How to force Intune configuration scripts to re-run

By Ben, In Intune, Powershell
10,110 views

Hi All and welcome.

As I am about to reach the pointy end of a project to implement an Intune MDM solution for a client, I've taken a moment to take stock of the lessons learned, problems faced and for the most part; the cool things I've run into and decided now is the time to start writing about them! Hopefully you find my posts interesting and I hope to keep the page updated fairly regularly.

Anyway, lets move onto the fun stuff!

As I mentioned, I've been working on an Intune MDM solution for a client who currently has no other management solutions in place (no SCCM, no mobile device management, nothing, nada, zilch. you get the idea) which was daunting to say the least, but it did give us a great opportunity to provide an entirely cloud-centric management solution (absolutely no on-premise requirements – devices are not domain-joined!).

Because of these design decisions, we have had to be very creative with how we deploy applications & how we can replicate group policy configurations – what that essentially means is that we relied **very heavily** on the Intune Management Extension – previously known as **sidecar**.

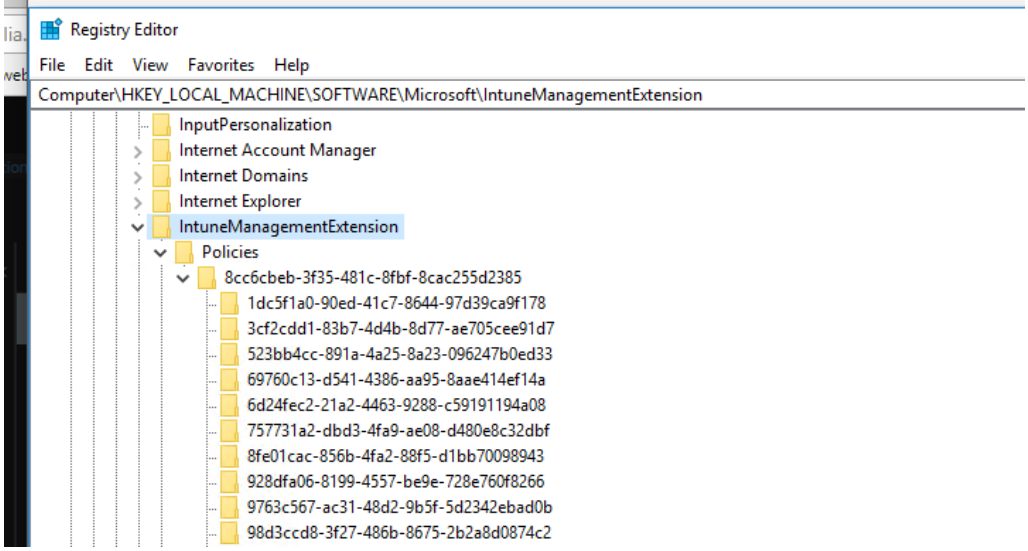
Because Intune *currently* only allows single file line-of-business applications, for anything more complex than that (read: most legacy LOB applications), handling the installation using Powershell via the Intune Management Extension is the best solution.

Now, while I am ecstatic that there is a script deployment solution within Intune; there is definitely challenges with the current implementation – case in point, the client reached out to me and asked me a very good question the other day… "how can we re-run the script if the script returned a successful result, but the expected result of the script was not achieved??"

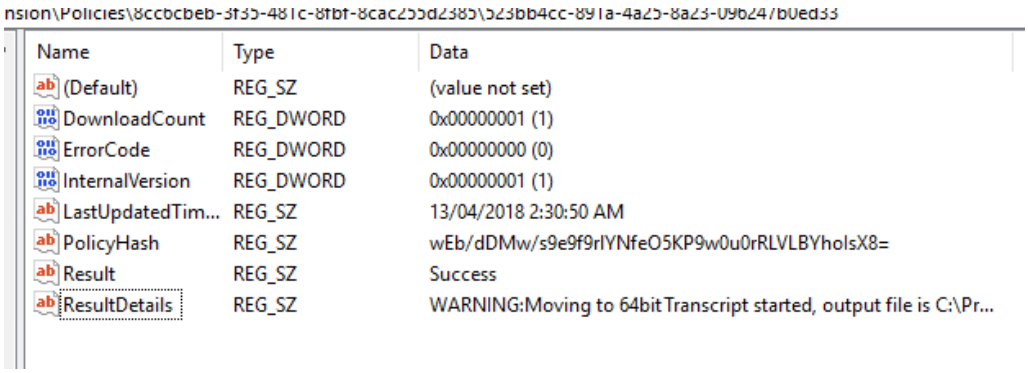
A quick explanation – The way that the Intune Management Extension handles execution of scripts is that it will attempt to run the script until it successfully completes. If it fails, it will attempt again in an hour (the Intune Management Extension synchronizes to Intune once every hour), however if for any reason you want a script to re-run, the only obvious solution is to delete the configuration item from within the Intune portal, recreate the configuration item and restart the **IntuneManagementExtension** service on the local device (as well as any other device or user that is in the assignment group) …

If you are shaking your head and saying "there has to be a better way", then read on for the solution!

