Quest

Microsoft Entra ID Intune Device Migration

Quick Start Guide

If you have any questions regarding your potential use of this material, contact:

Quest Software Inc.

Attn: LEGAL Dept

4 Polaris Way

Aliso Viejo, CA 92656

Refer to our Web site (https://www.quest.com) for regional and international office information.

Patents

Quest Software is proud of our advanced technology. Patents and pending patents may apply to this product. For the most current information about applicable patents for this product, please visit our website at https://www.quest.com/legal.

Trademarks

Quest, the Quest logo, and Join the Innovation are trademarks and registered trademarks of Quest Software Inc. For a complete list of Quest marks, visit https://www.quest.com/legal/trademark-information.aspx. All other trademarks and registered trademarks are property of their respective owners.

Legend

CAUTION: A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.

IMPORTANT, **NOTE**, **TIP**, **MOBILE**, or **VIDEO**: An information icon indicates supporting information.

Microsoft Entra ID Intune Device Migration Quick Start Guide
Updated - April 2024

Contents

Introduction

On Demand Migration for Active Directory (ODMAD) supports Microsoft Entra ID Join device migration for devices running Windows 10 or Windows 11 while preserving the User Profiles and File/Folder Security Permissions.

ODMAD successfully migrates these devices to the target Microsoft Entra ID using the default ODMAD settings, including migrating devices that are already Intune-enrolled and devices that were originally provisioned using Autopilot. In addition to migrating the devices to Microsoft Entra ID, a best practice is to also clear previous Autopilot and Intune settings to allow successful Intune enrollment and management in the target.

This step-by-step guide walks through how to perform Intune managed device migration between two Microsoft Entra ID (Cloud Only) tenants.

This guide is a supplementary document to the Microsoft Entra ID Device Join Quick Start Guide.

Topics

This guide covers the following topics:

- Requirements
- Intune/Autopilot Workstation Cutover High-Level Process
- High level Custom Task Explanation
- Implementation Process
- Intune Cutover Run Book

Requirements

General

- Client is licensed for On Demand Migration Active Directory and Directory Sync
- One Global Administrator Account for each Microsoft 365 tenant

  Accounts

  Microsoft Entra ID Application Account
- An account with Global Administrator Role is required to grant permissions and establish connection when adding a Cloud Environment.

  Microsoft Entra ID PowerShell Accounts
- Three (3) PowerShell accounts are automatically created to read and update objects in the cloud. To do this an OAuth token is used from the account used to add the Cloud Environment.
- These PowerShell accounts do not require any Microsoft 365 licenses.

Intune/Autopilot Workstation Cutover High-Level Process

The high-level process requires the modification of the Default Microsoft Entra ID Cutover action in ODMAD. There are additional tasks that need to be added:

- BT-EntraIDCutoverPreflight – Default Task
- BT-DownloadReACLConfig – Default Task
- BT-ReACLPrepareWin10Profiles – Default Task
- Autopilot Cleanup – Removes the Autopilot registry keys from the workstation. Should be done after the workstation has been removed from Enrolled Devices in the source tenant.
- Intune Cleanup – Removes the Registry keys and associated scheduled tasks from the workstation related to source tenant Intune Enrollment.
- SetUserEmailValues - Post migration, the UserEmail Registry value is automatically set to the Bulk Enrollment package, which can prevent the workstation from enrolling into the target Intune. This task creates a PowerShell script on the workstation and creates a Scheduled Task that will run the script after the user has logged on post migration. The script will set the registry value to the target user account.
- BitlockerBackupToEntraID (Only required if source workstations are BitLocker Enabled) – If the workstation is BitLocker enabled in the source, the Recovery key is not automatically transferred to the target Microsoft Entra ID. This task creates a PowerShell script on the workstation and creates a Scheduled Task that will run the script after the user has logged on post migration. The script will escrow the existing recovery key from workstation and write it to the target Microsoft Entra ID account.
- SetPrimaryUser (Optional) – Post migration, the Primary User field in the target Intune will be blank. This task creates a PowerShell script on the workstation and creates a Scheduled Task that will run the script after the user has logged on post migration. The script will set the Primary User value to the target user account. This script requires an additional Enterprise Application to be created and requires ODMAD Global Variables to be configured.
- CleanupLocalAdministratorsGroup (Optional) – If the source user was an Administrator on the machine, the Re-ACL process will put the target user in the Administrators group. This task will remove users from the Local Administrator Group.
- BT-EntraIDCutover – Default Task

High level Custom Task Explanation

Autopilot Cleanup

This script runs automatically when the ODMAD task runs.

This script searches the HKLM\SOFTWARE\Microsoft\Provisioning and the HKLM\SOFTWARE\Microsoft\Provisioning\Diagnostics for child keys that have Autopilot in their names. It will then delete these values and keys from the workstation.

Autopilot Cleanup.txt

```
Param (
)
$output = New-Object BinaryTree.ADM.Agent.PSHelpers.PSOutput

### Script Configurable Parameters ###

# Parent Regisry Keys to search #
$RegistryKeys = "HKLM:\SOFTWARE\Microsoft\Provisioning" ,
"HKLM:\SOFTWARE\Microsoft\Provisioning\Diagnostics"

#######################################

Write-Output "Evaluating $($RegistryKeys.Count) Registry Hive(s)"
ForEach ($Key in $RegistryKeys) {
                Write-Output " Processing: $($Key)"
                If (Test-Path -Path $Key) {
                   $StoreKey = Get-ChildItem -Path $Key | Where-Object {$_.Name -like
'*Autopilot*'}
                   Write-Output "  Processing $($Storekey.Name.Count) Registry Entries"
                   ForEach($S in $StoreKey){
                     Write-Output "    $($S)"
                   }
                   Write-Output "  Removing Registry Keys"
                   ForEach($S in $Storekey){
                      Write-Output "    Removing: $($S)"
                      $S | Remove-Item -Recurse -Force -Confirm:$false -ErrorAction
SilentlyContinue
                      $KeyTest = Get-Item -Path Registry::$S -ErrorAction
SilentlyContinue
                      If([string]::IsNullOrWhitespace($KeyTest)){
                         Write-Output "      Delete Suceeded"
                      }
                      Else{
                         Write-Output "      Delete Failed"
                         $KeyTest = $Null
                   `  }
                   }
    }
}

return ($output)
```

Intune Cleanup

To allow enrolling the workstation into the target Intune, it is important to remove the source Intune Enrollment information. Otherwise, the workstation thinks that it is already part of an Intune Enrollment and will not try to enroll in the target.

This script will find the Intune Enrollment ID from the Scheduled Task that Intune uses. The script then searches the following parent registry keys for the EnrollmentID and removes the registry values and Keys:

- HKLM:\SOFTWARE\Microsoft\Enrollments
- HKLM:\SOFTWARE\Microsoft\Enrollments\Status
- HKLM:\SOFTWARE\Microsoft\EnterpriseResourceManager\Tracked
- HKLM:\SOFTWARE\Microsoft\PolicyManager\AdmxInstalled
- HKLM:\SOFTWARE\Microsoft\PolicyManager\Providers
- HKLM:\SOFTWARE\Microsoft\Provisioning\OMADM\Accounts
- HKLM:\SOFTWARE\Microsoft\Provisioning\OMADM\Logger
- HKLM:\SOFTWARE\Microsoft\Provisioning\OMADM\Sessions

  The script will also remove the Intune Enrollment Scheduled Task (this task will automatically get recreated when the machine joins the target tenant) and will remove the existing Intune Certificate from the Computer Certificate store.

