Microsoft Intune

- Experiences with Multi Admin Approval
- Export Intune device script content via Graph API
- Import enrollment devices via Graph API
- Create Win32 line of business app with file upload via Graph API
- Set primary user of Windows devices by last logged in users with automation
- Set scope tag by domain of primary user with automation
- Start Intune Device Sync via Graph API
- Interactively enroll Windows Device with Autopilot
- Evaluate installed app version from devices via Graph API
- Get duplicate Intune devices by serialnumber
- Download win32 intunewin content file via Graph API
- Windows Update during OOBE using Intune App
- Restrict app installation only during OOBE

Experiences with Multi Admin Approval

Multi Admin Approval is a feature in Intune, that require a second administrative account to approve a change before the change is applied.

With Multi Admin Approval (MAA), you configure access policies that protect specific configurations, like Apps or Scripts for devices. Access policies specify what is protected and which group of accounts are permitted to approve changes to those resources.

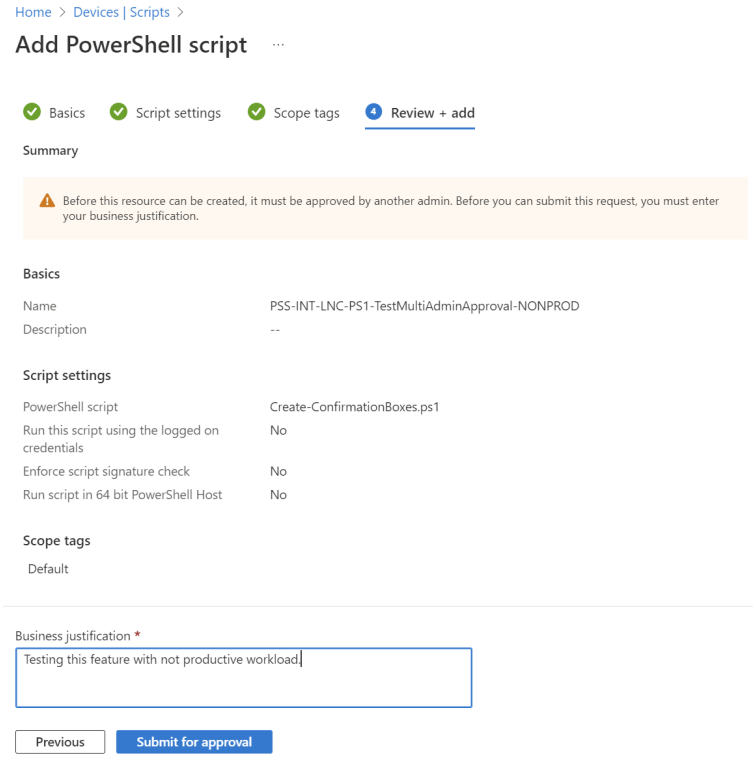
When any account in the Tenant is used to make a change to a resource that’s protected by an access policy, Intune won't apply the change until a different account explicitly approves it. Only administrators who are members of an approval group that’s assigned a protected resource in an access protection policy can approve changes. Approvers can also reject change requests.

Field report

- This feature is currently only applicable for Intune apps and Windows / MacOs scripts.
- To create or approve an approval request the account needs the role Intune Administrator even when in the account is in the approver group.
- The appropriately protected Intune resources (apps, scripts) cannot be restricted individually but are tenant wide protected for everyone via Multi Admin Approval.
- At the time of writing every request with scripts is only valid for one hour and then the status changes to expired. This does not apply to the Intune applications.
- Following entity actions need a separate approval request, whenever one of the actions is performed:.
  - Edit
  - Create
  - Modify
  - Delete
  - Assign

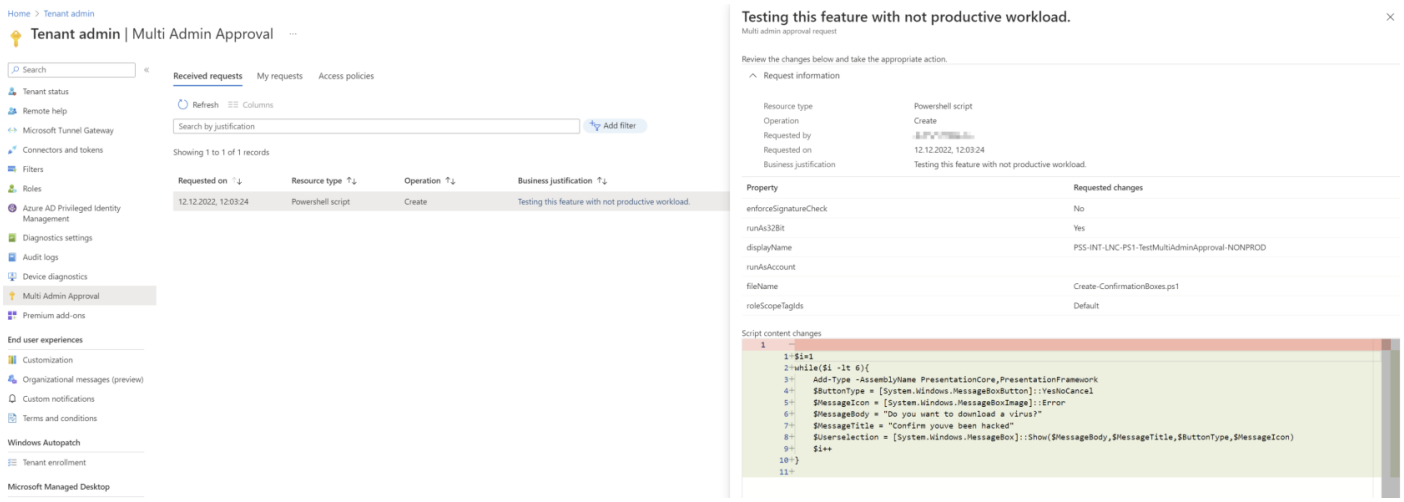
Steps of approval requests

After doing a described action (create, modify, delete, etc.) on an Intune resource which is protected by an access policy, will create an approval request in the Intune Admin Center. To submit the change you can use the normal Intune Admin Center.



Needs approval

After that submission a new approval request is created in the Intune Admin Center which needs to be approved or rejected from an other administrator account.



This approval is then moved to approved or rejected status according to the selection made.

Rejected

Rejected means that no further actions have to be made. The entity is archived and the status is set to rejected.