The Intune Management Extension stores details of configuration scripts that have executed in a specific registry location:
HKLM:\SOFTWARE\Microsoft\IntuneManagementExtension\Policies
If you have a look there, you'll see a list of executed items – all with unique GUIDs.



Inside each folder, you will see a breakdown of what is stored locally.



As you can see above, the script has downloaded once, there are no errors, and even cooler – the **ResultDetails** property has the full transcript of the script.

Now, the downside here is that aside from digging into the ResultDetails item property, there isn't an easy way to decipher which configuration item you are looking at. If you can figure out how to identify the script from the ResultDetails item property, then all that is required to trigger a re-run is to delete that item from the registry and restart the **IntuneManagementExtension** service on the local device.

Now we are getting somewhere.

Because the configuration items are stored in keys named with GUIDs, this should give anyone with experience with Intune or Azure in general, that if we can get a GUID id, then we should be able to extract more data by using the Graph API.

Alright, lets break down the solution.

First up – lets connect to the API...

In the code below I am using a module written by Jan Egil Ring to allow unattended authentication // non-interactive authentication against the generic Intune client application. I'm also checking to see if we have already authenticated and if so, only requesting a new token if the existing one has expired which is helpful for this scenario where the script may need to be run multiple times while doing functional testing / validation.

```
function Get-IntuneToken {
    param (
        $credential,
        $token
    )
    if (!(Get-Module -Name MSGraphIntuneManagement -ListAvailable -Err
        Install-Module -Name MSGraphIntuneManagement -Scope CurrentUse
    }
    $GMTDate = [System.TimeZoneInfo]::ConvertTimeBySystemTimeZoneId($(
    if ($token -ne $null) {
        $tokenExpDate = ([System.DateTimeOffset]$token.ExpiresOn).Date
        if ($GMTDate -le $tokenExpDate) {
            write-host "Token is still fresh." -ForegroundColor Green
            return $token
        }
        #token is technically expired or never existed.
    }
    Write-Host "Token is stale or never existed." -ForegroundColor Red
    $clientId = "d1ddf0e4-d672-4dae-b554-9d5bdfd93547"
    $token = Get-MSGraphAuthenticationToken -Credential $Credential -C
    return $token
}
if (!($cred)) {
    $cred = Get-Credential
}

$token = Get-IntuneToken -credential $cred -token $token
```

Reset-SidecarScript.ps1 hosted with ❤ by GitHub view raw

Once we have our auth token, lets capture some handy information to identify each script stored in the IntuneManagementExtension registry hive.

First up, lets get some info about the device.

```
$deviceProps = (invoke-RestMethod -Method Get -Uri "https://graph.micro
```

Reset-SidecarScript.ps1 hosted with ❤ by GitHub view raw

Next, using the device id captured above, lets grab some info about the registered user of that device.

```
$owner = (Invoke-RestMethod -Method Get -Uri "https://graph.microsoft.c
```

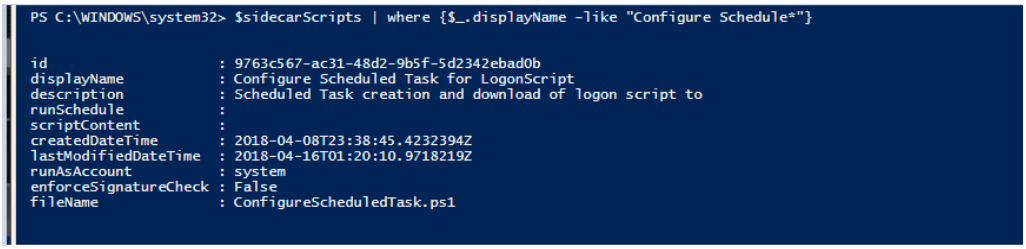
Reset-SidecarScript.ps1 hosted with ❤ by GitHub view raw

and finally, lets capture the script properties from Intune.

```
$sidecarScripts = (Invoke-RestMethod -Method Get -Uri "https://graph.mi
```

Reset-SidecarScript.ps1 hosted with ❤ by GitHub view raw

Here's an example of the data returned from the above API call.