Intune Cleanup.txt

```
Param (
)
$output = New-Object BinaryTree.ADM.Agent.PSHelpers.PSOutput

### Script Configurable Parameters ###

# Declare Parent Registry Keys  to search for EnrollmentID #
$RegistryKeys = "HKLM:\SOFTWARE\Microsoft\Enrollments",
"HKLM:\SOFTWARE\Microsoft\Enrollments\Status","HKLM:\SOFTWARE\Microsoft\EnterpriseReso
urceManager\Tracked", "HKLM:\SOFTWARE\Microsoft\PolicyManager\AdmxInstalled",
"HKLM:\SOFTWARE\Microsoft\PolicyManager\Providers","HKLM:\SOFTWARE\Microsoft\Provision
ing\OMADM\Accounts", "HKLM:\SOFTWARE\Microsoft\Provisioning\OMADM\Logger",
"HKLM:\SOFTWARE\Microsoft\Provisioning\OMADM\Sessions"


#######################################

Write-Output "Getting Intune Enrollment ID:"
### Find the Intune EnrollmentID from Intune Enrollment Scheduled Task ###
$EnrollmentID = Get-ScheduledTask | Where-Object {$_.TaskPath -like
"*Microsoft*Windows*EnterpriseMgmt*"} | Select-Object -ExpandProperty TaskPath -Unique
| Where-Object {$_ -like "*-*-*"} | Split-Path -Leaf
Write-Output "  Intune Enrollment ID: $($EnrollmentID)"
If([string]::IsNullOrWhitespace($EnrollmentID)){
    Write-Output "  No Intune Enrollment ID Found..Terminating"
    Exit 0
}

### Remove Intune Certificate from Certificate Store ###
Write-Output "Removing Intune MDM Certificate"

If((Get-ChildItem -Path Cert:\LocalMachine\My | Where-Object {$_.Issuer -match "Intune
```

Intune Cleanup.txt

```
MDM"})){
    Write-Output "  Intune MDM Certificate Found..Collecting Certificate Information"
    $IntuneCert = Get-ChildItem -Path Cert:\LocalMachine\My | Where-Object {$_.Issuer -
match "Intune MDM"}
    ForEach($Cert in $IntuneCert){
        Write-Output "  Found $($Cert.Subject)..Certificate will be removed"
        $Cert | Remove-Item
        $IntuneCertTest = Get-ChildItem -Path Cert:\LocalMachine\My | Where-Object
{$_.Subject -match $Cert.Subject}
        If([string]::IsNullOrWhitespace($IntuneCertTest)){
        Write-Output "    Delete Succeeded"
        }
        Else{Write-Output "    Delete Failed"}
        $IntuneCertTest = $Null

    }
}
Else {Write-Output "Intune MDM Certificate Not Found"}


### Delete Scheduled Tasks ###
Write-Output "Finding Enrollment Scheduled Tasks"
$ScheduledTasks = Get-ScheduledTask | Where-Object {$_.Taskpath -match $EnrollmentID}
Write-Output "  Evaluating $($ScheduledTasks.count) Scheduled Tasks:"
ForEach($Task in $ScheduledTasks){
    Write-Output "    Found Task: $($Task.Taskname) with path: $($Task.TaskPath)"
}

Write-Output "Removing Intune Enrollment Scheduled Tasks"
ForEach($T in $ScheduledTasks){
    Write-Output " Removing $($T.Taskname)"
    #Get-ScheduledTask | Where-Object {$_.Taskname -match $T.TaskName}
    Get-ScheduledTask | Where-Object {$_.TaskName -match $T.TaskName} | Unregister-
ScheduledTask -Confirm:$false
    $TestScheduledTask = Get-ScheduledTask | Where-Object {$_.Taskname -match
$T.TaskName}
    If([string]::IsNullOrWhitespace($TestScheduledTask)){
        Write-Output "    Delete Succeeded"
    }
    Else{Write-Output "    Delete Failed"}
    $TestScheduledTask = $Null
}


### Search Registry keys for EnrollmentID and Remove Registry Keys
Write-Output "Removing Enrollment Registry Keys"
Write-Output "  Evaluating $($RegistryKeys.Count) Registry Hive(s)"
Foreach ($K in $RegistryKeys) {
    Write-Output "    $($K)"
}
Foreach ($Key in $RegistryKeys) {
    If (Test-Path -Path $Key) {
       Write-Output "  Found: $($Key)"
```

Intune Cleanup.txt

```
        Get-ChildItem -Path $Key | Where-Object {$_.Name -match $EnrollmentID} | Remove-
Item -Recurse -Force -Confirm:$false -ErrorAction SilentlyContinue
    }
    If (Test-Path -Path $Key){
       Write-Output "    Deleted Failed"
    }
    Else{Write-Output "    Deleted Succeeded"}
}

return ($output)
```

SetUserEmailValues

When the machine enrolls in the target Intune, it will look for an Intune Licensed user in M365 using the UserEmail value found in the workstation registry. By default, this value is set to the Bulk Enrollment user, which does not have the relevant license, and prevents the Intune service from running correctly.

This script creates a separate PowerShell script on the workstation called UpdateCloudJoinInfo.ps1 in the ODMAD agent folder and creates a Scheduled Task to execute UpdateCloudJoinInfo.ps1 when the first target user logs on.

When the UpdateCloudJoinInfo script runs during the first login post-migration, it will update the UserEmail value in the following registry key, setting it to the UPN of the logged-on target user.

- HKLM:\System\CurrentControlSet\Control\CloudDomainJoin\JoinInfo

  The script will also create a log file in the ODM agent Files folder and then perform cleanup to remove the Scheduled Task and remove the script itself.

SetUserEmailValues.txt

```
Param (
)

$output = New-Object BinaryTree.ADM.Agent.PSHelpers.PSOutput
Try{
   $ODMADService = Get-Service -Name ODMActiveDirectory -ErrorAction SilentlyContinue
   }
Catch{
    Write-Output "Error Retrieving Service Status...Terminating with error: $($Error)"
    Exit 1
    }
If($ODMADService){
    Write-Output "ODM AD Agent Service Found...Finding ODM AD Agent Service Path"
    $ODMADServicePath = (Get-ItemProperty -Path
HKLM:SYSTEM\CurrentControlSet\Services\ODMActiveDirectory).ImagePath
    $ODMAgentPath = Split-Path $ODMADServicePath
    $ODMAgentPath = $ODMAgentPath.Trim("`"")
    Write-Output "ODM AD Service Path: $($ODMAgentPath)"
}
Else{
```

SetUserEmailValues.txt

```
    Write-Output "No ODM Agent Service Found...Terminating"
    Exit 1
    }

$ScriptName = "UpdateCloudJoinInfo.ps1"


##### The $Script contains the PS script that gets create in the ODMAD agent folder.
$Script = @"

`$TranscriptFile = "`$(`$ODMAgentPath)\Files\PowerShell-`$(Get-Date -f yyyyMMdd-HHMM)-
UpdateCloudDomainJoinReg.log"
Start-Transcript -Path `$TranscriptFile

Try{
   `$ODMADService = Get-Service -Name ODMActiveDirectory
   }
Catch{
     Write-Output "Error Retrieving Service Status...Terminating with error:
`$(`$Error)"
     Exit 1
     {
If(`$ODMADService){
    Write-Output "ODM AD Agent Service Found...Finding ODM AD Agent Service Path"
    `$ODMADServicePath = (Get-ItemProperty -Path
HKLM:SYSTEM\CurrentControlSet\Services\ODMActiveDirectory).ImagePath
    `$ODMAgentPath = Split-Path `$ODMADServicePath
    `$ODMAgentPath = `$ODMAgentPath.Trim("``"")
    Write-Output "ODM AD Service Path: `$(`$ODMAgentPath)"
}
Else{
    Write-Output "No ODM Agent Service Found...Terminating"
    Exit 1
    }

# Get Workgroup/AD User
`$CurrentLoggedOnUser = (Get-CimInstance -Classname Win32_ComputerSystem | Select-
Object -expand UserName)
Write-Output "  DEBUG: Current Logged On User: `$(`$CurrentLoggedOnUser)"
`$LoggedonUserArray = `$CurrentLoggedOnUser.Split("\")
`$LoggedonUser_Username = `$LoggedonUserArray[1]
Write-Output "  DEBUG: Current Logged On Username: `$(`$LoggedonUser_Username)"
`$LoggedonUser_Domain = `$LoggedonUserArray[0]
Write-Output "  DEBUG: Current Logged On Domain: `$(`$LoggedonUser_Domain)"
If (![String]::IsNullOrWhiteSpace(`$CurrentLoggedOnUser)))
    {
        `$CurrentUser = Get-Itemproperty "Registry::\HKEY_USERS\S-1-12*\Volatile
Environment"|Where-Object {`$_.USERDOMAIN -match `$LoggedonUser_Domain}
         Write-Output "  DEBUG: CurrentUser: `$(`$CurrentUser)"
        If (![String]::IsNullOrWhiteSpace(`$CurrentUser))
            {
                `$CurrentLoggedOnUser =
"`$(`$CurrentUser.USERDOMAIN)\`$(`$CurrentUser.USERNAME)"
```

