe
581       Last Updated: July 28, 2020
582   #>
583
584   [CmdletBinding(SupportsShouldProcess=$true,
585     ConfirmImpact="Medium")]
586   [OutputType([String])]
587   param (
588     [Parameter(ValueFromPipeline = $true)]
589     [string]$UserDN = $null
590   )
591
592   begin
593   {
594     [string]$UserOU = $null
595   }
596
597   process
598   {
599     if ($UserDN.indexOf("CN=") -ge 0)
600     {
601       # Step 1: Determine parent container OU DN
602       $UserOU = $UserDN.substring($UserDN.indexOf(",")+1).Trim()
603     }
604     else {
605       Write-Warning "Invalid User DN... Unable to add user to groups..."
606       return
607     }
608
609     Write-Verbose "User OU: $UserOU"
610
611     #Step 2: Add users to the DUO MFA and O365 license groups
612     #Add to O365 License Group
613     Write-Verbose "Add user to Office 365 Base Licensing Group"
614     $O365LicenseGroup = "CN=Office 365 Base Licensing Group,OU=Applications,OU=HSC
    AD Groups,OU=HSC,DC=HS,DC=wvu-ad,DC=wvu,DC=edu"
615
616     try {
617       if ($PSCmdlet.ShouldProcess("Adding user to O365 License Group"))
618       {
619         $O365LicenseGroup | Add-ADGroupMember -Members $UserDN -ErrorAction Stop
620       }
621
622       Write-Verbose "Successfully added user to OFfice 365 Base Licensing Group"
623     }
624     catch {
625       Write-Warning "Unable to add user to Office 365 Base Licesning Group"
```

```
626     }
627
628     #Add to MFA group
629     Write-Verbose "Adding user to DUO MFA Group"
630     try {
631       $DUOMFAGroup = Get-ADGroup "HSC DUO MFA" -ErrorAction Stop
632
633       if ($PSCmdlet.ShouldProcess("Adding user to DUO MFA Group"))
634       {
635         $DUOMFAGroup | Add-ADGroupMember -Members $UserDN -ErrorAction Stop
636       }
637
638       Write-Verbose "Successfully added user to MFA Group"
639     }
640     catch {
641       Write-Warning "Unable to add user to DUO MFA group"
642     }
643
644     #Step 3: Check group mapping file
645     #$GroupMappingFile = "C:\HSCGitHub\HSC-PowerShell-Repository\Create-
NewAccount\HSCGroupMapping.csv"
646     $GroupMappingFile = "C:\Users\microsoft\Documents\GitHub\HSC-PowerShell-
Repository\Create-NewAccount\HSCGroupMapping.csv"
647     $GroupMappings = Import-Csv $GroupMappingFile
648
649     [string]$GroupsToAdd = ($GroupMappings | where {$UserOU -eq $_.UserOU}).Groups
650
651     if ([string]::IsNullOrEmpty($GroupsToAdd))
652     {
653       [string]$GroupsToAdd = ($GroupMappings | where {$UserOU -match
$_.UserOU}).Groups
654     }
655
656     if ([string]::IsNullOrEmpty($GroupsToAdd))
657     {
658       Write-Verbose "No match from group mapping file"
659     }
660     else {
661       Write-Verbose "Groups to Add: $GroupsToAdd"
662
663       [string[]]$Groups = $GroupsToAdd -split ";"
664
665       foreach ($Group in $Groups)
666       {
667         $GroupDN = "CN=" + $Group + "," + $UserOU
668
669         try {
670           $GroupToAdd = Get-ADGroup -Identity $GroupDN -ErrorAction Stop
671         }
672         catch {
673           Write-Warning "Unable to find Group: $GroupDN"
674           $GroupToAdd = $null
675         }
676
677         if ($null -ne $GroupToAdd)
678         {
679           $User = Get-ADUser $UserDN #This may not be needed
680
681           $GroupToAdd | Add-ADGroupMember -Members $User.DistinguishedName
682         }
```

```
      }
    }
  }
}

Function Set-HSCPasswordRequired
{
  <#
  .SYNOPSIS
    This function sets the password required for a user

  .INPUTS
    This function can take a string(array) or ADUser object(array) and will
    set the password required attribute for those users.