Approved

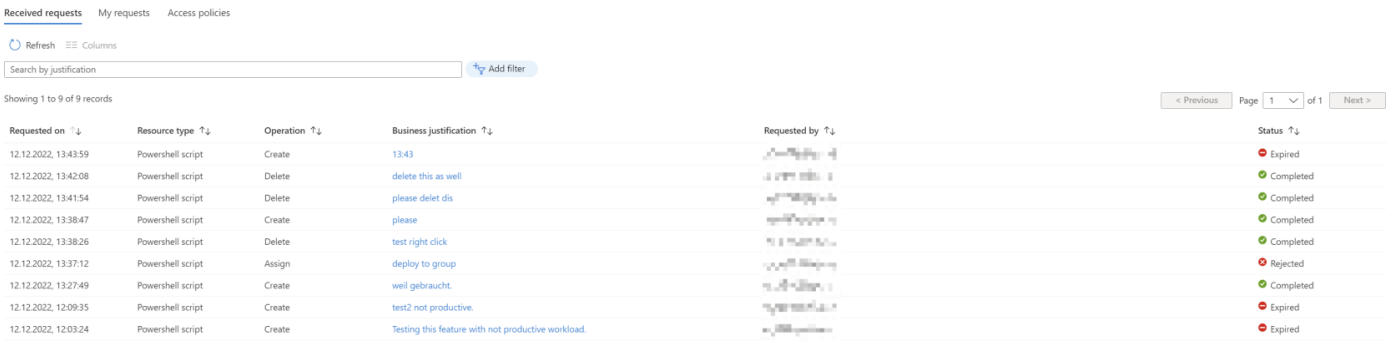
When approved by a different administrator your entity is then forwarded back to you so you can deploy the change at a time when it suits the creating person. The apps are implemented directly, without this following steps. This was tested with the scripts.

When "Complete request" is pressed by the owner of the approval request, the deployment of the change starts and gets implemented accordingly. The request then changes to the state of "Completed".

Completed

Completed are all requests which where approved by a different administrator and deployed by the owner. These changes were effectively made to the environment.

The Multi Administrator Approval is also very practical to trace changes.

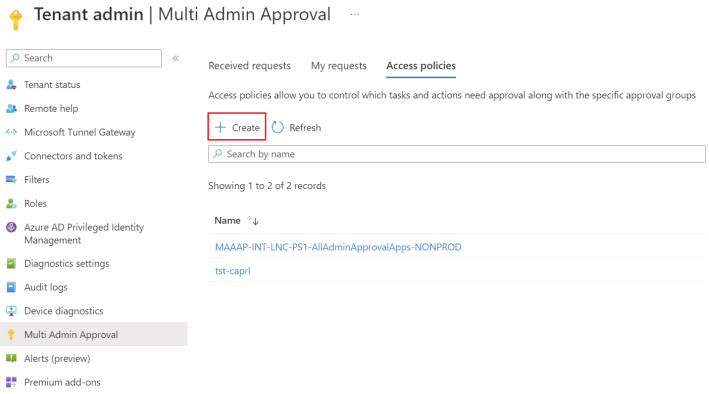


Expired

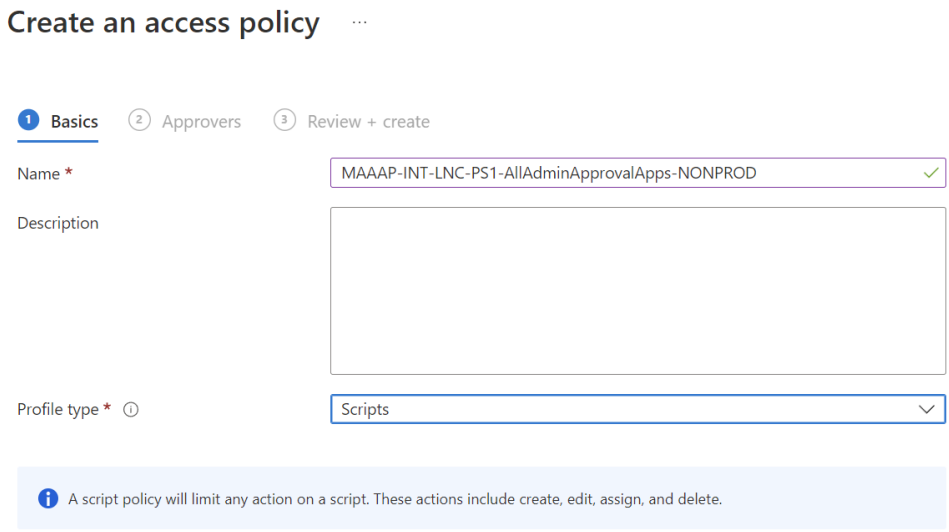
All requests which are not applied in one hour will get the status "Expired".

Create access policy

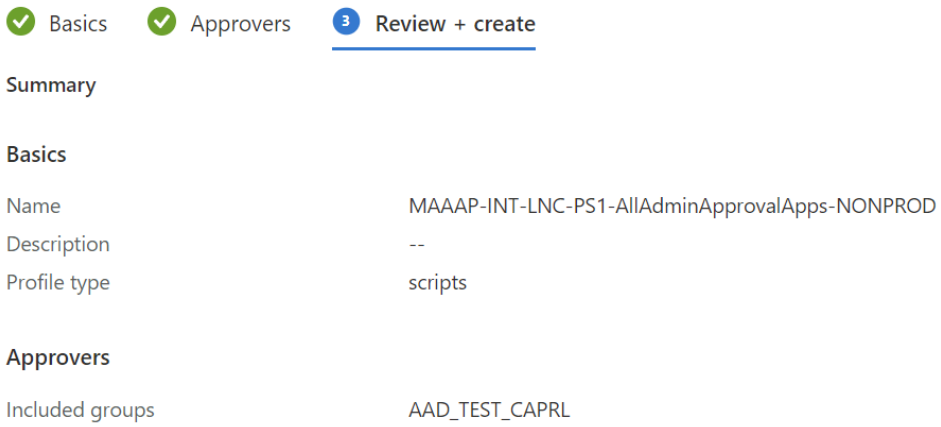
To create an access policy, you can change to "Multi Admin Approval" under "Tenant administration". There under "Access policies" you can create a new policy.



First you have to name the policy and choose the Profile type. Currently there are only two options; Scripts and Apps to select.



In addition, the approver group must be selected there. This group must contain the accounts which are authorized to approve or reject approval requests. These accounts must have to activate the "Intune Administrator" role.



Export Intune device script content via Graph API

> Prerequisites: Graph API access token and the script id from Intune device script.

Since the content of the device scripts cannot be read in the Intune Admin Center (as is possible with remediation scripts), the Graph API must be used for this.

Use case

Sometimes you want to see what device scripts have for content. Often such device scripts are only looked at again after several months or years and the documentation may no longer match. Then the solution is to export the content of the script so that the functions can be reverse engineered.

PowerShell script

This PowerShell script exports the contents of an Intune device script by id.

```
$FolderPath = "C:\Users\$($env:username)\Downloads\"
$ScriptId = "<yourdevicescriptid>"
$Token = "<yourgraphapitoken>"

$Header = @{
   "Authorization" = "Bearer $Token"
}

$script = Invoke-Restmethod -uri
"https://graph.microsoft.com/beta/deviceManagement/deviceManagementScripts" -Method GET -Header
$Header
```

```
[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String($($script.scriptContent))) | Out-File -Encoding ASCII -FilePath $(Join-Path $FolderPath $($script.fileName))
```

Import enrollment devices via Graph API

> Prerequisites: Graph API authorization header and the serial number & hardware identifier from the device. To read out the information such as serial number and hardware identifier from the device you can use the Get-WindowsAutopilotInfo tool provided by Microsoft.