SetUserEmailValues.txt

```
                Write-Output "  DEBUG: Current Logged On User:
`$(`$CurrentLoggedOnUser)"
                `$CurrentLoggedOnUserSID = split-path `$CurrentUser.PSParentPath -leaf
                Write-Output "  DEBUG: Current Logged On User SID:
`$(`$CurrentLoggedOnUserSID)"
                If(`$CurrentUser.USERDOMAIN -match `$LoggedonUser_Domain)
                    {
                        `$UPNKeys = `$(reg query
hklm\SOFTWARE\Microsoft\IdentityStore\LogonCache /reg:64).Split
([Environment]::NewLine)| where{`$_ -ne ""}
                         Write-Output "  DEBUG: UPN Keys (Can be Multi-Valued):
`$(`$UPNKeys)"
                        ForEach (`$item in `$UPNKeys)
                            {
                                `$UPN = reg @
('query',"`$item\Sid2Name\`$CurrentLoggedOnUserSID",'/v','IdentityName','/reg:64')
                                 Write-Output "  DEBUG: UPN: `$(`$UPN)"
                                If (`$LASTEXITCODE -eq 0){`$CurrentLoggedOnUserUPN =
(`$UPN[2] -split ' {2,}')[3] ; Break}
                            }
                    }
            }
    }

Write-Output "  DEBUG: Current Logged On User UPN: `$(`$CurrentLoggedOnUserUPN)"
`$LoggedonUser = `$CurrentLoggedOnUserUPN
Write-Output "  DEBUG: Logged On User: `$(`$LoggedonUser)"

## Get Tenant ID from Registry
`$TenantInfoKey = "HKLM:\System\CurrentControlSet\Control\CloudDomainJoin\TenantInfo"
If(Test-Path -Path `$TenantInfoKey) {
    `$TenantInfo = Get-ChildItem -Path `$TenantInfoKey
    `$TenantID = `$TenantInfo.Name.split("\")[-1]
     Write-Output " TenantID: `$(`$TenantID)"
}

`$JoinInfoPath = "HKLM:\System\CurrentControlSet\Control\CloudDomainJoin\JoinInfo"
If(Test-Path -Path `$JoinInfoPath) {
    `$JoinInfo = Get-ChildItem -Path `$JoinInfoPath
    `$JoinInfoGUID = `$JoinInfo.Name.Split("\")[-1]
    `$ValuePath = `$JoinInfoPath+"\"+(`$JoinInfo.Name.Split("\")[-1])
    `$UserEmailValue = (Get-ItemProperty -path `$ValuePath -Name UserEmail)
     Write-Output "  Current UserEmail Value: `$(`$UserEmailValue)"
    If(`$UserEmailValue.UserEmail -ne `$LoggedonUser) {
        Write-Output "  DEBUG: Will Change Value"
        Set-ItemProperty `$ValuePath -Name UserEmail -Value `$LoggedOnUser}
}

### Clean Up Environment
Write-Output "Cleaning Up Script Environment, Deleteing files and removing Scheduled
Task"
```

SetUserEmailValues.txt

```
Unregister-ScheduledTask -TaskName "Update Cloud Join Info" -Confirm:`$false
Remove-Item -path "`$ODMAgentPath\$($ScriptName)" -Force
Stop-Transcript

"@

#############################################################################################
######################

### Create Powershell Script
$AgentPath = "$ODMAgentPath\"
$ScriptFullName = $AgentPath+$ScriptName
If(!(Test-Path $ScriptFullName)) {
    New-item -path $AgentPath -Name $ScriptName -Type "File" -Value $Script
}

$TaskName = "Update Cloud Join Info"
$Argument = "-ExecutionPolicy Bypass -File `"$($ODMAgentPath)\$($ScriptName)`""
$Action = New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument $Argument
$Settings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries
$Principal = New-ScheduledTaskPrincipal -UserId "LOCALSERVICE" -LogonType
ServiceAccount
$Trigger = New-ScheduledTaskTrigger -Atlogon
$Trigger.Delay = "PT8M"
$ScheduledTask = New-ScheduledTask -Action $Action -Trigger $Trigger -Settings
$Settings
# Register Scheduled Task
Register-ScheduledTask -TaskName $TaskName -InputObject $ScheduledTask -User "NT
AUTHORITY\SYSTEM" -Force


return ($output)
```

BitlockerBackupToEntraID (Optional)

When a machine is BitLocker enabled in the source Environment, the key is stored in the source Microsoft Entra ID. During the Workstation migration process the BitLocker key is not automatically migrated into the target Environment. To ensure that the recovery key is stored in the target tenant, this task will escrow the BitLocker key from the workstation and push into the target tenant post migration.

This script creates a separate PowerShell script on the workstation called BackupBitlockerKeyToADD.ps1 in the ODMAD agent folder and creates a Scheduled Task to execute BackupBitlockerKeyToADD.ps1 when the first target user logs on.

When the BackupBitlockerKeyToADD script runs during the first login post-migration, it will escrow the BitLocker recovery keys from the machine and store them in the Microsoft Entra ID object of the logged-on user and become viewable in the target Intune tenant.

The script will also create a log file in the ODM agent Files folder and then perform cleanup to remove the Scheduled Task and remove the script itself.

BackupBitlockerKeytoAAD.txt

```
Param (
)

$output = New-Object BinaryTree.ADM.Agent.PSHelpers.PSOutput


$ScriptName = "BackupBitlockerKeyToADD.ps1"

$BacktoAAD = @"

ITry{
   `$ODMADService = Get-Service -Name ODMActiveDirectory
   }
Catch{
     Write-Output "Error Retrieving Service Status...Terminating with error:
`$(`$Error)"
     Exit 1
     {
If(`$ODMADService){
    Write-Output "ODM AD Agent Service Found...Finding ODM AD Agent Service Path"
    `$ODMADServicePath = (Get-ItemProperty -Path
HKLM:SYSTEM\CurrentControlSet\Services\ODMActiveDirectory).ImagePath
    `$ODMAgentPath = Split-Path `$ODMADServicePath
    `$ODMAgentPath = `$ODMAgentPath.Trim("``"")
    Write-Output "ODM AD Service Path: `$(`$ODMAgentPath)"
}
Else{
    Write-Output "No ODM Agent Service Found...Terminating"
    Exit 1
    }

`$TranscriptFile = "`$(`$ODMAgentPath)\Files\PowerShell-`$(Get-Date -f yyyyMMdd-HHMM)-
BackupBitlockerKeyToAAD.log"
Start-Transcript -Path `$TranscriptFile

`$DriveLetter = `$env:SystemDrive

#endregion declarations

#region functions

function Test-Bitlocker (`$BitlockerDrive) {
    #Tests the drive for existing Bitlocker keyprotectors
    try {
        Get-BitLockerVolume -MountPoint `$BitlockerDrive -ErrorAction Stop
    } catch {
        Write-Output "Bitlocker was not found protecting the `$BitlockerDrive drive.
Terminating script!"
        exit 0
    }
}

function Get-KeyProtectorId (`$BitlockerDrive) {
```