  .PARAMETER UserNames
    This parameter takes a string array of users names as input. It will attempt to
    set the password required attribute on all of these users.

  .PARAMETER ADUsers
    Similar to UserNames, but this paramter takes an array of ADUsers
(Microsoft.ActiveDirectory.Management.ADAccount)
    and attempts to set the password required field on them.

  .PARAMETER NoOutput
    This is a switch parameter that prevents displaying function output.

  .EXAMPLE
    PS C:\Windows\system32> "jbrusoe","krussell" | Set-HSCPasswordRequired
    Current user: jbrusoe
    Password Not Required: False
    ****************************
    Current user: krussell
    Password Not Required: False
    ****************************

  .EXAMPLE
    PS C:\Windows\system32>  $Jeff = Get-ADUser jbrusoe -Properties
PasswordNotRequired
    PS C:\Windows\system32> $Kevin = Get-ADUser krussell -Properties
PasswordNotRequired
    PS C:\Windows\system32> @($Jeff,$Kevin) | Set-HSCPasswordRequired
    Current user: jbrusoe
    Password Not Required: False
    ****************************
    Current user: krussell
    Password Not Required: False
    ****************************

  .NOTES
    Written by: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: July 13, 2020
  #>

  [CmdletBinding(SupportsShouldProcess=$true,
    ConfirmImpact="Medium")]

  param (
    [Parameter(ValueFromPipeline=$true,
```

```
      ParameterSetName="ADUserArray",
      Mandatory=$true,
      Position=0)]
    [Microsoft.ActiveDirectory.Management.ADAccount[]]$ADUsers,

    [Parameter(ValueFromPipeline=$true,
      ParameterSetName="UserNameArray",
      Mandatory=$true,
      Position=0)]
    [string[]]$UserNames,

    [switch]$NoOutput
  )

  begin
  {
    Write-Verbose "Beginning to set password required"

    $Error.Clear()

    if ($null -eq (Get-Module ActiveDirectory))
    {
      Write-Verbose "Importing Active Diretory Module"
    }
  }

  process
  {
    Write-Debug $("In process block - Parameter Set Name: " +
$PSCmdlet.ParameterSetName)

    #Get array of ADUsers if a string array is passed in
    if ($PSCmdlet.ParameterSetName -eq "UserNameArray")
    {
      $ADusers = $null

      Write-Debug "Process Block - If Statement"

      foreach ($UserName in $UserNames)
      {
        try
        {
          $ADUsers += Get-ADUser $UserName -Properties PasswordNotRequired -
ErrorAction Stop
        }
        catch
        {
          Write-Warning "Unable to find user name"
        }
      }
    }

    foreach ($ADUser in $ADUsers)
    {
      if (!$NoOutput)
      {
        Write-Output $("Current user: " + $ADUser.SamAccountName) | Out-Host
        Write-Output $("Password Not Required: " + $ADUser.PasswordNotRequired) |
Out-Host
      }
```

```powershell

      try
      {
        if ($PSCmdlet.ShouldProcess("Setting password required for " + $ADUser.SamAccountName))
        {
          $ADUser | Set-ADUser -PasswordNotRequired $false -ErrorAction Stop
        }
      }
      catch
      {
          Write-Warning "There was an error setting the password not required field"
      }

      if (!$NoOutput) {
        Write-Output "*****************************" | Out-Host
      }
    }
  }

  end
  {
    if ($Error.Count -gt 0)
    {
      $Error | FL
    }
    else
    {
      Write-Verbose "Password required has been set."
    }
  }
}

####################
# Export Functions #
####################

#Get Functions
Export-ModuleMember -Function "Get-HSCDirectoryMapping"
Export-ModuleMember -Function "Get-HSCLoggedOnUser"
Export-ModuleMember -Function "Get-HSCPrimarySMTPAddress"
Export-ModuleMember -Function "Get-HSCADUserFromString"

#Send functions
Export-ModuleMember -Function "Send-HSCNewAccountEmail"

#Set Functions
Export-ModuleMember -Function "Set-HSCGroupMembership"
Export-ModuleMember -Function "Set-HSCPasswordRequired"
```