To import devices directly via Graph API you can send a body to an API endpoint that contains the serial number and the hardware identifier.

Use case

To make an automation which imports devices automatically into Intune enrollment devices you can create an Azure Function that will then import the device id automatically to Intune without any user interaction or permission management. This action will be performed in the context of an app registration.

PowerShell script

This PowerShell script needs the GroupTag which should be set on the device. This value must be provided in clear text. In addition, the serial number must also be passed to the API as a string and the hardware ID as a binary.

```
$GroupTag = "<yourgrouptag>"
$SerialNumber = "<yourserialnumber>"
$HardwareIdentifier = "<yourcomputershashid>"

$Body = @"
{
  "groupTag": "$GroupTag",
  "serialNumber": "$SerialNumber",
  "hardwareIdentifier": "$HardwareIdentifier",
}
"@
```

```
$Response = Invoke-Restmethod -uri
"https://graph.microsoft.com/v1.0/deviceManagement/importedWindowsAutopilotDeviceIdentities" -Method
POST -Header $Header -Body $Body
```

Response: 201 Created

Corresponding Microsoft documentation

This documentation contains further information and optional values that can be transmitted to the following address so that the object can be enriched with more information.

Create importedWindowsAutopilotDeviceIdentity - Microsoft Graph v1.0 | Microsoft Learn

Create Win32 line of business app with file upload via Graph API

Here is a perfect manual:

Win32LOB intunewin file upload process explained for automation (rozemuller.com)

Set primary user of Windows devices by last logged in users with automation

This tutorial describes how an automation can be used to set the primary user according to the last signed in user. The Intune data is queried via an App Registration on Graph and modified accordingly. This automation is based on an Azure runbook and executes PowerShell code.

Prerequisites

First, an App Registration is used, which is used as an unattended authentication to the Graph API. This app registration requires "User.Read.All" and "DeviceManagementManagedDevies.ReadWrite.All". Create there the corresponding Client Secret and Client id as described here: Get app details and gr... | LNC DOCS (lucanoahcaprez.ch) Fill in the variables $tenantid, $clientid and $clientsecret with the corresponding values.

Subsequently, the $WebhookURI variable can be populated with a webhook from a team channel. Creating a webhook for notification in Microsoft Teams is described here: Teams webhook notification | LNC DOCS (lucanoahcaprez.ch)

The last function used is a LogCollection Function. You should copy the URL into the $FunctionURL variable. The instructions for a central log collection point can be found here: Centralize log collect... | LNC DOCS (lucanoahcaprez.ch)