BackupBitlockerKeytoAAD.txt

```
    #fetches the key protector ID of the drive
    `$BitLockerVolume = Get-BitLockerVolume -MountPoint `$BitlockerDrive
    `$KeyProtector = `$BitLockerVolume.KeyProtector | Where-Object {
`$_.KeyProtectorType -eq 'RecoveryPassword' }
    return `$KeyProtector.KeyProtectorId
}

function Invoke-BitlockerEscrow (`$BitlockerDrive,`$BitlockerKey) {
    #Escrow the key into Azure AD
    try {
        BackupToAAD-BitLockerKeyProtector -MountPoint `$BitlockerDrive -KeyProtectorId
`$BitlockerKey -ErrorAction SilentlyContinue
        Write-Output "Attempted to escrow key in Azure AD - Please verify manually!"
        exit 0
    } catch {
        Write-Error "Error Occurred"
        exit 1
    }
}

#endregion functions

#region execute

Test-Bitlocker -BitlockerDrive `$DriveLetter
`$KeyProtectorId = Get-KeyProtectorId -BitlockerDrive `$DriveLetter
Invoke-BitlockerEscrow -BitlockerDrive `$DriveLetter -BitlockerKey `$KeyProtectorId

#endregion execute


Remove-Item -path "`$ODMAgentPath\$($ScriptName)" -Force

Unregister-ScheduledTask -TaskName "$($TaskName)" -Confirm:`$false

Stop-Transcript

"@

#$output = New-Object BinaryTree.ADM.Agent.PSHelpers.PSOutput

### Get ODMAD Agent Information to determine path
Try{
   $ODMADService = Get-Service -Name ODMActiveDirectory -ErrorAction SilentlyContinue
   }
Catch{
    Write-Output "Error Retrieving Service Status...Terminating with error: $($Error)"
    Exit 1
    }
If($ODMADService){
    Write-Output "ODM AD Agent Service Found...Finding ODM AD Agent Service Path"
    $ODMADServicePath = (Get-ItemProperty -Path
```

BackupBitlockerKeytoAAD.txt

```
HKLM:SYSTEM\CurrentControlSet\Services\ODMActiveDirectory).ImagePath
    $ODMAgentPath = Split-Path $ODMADServicePath
    $ODMAgentPath = $ODMAgentPath.Trim("`"")
    Write-Output "ODM AD Service Path: $($ODMAgentPath)"
}
Else{
    Write-Output "No ODM Agent Service Found...Terminating"
    Exit 1
    }

$AgentPath = "$ODMAgentPath\"
$ScriptFullName = $AgentPath+$ScriptName
If(!(Test-Path $ScriptFullName)) {
    New-item -path $ODMAgentPath -Name $ScriptName -Type "File" -Value $BacktoAAD
}

# Create Scheduled Task
$TaskName = "Backup Bitlocker Key"
$Argument = "-ExecutionPolicy Bypass -File `"$($ODMAgentPath)\$($ScriptName)`""
$Action = New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument $Argument
$Settings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries
$Principal = New-ScheduledTaskPrincipal -UserId "LOCALSERVICE" -LogonType
ServiceAccount
$Trigger = New-ScheduledTaskTrigger -Atlogon
$Trigger.Delay = "PT20M"
$ScheduledTask = New-ScheduledTask -Action $Action -Trigger $Trigger -Settings
$Settings
# Register Scheduled Task
Register-ScheduledTask -TaskName $TaskName -InputObject $ScheduledTask -User "NT
AUTHORITY\SYSTEM" -Force

return ($output)
```

SetPrimaryUser (Optional)

The Primary User value is not automatically set in the target Microsoft Entra ID when performing an Microsoft Entra ID join. If the machine does not have a primary user, the machine is considered a shared machine in Intune, which can affect how Intune policies and applications are delivered to the workstation post migration. Assigning a Primary User value can help indicate the one-to-one relationship between a user and workstation.

This script creates a separate PowerShell script on the workstation called UpdatePrimaryUser.ps1 in the ODMAD agent folder, creates encrypted files on the workstation that will be used by the script, and creates a Scheduled Task to execute UpdatePrimaryUser.ps1 when the first target user logs on.

Since Microsoft Graph must be used to update the Primary User value, an Enterprise Application is required with the following permissions. Detailed instructions are included later in this guide for creating this application.

- Device.Read.All
- DeviceManagementManagedDevices.ReadWrite.All

- User.Read.All

  Once the Enterprise Application is created in the tenant, the following three global Variables must be configured in ODMAD. Detailed instructions are included later in this guide for configuring these variables in ODMAD.

- ApplicationClientID –The Application Client ID of the Enterprise application
- ApplicationSecretID – The Application Secret ID of the Enterprise application
- TenantID – The Tenant ID of the source tenant

  The script will get the values of the Global Variables and store these values in encrypted files on the workstation in the ODMAD agent folder, which will be created as follows:

- Application Client ID = CAPPID.ODMAD
- Application Secret ID – ENAPP.ODMAD
- TenantID – TID.ODMAD

  When the UpdatePrimaryUser script runs during the first login post-migration, it will open the encrypted files above to get the Enterprise Application information, will get information about the workstation including the machine serial number, will query Microsoft Entra ID for the Intune Device ID, will determine which Microsoft Entra ID workstation entry to write the Primary User to, and will get the information about the Microsoft Entra ID user currently authenticated to the workstation (Object ID). The script will then set the currently logged-on user as the primary user of the workstation.

  The script will also create a log file in the ODM agent Files folder and then perform cleanup to delete the encrypted ODMAD files containing the Enterprise application information, remove the Scheduled Task, and remove the script itself.

SetPrimaryUserTask.txt

```
### Get Variable Values from ODMAD
Param (
    [System.String] $ApplicationClientID = $null,
    [System.String] $ApplicationSecretID = $null,
    [System.String] $TenantID = $null
   )

### Setting Other Variables
$TaskName = "Update Intune Primary User"
$ScriptName = "UpdatePrimaryUser.ps1"
$output = New-Object BinaryTree.ADM.Agent.PSHelpers.PSOutput

### File Names to Store Encrypted Enterprise applications Details
$ClientAppIDSecureFile = "cappid.odmad"
$SecretValueSecureFile ="enapp.odmad"
$TenantIDSecureFile = "tid.odmad"


### Get ODMAD Agent Information to determine path
Try{
   $ODMADService = Get-Service -Name ODMActiveDirectory -ErrorAction SilentlyContinue
   }
Catch{
```