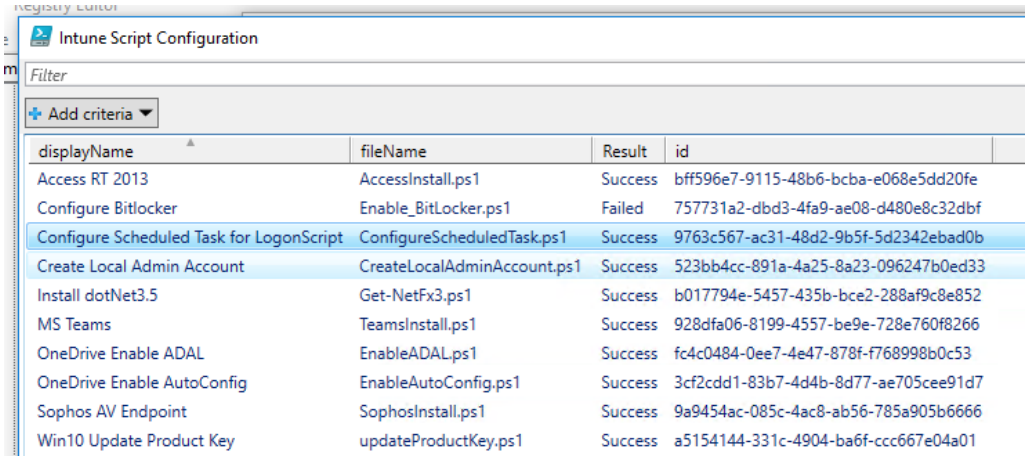
Now, using the user id GUID, we simply iterate through each script object stored in Intune, match it up with the policy objects stored locally and present the combined data to the end user.

```
$deviceScriptStatus = @()
foreach ($script in $sidecarScripts) {
    $tmpItem = Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\IntuneManage
    if ($tmpItem) {
        $tmpObj = [PSCustomObject]@{
            displayName = $script.displayName
            fileName    = $script.fileName
            Result      = $tmpItem.Result
            id          = $script.id
            psPath      = $tmpItem.PSPath
        }
        $deviceScriptStatus += $tmpObj
    }
```

```
}
$intuneScriptToRerun = $deviceScriptStatus | Select-Object displayName
```

Reset-SidecarScript.ps1 hosted with ❤ by GitHub view raw

Here's the example result of the above snippet – an interactive out-gridview datatable that will pass back any selected objects to the powershell window.



displayName	fileName	Result	id
Access RT 2013	AccessInstall.ps1	Success	bff596e7-9115-48b6-bcba-e068e5dd20fe
Configure Bitlocker	Enable_BitLocker.ps1	Failed	757731a2-dbd3-4fa9-ae08-d480e8c32dbf
Configure Scheduled Task for LogonScript	ConfigureScheduledTask.ps1	Success	9763c567-ac31-48d2-9b5f-5d2342ebad0b
Create Local Admin Account	CreateLocalAdminAccount.ps1	Success	523bb4cc-891a-4a25-8a23-096247b0ed33
Install dotNet3.5	Get-NetFx3.ps1	Success	b017794e-5457-435b-bce2-288af9c8e852
MS Teams	TeamsInstall.ps1	Success	928dfa06-8199-4557-be9e-728e760f8266
OneDrive Enable ADAL	EnableADAL.ps1	Success	fc4c0484-0ee7-4e47-878f-f768998b0c53
OneDrive Enable AutoConfig	EnableAutoConfig.ps1	Success	3cf2cdd1-83b7-4d4b-8d77-ae705cee91d7
Sophos AV Endpoint	SophosInstall.ps1	Success	9a9454ac-085c-4ac8-ab56-785a905b6666
Win10 Update Product Key	updateProductKey.ps1	Success	a5154144-331c-4904-ba6f-ccc667e04a01

So, for this example, I want to re-run the "ConfigureScheduledTask.ps1" script, so we select that row, hit OK on the Out-GridView to send that object back to the script, and using that object, we simply force a removal of that registry key and restart the IntuneManagementExtension service to trigger the script to re-run.

```
foreach ($item in $intuneScriptToRerun){
    $itemPath = ($deviceScriptStatus | Where-Object {$_.displayName -eq
    Remove-Item $itemPath -Force
}
Get-Service -Name IntuneManagementExtension | Restart-Service
```

Reset-SidecarScript.ps1 hosted with ❤ by GitHub view raw

You will find that the script / policy will re-run almost immediately once the registry key has been removed. This will save you countless hours over the course of setting up your sidecar scripts – something I wish I had worked out at the start of the project and not the end!!

Well that wraps up my first post – I will have the full solution available on my GitHub account for your perusal (link here), so please have a look, have a play, and if you use the example, or improve the solution, please feel free to let me know below in the comments or on my twitter @powers_hell.

Enjoy,
Ben

Azure, Intune, Powershell, Sidecar