PowerShell script

This is the PowerShell script that makes the automation possible. It is very important that you fill the variables correctly as described above or the unused parts are hidden. As written, this code is optimized to run in an Azure Runbook and also uses the variables from the Azure Automation account. Accordingly, these must be filled correctly.

```
#region Autorization Function
function Get-ApplicationOnlySourceAuthorization {

    param(
        $tenantId,
        $clientid,
        $clientSecret
    )



    $tokenBodySource = @{
        grant_type = "client_credentials"
        scope = "https://graph.microsoft.com/.default"
        client_id  = $clientid
        client_secret = "$clientSecret"
    }

    # Get OAuth Token
    while ([string]::IsNullOrEmpty($tokenRequestSource.access_token)) {

        $tokenRequestSource = try {
            Invoke-RestMethod -Method POST -Uri "https://login.microsoftonline.com/$tenantId/oauth2/v2.0/token" -
Body $tokenBodySource

        }
        catch {

            $errorMessageSource = $_.ErrorDetails.Message | ConvertFrom-Json

            # If not waiting for auth, throw error
            if ($errorMessageSource.error -ne "authorization_pending") {

                throw

            }

        }

    }
```

```
    $global:tokensource = $tokenRequestSource.access_token

    if($global:tokenSource){
        Write-output "Source Authorization successful!"

    }else{
        Write-Error "Source Authorization not successful. Do not continue! Check your credentials first!"
    }

}

Function Send-Logs(){

    param (
        [String]$LogType,
        [Hashtable]$LogBodyList
    )



    $LogBodyJSON = @"
    {
        "logtype": "$LogType",
        "logbody": {}
    }
"@

    $LogBodyObject = ConvertFrom-JSON $LogBodyJSON
    Foreach($Log in $LogBodyList.GetEnumerator()){
        $LogBodyObject.logbody | Add-Member -MemberType NoteProperty -Name $log.key -Value $log.value
    }
    $Body = ConvertTo-JSON $LogBodyObject

    $Response = Invoke-Restmethod -uri $FunctionUrl -Body $Body -Method POST -ContentType
"application/json"
    return $Response
}

function Send-ToTeams {
    $CurrentTime = Get-Date
    $Body = @{
```

```
"@context" = "https://schema.org/extensions"
"@type" = "MessageCard"
"themeColor" = "880808"
"title" = "RB-INT-ALL-PS1-ChangePrimaryUserByOwner-PROD ist durchgelaufen"
    "text" = @"
Parameter:
<ul><li>Successful primary user assignments in Intune: $($SuccessfulAssignmentDevices.Count)</li><li>Error
with assignments in Intune: $($ErrorAssignmentDevices.Count)</li><li>Error no LastSignIn on Device:
$($ErrorLastSignIn.count)</li></ul>
"@
}
$JsonBody = $Body | ConvertTo-JSON

Invoke-RestMethod -Method Post -Body $JsonBody -Uri $WebhookURI -Headers @{"content-type" =
"application/x-www-form-urlencoded; application/json; charset=UTF-8"} | out-null

}
#endregion Functions

#Graph Authentication
$tenantId=Get-AutomationVariable -Name "<yourfunctionstenantidvariable>"
$ClientId=Get-AutomationVariable -Name "yourfunctionsclientidvariable"
$CredentialObject=Get-AutomationPSCredential -Name 'yourfunctionsclientsecretsecret'
$ClientSecret = $CredentialObject.GetNetworkcredential().password

$WebhookURI = Get-AutomationVariable -Name "<yourteamsnotificationchannelwebhook>"
$FunctionUrl= Get-AutomationVariable -Name "<yourlogcollectionfunction>"

# SourceAuthorization
Get-ApplicationOnlySourceAuthorization -tenantId $tenantId -clientid $clientid -clientSecret $clientSecret

$SuccessfulAssignmentDevices = @()
$ErrorAssignmentDevices = @()
$ErrorLastSignIn = @()
$ExcludedServiceAccounts = @(
    "<yourserviceaccountswhichshouldnotbesetasprimaryusers>"
)

#Get Intune managed devices
$uri =
```

```
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices?`$filter=startswith(operatingSystem,'windows')"
$Results = Invoke-RestMethod -Method GET -Uri $uri -ContentType "application/json" -Headers @{Authorization = "Bearer $($Global:tokenSource)"; ConsistencyLevel = "eventual"}
$ResultsValue = $results.value
if ($results."@odata.nextLink" -ne $null) {
   $NextPageUri = $results."@odata.nextLink"
   ##While there is a next page, query it and loop, append results
   While ($NextPageUri -ne $null) {
      $NextPageRequest = (Invoke-RestMethod -Headers @{Authorization = "Bearer $($Global:tokenSource)"} -Uri $NextPageURI -Method Get)
      $NxtPageData = $NextPageRequest.Value
      $NextPageUri = $NextPageRequest."@odata.nextLink"
      $ResultsValue = $ResultsValue + $NxtPageData
   }
}
$IntuneDevices = $ResultsValue | where {($_.devicename -like "MW-*") -or ($_.devicename -like "CPC-*")}

foreach($IntuneDevice in $IntuneDevices){

   # get primary user if exists
   $uri = "https://graph.microsoft.com/beta/deviceManagement/managedDevices/$($IntuneDevice.id)/users"
   $UserObject = (Invoke-RestMethod -Method GET -Uri $uri -ContentType "application/json" -Headers @{Authorization = "Bearer $($Global:tokenSource)"}).value

   if(!$UserObject.userPrincipalName -or $ExcludedServiceAccounts -contains $UserObject.userPrincipalName){
      # get last signed in user of intune device
      $uri = "https://graph.microsoft.com/beta/deviceManagement/managedDevices/$($IntuneDevice.id)?`$select=usersLoggedOn"
      try{
         $Results = Invoke-RestMethod -Method GET -Uri $uri -ContentType "application/json" -Headers @{Authorization = "Bearer $($Global:tokenSource)"}
         $PrimaryUserId = $Results.UsersLoggedOn[0].userid
      }
      catch{
         Write-output "The intune device $($CurrentDevice.displayname) has no primary user!"
      }

      $PrimaryUser = ""
```

```
        if($PrimaryUserId){
            $uri = "https://graph.microsoft.com/v1.0/users/$PrimaryUserId"
            try{
                $PrimaryUser = Invoke-RestMethod -Method GET -Uri $uri -ContentType "application/json" -Headers
@{Authorization = "Bearer $($Global:tokenSource)"}
            }
            catch{
                Write-Error "The user id $PrimaryUserId has no UPN!"
            }
        }
        else{
            Write-Warning "No last user found for device $($IntuneDevice.deviceName)"
            $ErrorLastSignIn += $IntuneDevice.id
        }

        #configure primary owner
        if($PrimaryUser -and !($ExcludedServiceAccounts -contains $PrimaryUser.userPrincipalName)){
            $BodyPrimaryUser = @"
{
   "@odata.id": "https://graph.microsoft.com/beta/users/$($PrimaryUser.id)"
}
"@

            $uri =
"https://graph.microsoft.com/beta/deviceManagement/managedDevices('$($IntuneDevice.id)')/users/`$ref"
            try{
                $Results = Invoke-RestMethod -Method POST -Body $BodyPrimaryUser -Uri $uri -ContentType
"application/json" -Headers @{Authorization = "Bearer $($Global:tokenSource)"}
                Write-Output "Set intune primary user $($PrimaryUser.userPrincipalName) for device
$($IntuneDevice.deviceName)"
                $SuccessfulAssignmentDevices += $IntuneDevice.id
            }catch{
                Write-Error "Failed setting primary user $($PrimaryUser.userPrincipalName) for device
$($IntuneDevice.deviceName)"
                $ErrorAssignmentDevices += $IntuneDevice.id
            }
        }
    }
}
```

```
$Logs = @{
    "successassignment"="$($SuccessfulAssignmentDevices.count)"
    "errorassignment"="$($ErrorAssignmentDevices.count)"
    "errorlastsignnotfound"="$($ErrorLastSignIn.count)"
}

Send-Logs -LogType "ChangePrimaryUserByOwnerExecutions" -LogBodyList $Logs

Send-ToTeams
```

Set scope tag by domain of primary user with automation

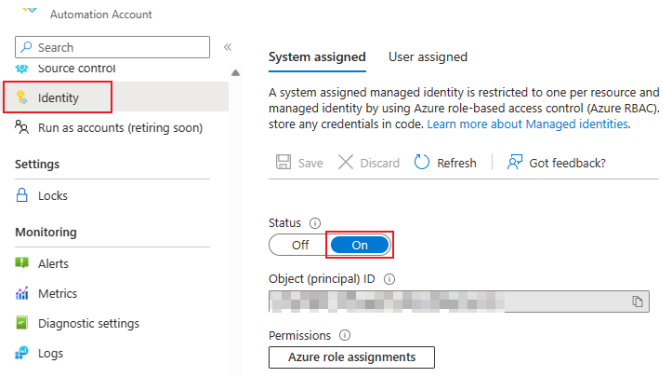
This automation solves a very small specific use case. As soon as the mobile devices (IOS, Android) are registered in Intune and are not set up via enrollment type profiles, no more scope tags can be set based on device parameters. This is where this automation comes into play.

The script runs regularly and goes through all IOS & Android devices in Intune. There it takes the primary user of the device and resolves the domain to the scope tag via an API. Then the script sets the scope tag in Intune via the API.

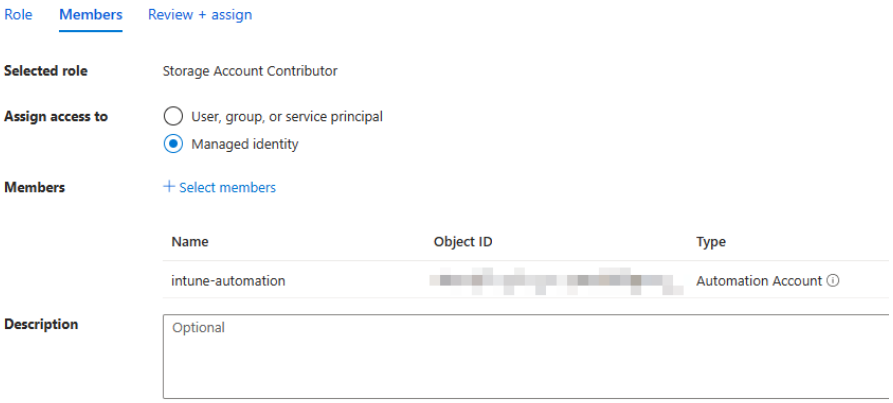
Prerequisites

Managed Identity

On the Automation Account you must activate the managed identity. This can be achieved in the settings under "Identity":



Then you can add the managed identity of the Azure Automation Account to the Storage Account with the permissions of "Storage Account Contributor":