SetPrimaryUserTask.txt

```
    Write-Output "Error Retrieving Service Status...Terminating with error: $($Error)"
    Exit 1
    }
If($ODMADService){
    Write-Output "ODM AD Agent Service Found...Finding ODM AD Agent Service Path"
    $ODMADServicePath = (Get-ItemProperty -Path
HKLM:SYSTEM\CurrentControlSet\Services\ODMActiveDirectory).ImagePath
    $ODMAgentPath = Split-Path $ODMADServicePath
    $ODMAgentPath = $ODMAgentPath.Trim("`"")
    Write-Output "ODM AD Service Path: $($ODMAgentPath)"
}
Else{
    Write-Output "No ODM Agent Service Found...Terminating"
    Exit 1
    }

###Secure Client App ID in Encrypted File
$SecClientAppID=ConvertTo-SecureString -AsPlainText -$ApplicationClientID -Force
$EncryptedClientAppID = ConvertFrom-SecureString $SecClientAppID
$EncryptedClientAppID | Out-File  "$($ODMAgentPath)\$($ClientAppIDSecureFile)"
$File = Get-Item "$($ODMAgentPath)\$($ClientAppIDSecureFile)"
$File.Attributes -="Hidden"

###Secure Entprise App Secret Value in Encrypted File
$SecAppValue=ConvertTo-SecureString -AsPlainText -$ApplicationSecretID -Force
$EncryptedEntAppIDValue = ConvertFrom-SecureString $SECAppValue
$EncryptedEntAppIDValue | Out-File  "$($ODMAgentPath)\$($SecretValueSecureFile)"
$File = Get-Item "$($ODMAgentPath)\$($SecretValueSecureFile)"
$File.Attributes -="Hidden"

###Secure TenantID in Encrypted File
$SecTenantValue=ConvertTo-SecureString -AsPlainText -$TenantID -Force
$EncryptedTenantValue = ConvertFrom-SecureString $SECTenantValue
$EncryptedTenantValue | Out-File  "$($ODMAgentPath)\$($TenantIDSecureFile)"
$File = Get-Item "$($ODMAgentPath)\$($TenantIDSecureFile)"
$File.Attributes -="Hidden"


#Create Script
$Script =@"
### Configure Encrypted File - This must align with the Script that creats the files
`$ODMADService = Get-Service -Name ODMActiveDirectory
If(`$ODMADService){
    Write-Output "ODM AD Agent Service Found...Finding ODM AD Agent Service Path"
    `$ODMADServicePath = (Get-ItemProperty -Path
HKLM:SYSTEM\CurrentControlSet\Services\ODMActiveDirectory).ImagePath
    `$ODMAgentPath = Split-Path `$ODMADServicePath
    `$ODMAgentPath = `$ODMAgentPath.Trim("``"")
    Write-Output "ODM AD Service Path: `$(`$ODMAgentPath)"
}

`$TranscriptFile = "`$(`$ODMAgentPath)\Files\PowerShell-`$(Get-Date -f yyyyMMdd)-
```

SetPrimaryUserTask.txt

```
UpdatePrimaryUser.log"
Start-Transcript -Path `$TranscriptFile

`$ClientAppIDSecureFile = "cappid.odmad"
`$SecretValueSecureFile ="enapp.odmad"
`$TenantIDSecureFile = "tid.odmad"

####Decrypt the Enterprise Application Access Files, Store values
If(Test-path -Path "`$ODMAgentPath\`$ClientAppIDSecureFile" -PathType Leaf){
    `$EncryptedClientAppID = Get-Content "`$ODMAgentPath\`$ClientAppIDSecureFile"
     Write-Output "DEBUG: Encrypted Application Client ID: `$(`$EncryptedClientAppID)"
    `$SecureClientAppID = ConvertTo-SecureString `$EncryptedClientAppID
     Write-Output "DEBUG: Secure Application Client ID: `$(`$SecureClientAppID)"
    `$DecryptedClientAppID = [System.Runtime.InteropServices.Marshal]::PtrToStringBSTR
([System.Runtime.InteropServices.Marshal]::SecureStringToBSTR(`$SecureClientAppID))
     Write-Output "DEBUG: Decrypted Application Client ID: `$(`$DecryptedClientAppID)"
    `$DecryptedClientAppID=`$DecryptedClientAppID.Trim("-")
}

If(Test-path -Path "`$ODMAgentPath\`$SecretValueSecureFile" -PathType Leaf){
    `$EncryptedEntAppID = Get-Content "`$ODMAgentPath\`$SecretValueSecureFile"
    `$SecureEntAppID = ConvertTo-SecureString `$EncryptedEntAppID
    `$DecryptedEntAppID = [System.Runtime.InteropServices.Marshal]::PtrToStringBSTR
([System.Runtime.InteropServices.Marshal]::SecureStringToBSTR(`$SecureEntAppID))
    `$DecryptedEntAppID=`$DecryptedEntAppID.Trim("-")
}

If(Test-path -Path "`$ODMAgentPath\`$TenantIDSecureFile" -PathType Leaf){
    `$EncryptedTenantID = Get-Content "`$ODMAgentPath\`$TenantIDSecureFile"
    `$SecureTenantID = ConvertTo-SecureString `$EncryptedTenantID
    `$DecryptedTenantID = [System.Runtime.InteropServices.Marshal]::PtrToStringBSTR
([System.Runtime.InteropServices.Marshal]::SecureStringToBSTR(`$SecureTenantID))
    `$DecryptedTenantID=`$DecryptedTenantID.Trim("-")
}

Write-Output "Authenticating to MS Graph"
`$URL = "https://login.microsoftonline.com/`$DecryptedTenantID/Oauth2/token"
`$Resource = "https://graph.microsoft.com/"
`$BaseURL = "https://graph.microsoft.com/v1.0"
`$RestBody =@{
    grant_type = 'client_credentials'
    client_id  = `$DecryptedClientAppID
    client_secret = `$DecryptedEntAppID
    resource = `$Resource
}
`$token = Invoke-RestMethod -Method POST -Uri `$URL -Body `$RestBody
`$Header = @{
    'Authorization' = "`$(`$token.token_type) `$(`$token.access_token)"
}

### Get Machine ID ###
`$SerialNumber = Get-WMIObject -Class Win32_Bios | Select-Object -ExpandProperty
serialNumber
```

SetPrimaryUserTask.txt

```
Write-Output "Serial Number is `$(`$SerialNumber)"
`$SerialNumber = `$SerialNumber -replace(' ','')
`$IntuneDevice = Invoke-RestMethod -Method Get -Headers `$Header -uri
"`$(`$BaseURL)/deviceManagement/managedDevices?```$filter=contains
(serialNumber,'`$(`$SerialNumber)')"
`$IntuneDeviceID = `$IntuneDevice.Value.ID
Write-Output "Intune Device ID is `$(`$IntuneDeviceID)"

### Get Logged On User from Local Machine ###
#Try to get the DisplayName for Domain User
`$ErrorActionPreference = "SilentlyContinue"

Try {
            Write-Output "Trying Identity LogonUI Registry Key for Domain User info..."
            `$User = Get-Itemproperty -Path
"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI" -Name
"LastLoggedOnDisplayName" | Select-Object -ExpandProperty LastLoggedOnDisplayName

            If (`$Null -eq `$User) {
                Write-Output "UserName not found at LogonUI"
                `$UserName = `$Null
            }
            else {
                Write-Output "UserName found at LogonUI"
                `$DisplayName = `$User.Split(" ")
                `$UserName = `$DisplayName[0]
            }
    }
        Catch [System.Management.Automation.PSArgumentException] {
            "Registry Key Property missing"
            Write-Warning "Registry Key for LastLoggedOnDisplayName could not be
found."
            `$UserName = `$Null
        }
        Catch [System.Management.Automation.ItemNotFoundException] {
            "Registry Key itself is missing"
            Write-Warning "Registry value for LastLoggedOnDisplayName could not be
found."
            `$UserName = `$Null
        }

    #Try to get the DisplayName for Azure AD User from Session 2 Reg Key
        If (`$Null -eq `$UserName) {
            Write-Output "Trying Identity LogonUI Session 2 Registry Key for Azure AD
User info..."
            Try {
                `$User = Get-Itemproperty -Path
"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\SessionData\2"
-Name "LoggedOnDisplayName" | Select-Object -ExpandProperty LoggedOnDisplayName

            If (`$Null -eq `$User) {
                `$UserName = `$Null
```

SetPrimaryUserTask.txt

```
            }
            else {
                Write-Output "UserName found at LogonUI Session 2"
                `$DisplayName = `$User.Split(" ")
                `$UserName = `$DisplayName[0]
            }
    }
        Catch [System.Management.Automation.PSArgumentException] {
            "Registry Key Property missing"
            Write-Warning "Registry Key for LastLoggedOnDisplayName could not be
found."
            `$UserName = `$Null
        }
        Catch [System.Management.Automation.ItemNotFoundException] {
            "Registry Key itself is missing"
            Write-Warning "Registry value for LastLoggedOnDisplayName could not be
found."
            `$UserName = `$Null
        }
        }
    #Try to get the DisplayName from current logged on User
        If (`$Null -eq `$UserName)
        {
            Write-Output "Trying Identity Store Cache from Locally logged on User..."

            Try {
        `$CurrentLoggedOnUser = tasklist /v /FI "IMAGENAME eq explorer.exe" /FO list |
find "User Name:"
                `$CurrentLoggedOnUser = `$CurrentLoggedOnUser.Substring(14)
                Write-Output "  DEBUG: Current Logged On User:
`$(`$CurrentLoggedOnUser)"
                `$LoggedonUserArray = `$CurrentLoggedOnUser.Split("\")
                `$LoggedonUser_Username = `$LoggedonUserArray[1]
                Write-Output "  DEBUG: Current Logged On Username: `$(`$LoggedonUser_
Username)"
                `$LoggedonUser_Domain = `$LoggedonUserArray[0]
                Write-Output "  DEBUG: Current Logged On Domain: `$(`$LoggedonUser_
Domain)"

            If (!([String]::IsNullOrWhiteSpace(`$CurrentLoggedOnUser)))
                    {
                    `$CurrentUser = Get-Itemproperty "Registry::\HKEY_USERS\S-1-
12*\Volatile Environment"|Where-Object {`$_.USERDOMAIN -match `$LoggedonUser_Domain}
                    Write-Output "  DEBUG: CurrentUser: `$(`$CurrentUser)"
                    }
            If (![String]::IsNullOrWhiteSpace(`$CurrentUser))
                    {
                    `$CurrentLoggedOnUser =
"`$(`$CurrentUser.USERDOMAIN)\`$(`$CurrentUser.USERNAME)"
                    Write-Output "  DEBUG: Current Logged On User:
`$(`$CurrentLoggedOnUser)"
                    `$CurrentLoggedOnUserSID = split-path `$CurrentUser.PSParentPath -
leaf
```

SetPrimaryUserTask.txt

```
                    Write-Output "  DEBUG: Current Logged On User SID:
`$(`$CurrentLoggedOnUserSID)"
                    `$LogonCacheSID = (Get-ChildItem
HKLM:\SOFTWARE\Microsoft\IdentityStore\LogonCache -Recurse -Depth 2 | Where-Object {
`$_.Name -match `$CurrentLoggedOnUserSID }).Name
                    }

            If (`$LogonCacheSID)
                    {
                    `$LogonCacheSID = `$LogonCacheSID.Replace("HKEY_LOCAL_MACHINE",
"HKLM:")
                    `$User = Get-ItemProperty -Path `$LogonCacheSID | Select-Object -
ExpandProperty DisplayName -ErrorAction Stop
                    `$DisplayName = `$User.Split(" ")
                    `$Username = `$DisplayName[0]
                    }
                else {
                    Write-Warning "Could not get DisplayName property from Identity
Store Cache for Azure AD User"
                    `$UserName = `$Null
                    }
                }
            Catch [System.Management.Automation.PSArgumentException]
                    {
                    Write-Warning "Could not get DisplayName property from Identity
Store Cache for Azure AD User"
                    `$UserName = `$Null
                    }
            Catch [System.Management.Automation.ItemNotFoundException]
                    {
                    Write-Warning "Could not get SID from Identity Store Cache for
Azure AD User"
                    `$UserName = `$Null
                    }
            Catch     {
                    Write-Warning "Could not get SID from Identity Store Cache for
Azure AD User"
                    `$UserName = `$Null
                    }
        }

        #If DisplayName could not be obtained, perform cleanup and Stop script
        If (`$Null -eq `$UserName) {
            Clear-Variable DecryptedClientAppID
            Clear-Variable DecryptedEntAppID
            Clear-Variable DecryptedTenantID
            Remove-Item -Path "`$ODMAgentPath\`$ClientAppIDSecureFile" -Force
            Remove-Item -Path "`$ODMAgentPath\`$SecretValueSecureFile" -Force
            Remove-Item -Path "`$ODMAgentPath\`$TenantIDSecureFile" -Force
            Remove-Item -path "`$ODMAgentPath\UpdatePrimaryUser.ps1" -Force
            Unregister-ScheduledTask -TaskName "Update Intune Primary User" -
Confirm:`$false
            Write-Error "DisplayName could not be obtained, Script Failed - Unable to
```

SetPrimaryUserTask.txt

```
update User" -ErrorAction Stop
            Stop-Transcript
        }


Write-Output "Current User DisplayName is `$(`$Username)"

### Get EntraID User Object ID ###
`$URL = `$BaseURL +'/users/' + `$UserName
`$UserInfo = Invoke-RestMethod -Method GET -Headers `$Header -Uri
"`$(`$BaseURL)/users?```$filter=displayname eq '`$(`$Username)'"
`$UserID = `$UserInfo.Value.id
Write-Output "Entra ID user object ID for `$(`$username) is `$(`$UserID)"

#### Write Primary User to Intune Device ###
Write-Output "Setting `$(`$UserName) as Primary User on Device ID:
`$(`$IntuneDeviceID)"
`$DeviceUsersUri = "`$(`$BaseURL)/deviceManagement/managedDevices
('`$IntuneDeviceID')/users/```$ref"
Write-Output "DEBUG: DeviceUsersUri: `$(`$DeviceUsersUri)"
`$UserUri = "`$(`$BaseURL)/users/" + `$UserID
Write-Output "DEBUG: USERURI: `$(`$Useruri)"
`$id = "@odata.id"
`$JSON = @{ `$id="`$userUri" } | ConvertTo-Json -Compress
Write-Output "DEBUG: JSON: `$(`$JSON)"
Invoke-RestMethod -Method Post -Headers `$Header -Body `$JSON -Uri `$DeviceUsersUri -
ContentType "application/Json"


### Clean Up
Clear-Variable DecryptedClientAppID
Clear-Variable DecryptedEntAppID
Clear-Variable DecryptedTenantID

Remove-Item -Path "`$ODMAgentPath\`$ClientAppIDSecureFile" -Force
Remove-Item -Path "`$ODMAgentPath\`$SecretValueSecureFile" -Force
Remove-Item -Path "`$ODMAgentPath\`$TenantIDSecureFile" -Force
Remove-Item -path "`$ODMAgentPath\UpdatePrimaryUser.ps1" -Force

Unregister-ScheduledTask -TaskName "Update Intune Primary User" -Confirm:`$false

Stop-Transcript

"@

$ScriptFullName = "$($ODMAgentPath)\$($ScriptName)"
If(!(Test-Path $ScriptFullName)) {
    New-item -path $ODMAgentPath -Name $ScriptName -Type "File" -Value $Script
}

# Create Scheduled Task
$Argument = "-ExecutionPolicy Bypass -File `"$($ODMAgentPath)\$($ScriptName)`""
$Action = New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument $Argument
```

SetPrimaryUserTask.txt

```
$Settings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries
$Trigger = New-ScheduledTaskTrigger -Atlogon
$ScheduledTask = New-ScheduledTask -Action $Action -Trigger $Trigger -Settings
$Settings
# Register Scheduled Task
Register-ScheduledTask -TaskName $TaskName -InputObject $ScheduledTask -User "NT
AUTHORITY\SYSTEM" -Force

Return($output)
```

CleanUpLocalAdministratorsGroup (Optional)

If the ReACL profile is configured to process local users & groups, the ReACL process will add the target user's Microsoft Entra ID account to the local Administrators group if the source user is a member of that group. If this is not allowed by target security policies, then the target user accounts should be removed from the local Administrators group before migration, as local groups can be managed in the Target Intune environment post-migration.