App Registration

After that you have to create an App Registration. The process to create and how the variables can be found out, you will find in this tutorial: Get app details and gr... | LNC DOCS (lucanoahcaprez.ch)

The permissions on the App Registration that are necessary for this automation are as follows:

- Device.ReadWrite.All
- User.Read.All
- DeviceManagementConfiguration.ReadWrite.All,
- DeviceManagementRBAC.ReadWrite.All
- DeviceManagementManagedDevices.ReadWrite.All

PowerShell Script

With the following script this automation can be achieved. This is intended to be executed in a runbook. Accordingly, it is important that the variables are added to the runbook and automation account.

```
Param(
    $Device,
    $Domain
)

$Success = @()
$NoPM = @()
$NoScopeTag = @()
$NoShortName = @()
$NoOwner = @()
$NoDevice = @()
```

```
#Azure Login with Identity
try
{
   # "Logging in to Azure..."
   Connect-AzAccount -Identity
}
catch {
   Write-Error -Message $_.Exception
   throw $_.Exception
}

$resourcegroupname = "<yourresourcegroupname>"
$storageaccountname = "<yourstorageaccountname>"
$storagetablename = "<yourstoragetablename>"

#storage account/table connection
$storageaccountkey = (Get-AzStorageAccountKey -ResourceGroupName $resourcegroupname -AccountName
$storageaccountname | where {$_.keyname -eq "key1"}).value
$storageContext = New-AzStorageContext -StorageAccountName $storageaccountname -StorageAccountKey
$storageaccountkey
$table = Get-AzStorageTable -name $storagetablename -context $storageContext
$hostnames = get-AzTableRow -table $table.CloudTable

#Collect Credential Data for Graph API
$tenantId = Get-AutomationVariable -Name "<yourunbooktenantidvariable>"
$ClientId = Get-AutomationVariable -Name "<yourunbookclientidvariable>"
$CredentialObject = Get-AutomationPSCredential -Name "<yourrunbookclientsecretsecret>"
$ClientSecret = $CredentialObject.GetNetworkcredential().password
$WebhookURI = Get-AutomationVariable -Name "<yourteamswebhookuri>"
$FunctionURL = Get-AutomationVariable -Name "<yourfunctionapiurl>"

$Body = @{
   "tenant" = $TenantId
   "client_id" = $ClientId
   "scope" = "https://graph.microsoft.com/.default"
   "client_secret" = $ClientSecret
   "grant_type" = "client_credentials"
}

$Params = @{
```

```
    "Uri" = "https://login.microsoftonline.com/$TenantId/oauth2/v2.0/token"
    "Method" = "Post"
    "Body" = $Body
    "ContentType" = "application/x-www-form-urlencoded"
}
$AuthResponse = Invoke-RestMethod @Params

$Header = @{
    "Authorization" = "Bearer $($AuthResponse.access_token)"
    #"ConsistencyLevel" = "eventual"
}

Function Send-Logs(){

    param (
        [String]$LogType,
        [Hashtable]$LogBodyList
    )



    $LogBodyJSON = @"
    {
        "logtype": "$LogType",
        "logbody": {}
    }
"@

    $LogBodyObject = ConvertFrom-JSON $LogBodyJSON
    Foreach($Log in $LogBodyList.GetEnumerator()){
        $LogBodyObject.logbody | Add-Member -MemberType NoteProperty -Name $log.key -Value $log.value
    }
    $Body = ConvertTo-JSON $LogBodyObject

    $Response = Invoke-Restmethod -uri $FunctionURL -Body $Body -Method POST -ContentType
"application/json"
    return $Response
}

function Send-ToTeams {
    $CurrentTime = Get-Date
```

```
    $Body = @{
"@context" = "https://schema.org/extensions"
"@type" = "MessageCard"
"themeColor" = "880808"
"title" = "Automation completed:"
    "text" = @"
Parameter:
<ul><li>Successful assignments in Intune: $($SuccessChanged.Count)</li><li>Already correct assignment in
Intune: $($SuccessAlready.Count)</li><li>Scope tag not found in Intune:
$($NoScopeTag.Count)</li><li>Shortname by domain not found in storage table:
$($NoShortName.Count)</li><li>Owner not found in Intune: $($NoOwner.Count)</li></ul>
"@
}

    $JsonBody = $Body | ConvertTo-JSON

    Invoke-RestMethod -Method Post -Body $JsonBody -Uri $WebhookURI -Headers @{"content-type" =
"application/json; charset=UTF-8"} | out-null
}

#get all Scope Tags
$uri = "https://graph.microsoft.com/beta/deviceManagement/roleScopeTags"
$Results = Invoke-RestMethod -Method GET -Uri $uri -ContentType "application/json" -Headers $header
$scopeTags = $results.value

if($device){

    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices/$($device)" -ContentType
"Application/Json" -Header $Header
    $managedDevices += $Results
    # $managedDevices
}elseif($Domain){
    Add-Type -AssemblyName System.Web

    $encodedURL = [System.Web.HttpUtility]::UrlEncode($Domain)

    #get android and ios managed devices
    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices?`$filter=(contains(activationlockbypass
```

```
code,%20%27$encodedURL%27))%20and%20startswith(operatingSystem,'ios')" -ContentType "Application/Json"
-Header $Header
    $ResultsValue = $Results.value
    if ($results."@odata.nextLink" -ne $null) {
        $NextPageUri = $results."@odata.nextLink"
        ##While there is a next page, query it and loop, append results
        While ($NextPageUri -ne $null) {
            $NextPageRequest = (Invoke-RestMethod -Headers $Header -Uri $NextPageURI -Method Get)
            $NxtPageData = $NextPageRequest.Value
            $NextPageUri = $NextPageRequest."@odata.nextLink"
            $ResultsValue = $ResultsValue + $NxtPageData
        }
    }
    $managedDevices += $ResultsValue


    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices?`$filter=(contains(activationlockbypass
code,%20%27$encodedURL%27))%20and%20startswith(operatingSystem,'android')" -ContentType
"Application/Json" -Header $Header
    $ResultsValue = $Results.value
    if ($results."@odata.nextLink" -ne $null) {
        $NextPageUri = $results."@odata.nextLink"
        ##While there is a next page, query it and loop, append results
        While ($NextPageUri -ne $null) {
            $NextPageRequest = (Invoke-RestMethod -Headers $Header -Uri $NextPageURI -Method Get)
            $NxtPageData = $NextPageRequest.Value
            $NextPageUri = $NextPageRequest."@odata.nextLink"
            $ResultsValue = $ResultsValue + $NxtPageData
        }
    }

    $managedDevices += $ResultsValue

    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices/$($device)" -ContentType
"Application/Json" -Header $Header
    $managedDevices += $Results
    # $managedDevices
```

```
}else{
    #get android and ios managed devices
    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices?`$filter=startswith(operatingSystem,'io
s')" -ContentType "Application/Json" -Header $Header
    $ResultsValue = $Results.value
    if ($results."@odata.nextLink" -ne $null) {
        $NextPageUri = $results."@odata.nextLink"
        ##While there is a next page, query it and loop, append results
        While ($NextPageUri -ne $null) {
            $NextPageRequest = (Invoke-RestMethod -Headers $Header -Uri $NextPageURI -Method Get)
            $NxtPageData = $NextPageRequest.Value
            $NextPageUri = $NextPageRequest."@odata.nextLink"
            $ResultsValue = $ResultsValue + $NxtPageData
        }
    }
    $managedDevices += $ResultsValue


    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices?`$filter=startswith(operatingSystem,'an
droid')" -ContentType "Application/Json" -Header $Header
    $ResultsValue = $Results.value
    if ($results."@odata.nextLink" -ne $null) {
        $NextPageUri = $results."@odata.nextLink"
        ##While there is a next page, query it and loop, append results
        While ($NextPageUri -ne $null) {
            $NextPageRequest = (Invoke-RestMethod -Headers $Header -Uri $NextPageURI -Method Get)
            $NxtPageData = $NextPageRequest.Value
            $NextPageUri = $NextPageRequest."@odata.nextLink"
            $ResultsValue = $ResultsValue + $NxtPageData
        }
    }

    $managedDevices += $ResultsValue
}

write-output "Intune Devices: $($managedDevices.count)"

#compare and collect MEID devices, which are managed by intune
```

```
foreach($managedDevice in $managedDevices){
    #configure hostname if device has an owner
    if($managedDevice.userPrincipalName){
        #get domainname of user
        $dn = ($managedDevice.userprincipalname -split "@")[1]
        $shortname = ($hostnames | where {$_.dn -eq $dn}).shortname
        if($shortname.count -gt 1){
            $shortname = $shortname[0]
        }
        if($shortname){
            #check if scope tag exists based on azure table
            $scopeTagName = "SCT-INT-$shortname-INTUNE-*-PROD"
            $scopeTag = $scopeTags | where {$_.displayName -like $scopeTagName}
            if($scopeTag){

                $uri =
"https://graph.microsoft.com/beta/deviceManagement/managedDevices('$($managedDevice.id)')"
                $DeviceObject = Invoke-RestMethod -Uri $uri -Headers $header -Method GET -ContentType
"application/json"
                if($DeviceObject.roleScopeTagIds -contains $scopeTag.id){
                    $SuccessAlready += $managedDevice.id
                }
                else{
                    $assignBody = @{
                        roleScopeTagIds = @("$($scopeTag.id)")
                    }

                    $JSON = $assignBody | ConvertTo-Json
                    Invoke-RestMethod -Uri $uri -Headers $header -Method Patch -Body $JSON -ContentType
"application/json"
                    $SuccessChanged += $managedDevice.id
                }

            } else {
                write-warning "$($managedDevice.id) - Scope Tag $scopeTagName does not exist!"
                $NoScopeTag += $managedDevice.id
            }
        } else {
            write-warning "$($managedDevice.id) - Domain $dn not in Storage Table!"
            $NoShortName += $managedDevice.id
```

```
            }
        } else {
            write-warning "$($managedDevice.id) - Device has no Owners"
            $NoOwner += $managedDevice.id
        }
}

$Logs = @{
    "successchangedassignment"="$($SuccessChanged.count)"
    "successalreadyassigned"="$($SuccessAlready.count)"
    "errornoscopetag"="$($NoScopeTag.count)"
    "errornoshortname"="$($NoShortName.count)"
    "errornoowner"="$($NoOwner.count)"
}

Send-Logs -LogType "MobileDeviceScopingByPrimaryUserExecutions" -LogBodyList $Logs

Send-ToTeams
```