This script will check the Local Administrators group (identified by SID in case the group has been renamed) and will remove any users where the domain portion of their username matches "Microsoft Entra ID"

CleanUp Local Administrators Group.txt

```
Param (
)

$output = New-Object BinaryTree.ADM.Agent.PSHelpers.PSOutput

$CleanUnresolvedSIDS = $false
If($CleanUnresolvedSIDS -eq $true){Write-Output "Clean up of unresolved SIDs is
Enabled"}
Else{Write-Output "Clean up of unresolved SIDs is Disabled"}


### Get Local Administrators Group
$Get_Local_AdminGroup = Get-WmiObject win32_group -Filter "Domain='$env:computername'
and SID='S-1-5-32-544'"
$Get_Local_AdminGroup_Name = $Get_Local_AdminGroup.Name
Write-Output "Administrators group name is: $($Get_Local_AdminGroup_Name)"

## Get Local Administrators group owners
$group = [ADSI]"WinNT://$env:COMPUTERNAME/$($Get_Local_AdminGroup_Name)"
    $admins = $group.Invoke('Members') | % {
        $path = ([adsi]$_).path
        [pscustomobject]@{
            Computer = $env:COMPUTERNAME
            Domain = $(Split-Path (Split-Path $path) -Leaf)
            User = $(Split-Path $path -Leaf)
        }
    }
```

CleanUp Local Administrators Group.txt

```
### Filter for AzureAD Acounts only - Ignore all other accounts

foreach($admin in $admins){
   If($admin.Domain -eq "AzureAD"){
     Write-Output "Removing AzureAD Users from Local Administrators Group"
     Write-Output "  Removing AzureAD User: $($admin.User)"
     Try{
         Remove-LocalGroupMember -Group $Get_Local_AdminGroup_Name -Member
"$($admin.domain)\$($admin.user)"
         }
     Catch{
          Write-Output "Error occured removing $($admin.user) from $($Get_Local_
AdminGroup_Name) group"
          }
   }

}

### OPTIONAL: Clean up unresolved SIDs - Controled by status of the
$CleanUnresvoldeSIDS Variable ($True=Enabled, $False=Disabled)

If($CleanUnresolvedSIDS -eq $True){
    Write-Output "Removing unresolved SIDs from Group"
    foreach($admin in $admins){
        $admin
        #### Check if SID starts with S-1-12-1 (AzureAD objects) -If Yes then ignore
        If($admin.user.StartsWith('S-1-12-1')){
            Write-Output "AzureAD User Found - Ignoring unresolved SID"
            Continue
        }
        ElseIf($admin.Domain -eq "WinNT:\"){
            Write-Output "  Removing unresolved SID: $($admin.User) from $($Get_Local_
AdminGroup_Name)"
            Try{
                Remove-LocalGroupMember -Group $Get_Local_AdminGroup_Name -Member
$admin.user
                }
            Catch{
                 Write-Output "Error occured removing $($admin.user) from $($Get_Local_
AdminGroup_Name) group"
                 }
        }
   }
}


return ($output)
```

Implementation Process

1. Copy the Default EntraIDCutover Action

1. In ODMAD using Select CONFIGURATIONS from the main ODMAD Menu.
2. Select ACTIONS.
3. In the ACTIONS section select click SHOW SYSTEM.
4. Find the EntraIDCutoverAction and select it.
5. Click COPY, which will open the Edit a Custom Action dialog window. Configure the action as follows:
    a. ACTION NAME: IntuneMicrosoftEntraIDCutover
    b. ACTION DISPLAY NAME: Intune Microsoft Entra ID Cutover
    c. DESCRIPTION: Process to join an Intune/Autopilot workstation to an Microsoft Entra ID
    d. ACTION TARGET: Computer
    e. ACTION TYPE: Microsoft Entra ID Cutover
6. Click the SAVE button to continue.

2. Add Autopilot Clean Up Task

1. Scroll down to the TASKS section of the Action window and click NEW.
2. The ADD A Custom Task window will appear. Configure this as follows:
    a. TASK NAME: AutopilotCleanUp
    b. DESCRIPTION: Removes Autopilot Registry Keys from Workstations, to prepare for the integration into Target Tenant Intune/Autopilot configuration.
    c. TASK TYPE: PowerShell Script
3. Click NEXT to Continue.
4. Copy the Autopilot Cleanup Script into the SCRIPT Section.
   Note: There is no need to click the LOAD SCRIPT FRAMWORK button as this is included in the PS1 file.
   Autopilot Cleanup.txt
5. Leave all other settings as default and click the SAVE button.
6. Select the Task just created and select the IntuneMicrosoftEntraIDCutover Action that was created earlier. Click the ADD TO button to add this task to the action.

7. Scroll up the ACTIONS section and expand the IntuneMicrosoftEntraIDCutover Action. The task just added will appear as the last step of the action, click+hold on the task and drag to correct position in the script (after the BT-ReACLPrepareWin10Profiles task). The change will be saved automatically.

3. Add Intune Clean Up Task

1. Scroll down to the TASKS section of the Action window and click NEW.
2. The ADD A Custom Task window will appear. Configure this as follows:
    a. TASK NAME: IntuneCleanUp
    b. DESCRIPTION: Removes the Registry keys associated with source tenant Intune enrollment, this script must be run to allow the workstation to join the target tenants Intune environment.
    c. TASK TYPE: PowerShell Script
3. Click NEXT to Continue.
4. Copy the Intune Cleanup Script into the SCRIPT Section.
   Note: There is no need to click the LOAD SCRIPT FRAMWORK button as this is included in the PS1 file.
   Intune Cleanup.txt
5. Leave all other settings as default and click the SAVE button.
6. Select the Task just created and select the IntuneMicrosoftEntraIDCutover Action that was created earlier. Click the ADD TO button to add this task to the action.
7. Scroll up the ACTIONS section and expand the IntuneMicrosoftEntraIDCutover Action. The task just added will appear as the last step of the action, click+hold on the task and drag to correct position in the script (after the AutopilotCleanUp task). The change will be saved automatically.

4. Add SetUserEmailValues Task

1. Scroll down to the TASKS section of the Action window and click NEW.
2. The ADD A Custom Task window will appear. Configure this as follows:
    a. TASK NAME: SetUserEmailValues
    b. DESCRIPTION: Creates Scheduled Task and Script to execute on user logon to set the UserEmail value in the CloudDomainJoin Registry value
    c. TASK TYPE: PowerShell Script
3. Click NEXT to continue.
4. Copy the SetUserEmailValues Script into the SCRIPT Section.
   Note: There is no need to click the LOAD SCRIPT FRAMWORK button as this is included in the PS1 file.
   SetUserEmailValues.txt

5. Leave all other settings as default and click the SAVE button.
6. Select the Task just created and select the IntuneMicrosoftEntraIDCutover Action that was created earlier. Click the ADD TO button to add this task to the action.
7. Scroll up the ACTIONS section and expand the IntuneMicrosoftEntraIDCutover Action. The task just added will appear as the last step of the action, click+hold on the task and drag to correct position in the script (after the IntuneCleanUp task). The change will be saved automatically.