Start Intune Device Sync via Graph API

> Requirements: Microsoft Entra ID Authentication token is needed to use this script and the Graph API.

This tool allows you to initiate Intune Sync on multiple or all devices. The Graph API is accessed via PowerShell and triggers the sync on the devices.

Permissions

This script uses Graph API and authenticates with an App Registration or User based access token. The App Registration or the user context needs the following Microsoft Graph permission:

DeviceManagementManagedDevices.PrivilegedOperations.All

This permission can be set either as application permission or as delegated permission.

PowerShell Script

In preparation, the Microsoft Entra ID access token from the previous step must be stored in this variable: $Global:AzureADAccessToken

Then this script can be executed. Here, the sync of all Windows devices in Intune is triggered.

```
$uri =
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices?`$filter=startswith(operatingSystem,'windows')"
$Results = Invoke-RestMethod -Method GET -Uri $uri -ContentType "application/json" -Headers @{Authorization = "Bearer $($Global:MicrosoftEntraIDAccessToken)"; ConsistencyLevel = "eventual"}
$ResultsValue = $results.value
if ($results."@odata.nextLink" -ne $null) {
    $NextPageUri = $results."@odata.nextLink"
    ##While there is a next page, query it and loop, append results
    While ($NextPageUri -ne $null) {
```

```
        $NextPageRequest = (Invoke-RestMethod -Headers @{Authorization = "Bearer
$($Global:MicrosoftEntraIDAccessToken)"} -Uri $NextPageURI -Method Get)
        $NxtPageData = $NextPageRequest.Value
        $NextPageUri = $NextPageRequest."@odata.nextLink"
        $ResultsValue = $ResultsValue + $NxtPageData
    }
}
$IntuneDevices = $ResultsValue | where {$_.devicename -like "MW-*"}

$SuccessDevices = @()
$ErrorDevices = @()

foreach($IntuneDevice in $IntuneDevices){
    try{
        $uri =
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices('$($IntuneDevice.id)')/syncDevice"
        Invoke-RestMethod -uri $uri -Method POST -Headers @{Authorization =
"$($Global:MicrosoftEntraIDAccessToken)"}
        Write-Output "Started Sync for " $IntuneDevice.devicename
        $SuccessDevices += $IntuneDevice
    }catch{
        Write-Output "Error while syncing " $IntuneDevice.devicename
        $ErrorDevices += $IntuneDevice
    }
}
```

Interactively enroll Windows Device with Autopilot

To enroll a Windows device into Intune via Windows Autopilot, it is needed to register the serial number and hardware id in the tenant via an interactive login. The user has to log in as "Intune Administrator" and upload the information via Graph API into Intune. A GroupTag can also be set in the same step.

Guide

Open CMD window with "SHIFT" + "F10".

Then enter the following commands in this order. **Important:** Change the values for your GroupTag and tenant name.

```
powershell
```

```
Set-ExecutionPolicy Unrestricted
```

```
Install-Script -Name "Get-WindowsAutopilotInfo" -Force
```

```
Get-WindowsAutopilotInfo.ps1 -Online -GroupTag "<yourgrouptagname>" -TenantId "<yourtenantname>.onmicrosoft.com”
```

Then you have to login with an account that has Intune Administrator activated.

```
Restart-Computer
```

After the restart it will show you the welcome screen from your company there you can login with a normal account, if the person is authorized to enroll (device enrollment settings).