5. Add BitlockerBackupToEntraID Task (Optional: Only required if source workstations are Bitlockered)

1. Scroll down to the TASKS section of the Action window and click NEW.
2. The ADD A Custom Task window will appear. Configure this as follows:
    a. TASK NAME: BitlockerBackupToEntraID
    b. DESCRIPTION: Backups the Bitlocker key from the Workstation to Entra ID user that logged on to the workstation
    c. TASK TYPE: PowerShell Script
3. Click NEXT to Continue.
4. Copy the BackupBitlockerToAAD Script into the SCRIPT Section.
   Note: There is no need to click the LOAD SCRIPT FRAMWORK button as this is included in the PS1 file.
   BackupBitlockerKeytoAAD.txt
5. Leave all other settings as default and click the SAVE button.
6. Select the Task just created and select the IntuneMicrosoftEntraIDCutover Action that was created earlier. Click the ADD TO button to add this task to the action.
7. Scroll up the ACTIONS section and expand the IntuneMicrosoftEntraIDCutover Action. The task just added will appear as the last step of the action, click+hold on the task and drag to correct position in the script (after the SetUserEmailValues task, but before the BT-EntraIDCutover task). The change will be saved automatically.

6. Add SetPrimaryUser Task (Optional)

The SetPrimaryUser is configured in three separate steps that should be completed in order:

1. Configure the Enterprise Application in the target Tenant (Required Elevated Permissions).
2. Configure Global Variables in On Demand.
3. Configure the SetPrimaryUser Task.

6a. Configure Enterprise Application

1. Logon to the Target tenant with an Administrator account that has the permissions to create and configure Enterprise applications.
2. Navigate into the Identity Admin Center (https://entra.microsoft.com)
3. Navigate to Applications à App Registrations.
4. Configure the App Registration with the following settings:
    a. Name: Migration Intune Access
    b. Scoped account Type: Accounts in this Organizational Directory Only
5. Click Register at the bottom the screen.
6. From the Application pane, note the Application (Client) ID:

    - Application Client ID: _____

7. Select API Permissions. Click Add a Permission; then click Microsoft Graph.
8. Select Application Permissions and add the following permissions:

- Device.Read.All
- DeviceManagementManagedDevices.ReadWrite.All
- User.Read.All

9. Click Add Permissions.
10. Click Grant admin consent for <Tenant Name>; click Yes to confirm.
11. Click on Certificates and Secrets. Select Client Secrets and click New Client Secret
    a. Enter a Description
    b. Configure an Expires time
12. Copy and store the Secret ID and the Secret ID Value:

- Secret ID: _____

- Secret ID Value: _____

6b. Configure On Demand Variables

1. Logon to On Demand Migration for Active Directory.
2. Select CONFIGURATIONS from the menu and click VARIABLES.
3. Click ADD and create three Variables with the flowing configuration, clicking SAVE after configurating each Variable.
    a. Client Application ID:
        a. Variable Name: ApplicationClientID
        b. Variable Value: <Copy in the Application Client ID from above>
        c. Encrypted: Yes (Tick box Selected)
    b. Secret ID Value:
        a. Variable Name: ApplicationSecretID
        b. ApplicationSecretID
        c. Variable Value: <Copy in the Secret ID Value from above>
        d. Encrypted: Yes (Tick box Selected)
    c. Tenant ID:
        a. Variable Name: TenantID
        b. Variable Value: <Copy in the Tenant ID of the target Tenant>
        c. Encrypted: No (Tick box unselected)
        Note: When a Variable is Encrypted, there is no way to retrieve this value in On Demand for Active Directory.

6c. Configure SetPrimaryUser Task

1. Scroll down to the TASKS section of the Action window and click NEW.
2. The ADD A Custom Task window will appear. Configure this as follows:
    a. TASK NAME: SetPrimaryUser
    b. DESCRIPTION: Creates the Script and the Scheduled Task that will be executed to set the primary user. This task will download the Enterprise App Configuration to the workstation.
    c. TASK TYPE: PowerShell Script
3. Click NEXT to Continue.
4. Copy the SetPrimaryUser Script into the SCRIPT Section.
   Note: There is no need to click the LOAD SCRIPT FRAMWORK button as this is included in the PS1 file.
   SetPrimaryUserTask.txt
5. Select the Global Variables check-box in the "Include Variables" section.

6. Leave all other settings as default and click the SAVE button.
7. Select the Task just created and select the IntuneMicrosoftEntraIDCutover Action that was created earlier. Click the ADD TO button to add this task to the action.
8. Scroll up the ACTIONS section and expand the IntuneMicrosoftEntraIDCutover Action. The task just added will appear as the last step of the action, click+hold on the task and drag to correct position in the script (after the SetUserEmailValues task, but before the BT-EntraIDCutover task). The change will be saved automatically.

7. Add CleanupLocalAdministratorsGroup Task (Optional)

1. Scroll down to the TASKS section of the Action window and click NEW.
2. The ADD A Custom Task window will appear. Configure this as follows:
    a. TASK NAME: CleanupLocalAdministratorsGroup
    b. DESCRIPTION: Removes Microsoft Entra ID Domain users from the local Administrators group before cutover.
    c. TASK TYPE: PowerShell Script
3. Click NEXT to Continue.
4. Copy the CleanupLocalAdministratorsGroup Script into the SCRIPT Section.
   Note: There is no need to click the LOAD SCRIPT FRAMWORK button as this is included in the PS1 file.
   CleanUp Local Administrators Group.txt
5. Leave all other settings as default and click the SAVE button.
6. Select the Task just created and select the IntuneMicrosoftEntraIDCutover Action that was created earlier. Click the ADD TO button to add this task to the action.
7. Scroll up the ACTIONS section and expand the IntuneMicrosoftEntraIDCutover Action. The task just added will appear as the last step of the action, click+hold on the task and drag to correct position in the script (after the SetUserEmailValues task, but before the BT-EntraIDCutover task). The change will be saved automatically.

Intune Cutover Run Book

This runbook assumes that the computer had been read in to On Demand and the workstation has the agent installed, configured, and registered.

1. Run Re-ACL Process

1. In On Demand, navigate to Devices and Servers.
2. Select the Device and from the drop-down menu select Re-ACL.
3. Select the Re-ACL profile and follow the on-screen prompts.

2. Run Cutover Process

2a. Remove Workstation from Source Autopilot

The is process must be completed manually and will remove the serial number from the source tenant. If the device is not removed from the source Autopilot, any attempt to run Autopilot on the device in the future will result in it being auto-deployed to the source tenant again.

1. Logon to the M365 Tenant and navigate to Endpoint Management (https://endpoint.microsoft.com)
2. Navigate to Devices à By Platform | Windows
    a. Find the workstation(s) that are to be removed.
    b. Select the workstation and note the device Serial Number
3. Navigate to Devices à Device Enrollment | Enroll Devices
4. Under Autopilot Deployment Program, click on Devices then select:
    a. Find the Device Serial Number and select it.
    b. Click the Delete button to remove the device from Autopilot. The Microsoft Entra ID and the Intune device will remain intact.

2b. Cutover the Device using ODMAD

1. In On Demand, navigate to Devices and Servers.
2. Select the Device(s) to be cutover and from the drop-down menu select Intune Microsoft Entra ID Cutover.
3. Select the Microsoft Entra ID Cutover Profile and follow the on-screen prompts.

About us

Quest creates software solutions that make the benefits of new technology real in an increasingly complex IT landscape. From database and systems management, to Active Directory and Office 365 management, and cyber security resilience, Quest helps customers solve their next IT challenge now. Around the globe, more than 130,000 companies and 95% of the Fortune 500 count on Quest to deliver proactive management and monitoring for the next enterprise initiative, find the next solution for complex Microsoft challenges and stay ahead of the next threat. Quest Software. Where next meets now. For more information, visit www.quest.com.

Technical support resources

Technical support is available to Quest customers with a valid maintenance contract and customers who have trial versions. You can access the Quest Support Portal at https://support.quest.com.

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. The Support Portal enables you to:

- Submit and manage a Service Request
- View Knowledge Base articles
- Sign up for product notifications
- Download software and technical documentation
- View how-to-videos
- Engage in community discussions
- Chat with support engineers online
- View services to assist you with your product