Troubleshooting

If the time is not set correctly or is not recognized, the install script may fail. The SSL certificate check requires a correct time.

Evaluate installed app version from devices via Graph API

Managing applications across a fleet of devices is a critical task for IT administrators. With this PowerShell script you can leverage the power of Microsoft Intune and the Microsoft Graph API to streamline application inventory management.

By using this script, you can efficiently track application installations and versions, aiding in license compliance, security updates, and software distribution planning. It empowers IT teams to make informed decisions about their application landscape, ultimately enhancing device management and security.

PowerShell Script

Add your own access token and app name in the corresponding PowerShell variables.

```
$Global:MicrosoftEntraIDAccessToken = "<youraccesstoken>"
$AppName = "<yourappname>"
$PlattformOS = "<yourplattform>" #possible values are windows, ios, macos, android
$AllDevicesWithAppVersion = @()

#Get Intune managed devices
$uri =
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices?`$filter=startswith(operatingSystem,'$P
lattformOs')"
$Results = Invoke-RestMethod -Method GET -Uri $uri -ContentType "application/json" -Headers @{Authorization
= "Bearer $($Global:MicrosoftEntraIDAccessToken)"; ConsistencyLevel = "eventual" }
$IntuneDevices = $results.value
if ($results."@odata.nextLink" -ne $null) {
    $NextPageUri = $results."@odata.nextLink"
    # While there is a next page, query it and loop, append results
    While ($NextPageUri -ne $null) {
        $NextPageRequest = (Invoke-RestMethod -Headers @{Authorization = "Bearer
```

```
$($Global:MicrosoftEntraIDAccessToken)" } -Uri $NextPageURI -Method Get)
        $NxtPageData = $NextPageRequest.Value
        $NextPageUri = $NextPageRequest."@odata.nextLink"
        $IntuneDevices += $NxtPageData
    }
}

# Get Apps when
foreach ($IntuneDevice in $IntuneDevices) {
    try {
        $AppsUri =
"https://graph.microsoft.com/beta/deviceManagement/manageddevices('$($IntuneDevice.id)')/detectedApps?`$t
op=100&`$filter=contains(displayName,%20%27$($AppName)%27)&`$orderBy=displayName%20asc"
        $Apps = Invoke-RestMethod -Method GET -Uri $AppsUri -ContentType "application/json" -Headers
@{Authorization = "Bearer $($Global:MicrosoftEntraIDAccessToken)"; ConsistencyLevel = "eventual" }
        $IntuneDevice | Add-Member -NotePropertyName "$AppName AppVersion" -NotePropertyValue
$apps.value.version
        Write-Output "$($IntuneDevice.deviceName) -> $($apps.value.version)"
    }
    catch {
        Write-Output "Sleeping..."
        Start-Sleep 10
        $AppsUri =
"https://graph.microsoft.com/beta/deviceManagement/manageddevices('$($IntuneDevice.id)')/detectedApps?`$t
op=100&`$filter=contains(displayName,%20%27$($AppName)%27)&`$orderBy=displayName%20asc"
        $Apps = Invoke-RestMethod -Method GET -Uri $AppsUri -ContentType "application/json" -Headers
@{Authorization = "Bearer $($Global:MicrosoftEntraIDAccessToken)"; ConsistencyLevel = "eventual" }
        $IntuneDevice | Add-Member -NotePropertyName "$AppName AppVersion" -NotePropertyValue
$apps.value.version
        Write-Output "$($IntuneDevice.deviceName) -> $($apps.value.version)"
    }
    $AllDevicesWithAppVersion += $IntuneDevice
}

$AllDevicesWithAppVersion | Export-CSV ".\$(Get-Date -Format yyMMdd) AllDevicesWithAppVersion.csv"
```

Get duplicate Intune devices by serialnumber

Sometimes it happens that there are several devices with the same serial number in Intune. This can happen for example when switching from AD/SCCM built clients to Intune only clients.

This script helps to find the duplicate entries. At the end a CSV is output, which can be cleaned manually or with another script.

This script fetches all Intune Devices and then goes through and, if there are multiple devices per serial number, stores the device information in an array.

PowerShell Script

Add your own access token and app name in the corresponding PowerShell variables.

```
$Global:MicrosoftEntraIDAccessToken = ""

# Get All Intune Windows Devices
$uri =
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices?`$filter=startswith(operatingSystem,'windows')"
$Results = Invoke-RestMethod -Method GET -Uri $uri -ContentType "application/json" -Headers @{Authorization = "Bearer $($Global:MicrosoftEntraIDAccessToken)"; ConsistencyLevel = "eventual" }
$IntuneDevices = $results.value
if ($results."@odata.nextLink" -ne $null) {
   $NextPageUri = $results."@odata.nextLink"
   While ($NextPageUri -ne $null) {
      $NextPageRequest = (Invoke-RestMethod -Headers @{Authorization = "Bearer $($Global:MicrosoftEntraIDAccessToken)" } -Uri $NextPageURI -Method Get)
      $NxtPageData = $NextPageRequest.Value
      $NextPageUri = $NextPageRequest."@odata.nextLink"
      $IntuneDevices += $NxtPageData
   }
}
```

```
# Get all devices by SerialNumber
$DuplicateDevices = @()
foreach($IntuneDevice in $IntuneDevices){
    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/beta/deviceManagement/managedDevices?`$filter=contains(serialNumber,'$($Intu
neDevice.serialnumber)')" -ContentType "application/json" -Headers @{Authorization = "Bearer
$($Global:MicrosoftEntraIDAccessToken)"}
    if($Results.value.count -gt 1){
        $Results.value.serialnumber
        if($DuplicateDevices.serialnumber -contains $Results.value.serialnumber){
            $DuplicateDevices += $Results.value
            $DuplicateDevices
        }
        else{
            $DuplicateDevices += $Results.value
        }
    }
}

# Output all duplicate Devices
$DuplicateDevices | Export-CSV "<yourpathtocsv>" -NoTypeInformation
```

Download win32 intunewin content file via Graph API

This tutorial is about how to download intunewin content according to an Intune App ID. It is only about the file. The other configurations outside the file can be fetched via another endpoint as JSON.

The script first gets the file version and then the storage URL from the storage account that Microsoft uses for storing the intunewin file.

Authentication

This script relies only on REST Calls to the Microsoft Graph API. The authentication is based on OAuth 2.0 which relies on access tokens. How you can create an access token as a user or system identity is described here: Create user access tok... | LNC DOCS (lucanoahcaprez.ch)

Limitations (currently)

ATTENTION: The file is just downloaded. It will then be available in the specified folder. Unfortunately, it is currently not possible to upload the file again because the header is somehow cut off and the file size is therefore not the same.

PowerShell Script

This script requires an access token for Microsoft Entra ID, the Intune App ID and the output folder. Save the values into the corresponding PowerShell variables.

```
$Global:MicrosoftEntraIDAccessToken = "<youraccesstoken>"
$AppID = "<yourintuneappid>"
$Path = "<youroutputpathforfile>"


$Win32AppContentVersions = (Invoke-RestMethod -Uri
```

```
"https://graph.microsoft.com/beta/deviceAppManagement/mobileApps/$($AppID)/microsoft.graph.win32LobApp/contentVersions" -Method "GET" -Headers @{Authorization = "$($Global:MicrosoftEntraIDAccessToken)" } -ContentType 'application/json').value
switch ($Win32AppContentVersions.Count) {
    0 {
        Write-Warning -Message "Unable to locate any contentVersions resources for specified Win32 app"
    }
    1 {
        Write-Verbose -Message "Located contentVersions resource with ID: $($Win32AppContentVersions.id)"
        $Win32AppContentVersionID = $Win32AppContentVersions.id
    }
    default {
        Write-Verbose -Message "Located '$($Win32AppContentVersions.Count)' contentVersions resources for specified Win32 app, attempting to determine the latest item"
        $Win32AppContentVersionID = $Win32AppContentVersions | Sort-Object -Property id -Descending | Select-Object -First 1 -ExpandProperty id
    }
}
if ($Win32AppContentVersions.Count -ge 1) {
    $Win32AppContentVersionsFiles = (Invoke-RestMethod -Uri "https://graph.microsoft.com/beta/deviceAppManagement/mobileApps/$($AppID)/microsoft.graph.win32LobApp/contentVersions/$($Win32AppContentVersionID)/files" -Method "GET" -Headers @{Authorization = "$($Global:MicrosoftEntraIDAccessToken)" } -ContentType 'application/json').value
    if ($Win32AppContentVersionsFiles -ne $null) {
        foreach ($Win32AppContentVersionsFile in $Win32AppContentVersionsFiles) {
            try {
                $Win32AppContentVersionsFileResource = Invoke-RestMethod -Uri "https://graph.microsoft.com/beta/deviceAppManagement/mobileApps/$($AppID)/microsoft.graph.win32LobApp/contentVersions/$($Win32AppContentVersionID)/files/$($Win32AppContentVersionsFile.id)" -Method "GET" -Headers @{Authorization = "$($Global:MicrosoftEntraIDAccessToken)" } -ContentType 'application/json'
            }
            catch {}
            if ($Win32AppContentVersionsFileResource -ne $null) {
                Invoke-RestMethod -Uri $Win32AppContentVersionsFileResource.azureStorageUri -Method "GET" -OutFile "$Path\$($Win32AppContentVersionsFiles.indexOf($Win32AppContentVersionsFile)).intunewin"
            }
        }
    }
}
```

Windows Update during OOBE using Intune App

Windows updates are not installed by default when the device is in the Autopilot phase. This guide helps you to omit some manual tasks during OOBE phase. In some cases the time until WufB takes care of the update process takes too long. If you want to install the latest Windows updates during Autopilot deployment, you've come to the right place.

Update OS solution by

oofhours.com

First we have to create an Intune application. This app contains the script that will force to update the Windows OS. Its based on the PowerShell Module PSWindowsUpdate. So that we don't have to reinvent the wheel, here are instructions for providing the solution: Installing Windows updates during a Windows Autopilot deployment – Out of Office Hours (oofhours.com)

All scripts and sources are located in this Github repository: mtniehaus/UpdateOS: Sample app for installing Windows updates during an Autopilot deployment (github.com)

Since we only want this app to be deployed during OOBE, we can use this solution to restrict the installation to OOBE session: Restrict app installat... | LNC DOCS (lucanoahcaprez.ch)

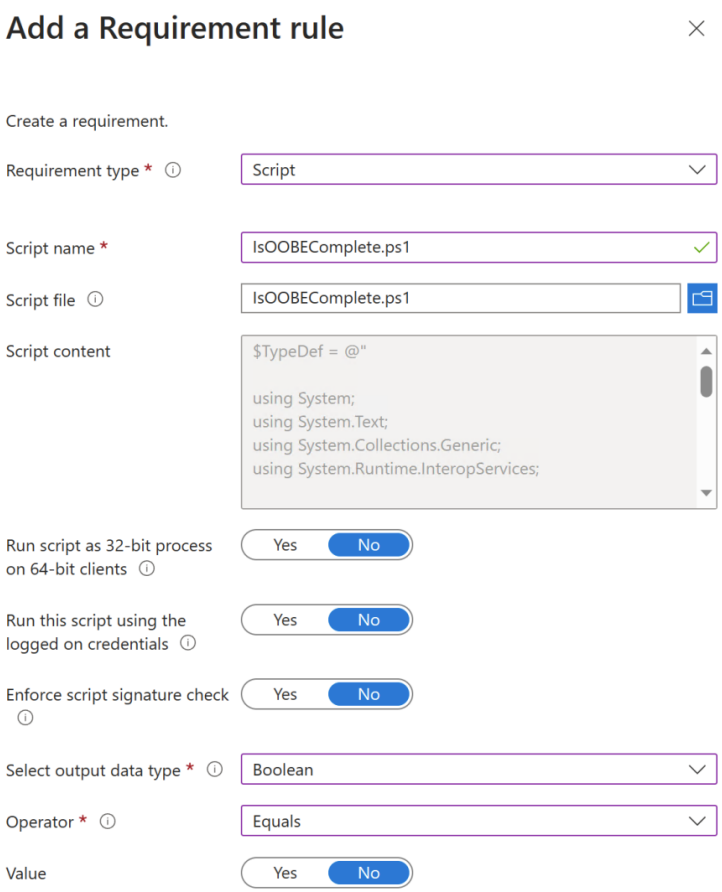
Restrict app installation only during OOBE

Some application have the requirement to only be installed during the Autopilot provisioning.

Since Intune apps offers the possibility to run a PowerShell script before app installation, we are going to use this function to check whether the device is in OOBE mode or not. This feature is referred to as “Requirement rule” and can be configured the follows:

Instructions for replicating

Open or create an Intune app. Go to requirement rule and create a new rule with type “script”.



Then enter all the necessary steps and configure the rule as in the screenshot.

Script to report OOBE status

This script return if the device currently is in OOBE mode or not. Output "true" means the Autopilot mode is finished. Output "false" means, its currently in OOBE / Autopilot mode.

```
$TypeDef = @"

using System;
using System.Text;
using System.Collections.Generic;
using System.Runtime.InteropServices;

namespace Api
{
 public class Kernel32
 {
  [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
  public static extern int OOBEComplete(ref int bIsOOBEComplete);
 }
}
"@

Add-Type -TypeDefinition $TypeDef -Language CSharp

$IsOOBEComplete = $false
$hr = [Api.Kernel32]::OOBEComplete([ref] $IsOOBEComplete)

$IsOOBEComplete
```

Original instructions and credits: Detecting when you are in OOBE – Out of Office Hours (oofhours.com)