Onevinn AB

IPU Installer

Installation and user's manual

onevinn

CONTENTS

1. VERSION

Version	Author	Date	Remark
1.0	Johan Schrewelius	2021-02-10	Document created

2. DISCLAIMER

The methods and implementations described in this document are used entirely at own risk.

3. DESCRIPTION

This document covers the **Onevinn IPU Installer**, a .Net executable that wraps necessary commands, logic and processes when using a Configuration Manager Application to apply a Windows feature update.

As an addition to the <<Installer>> we also wish to introduce our <<DeploymentScheduler>> to the world, it is still somewhat a beta product due to incomplete testing (primarily not enough, we consider it stable thou).

4. BACKGROUND

2020 was a challenging year in more than one way, not only did the pandemic impact us all personally, still does, it also changed many things on a professional level, specifically how people work. Where “work-from-home” scenarios had previously been edged cases is now the new normal.

We can add to the equation that many companies (customers in our case) have what must be considered outdated security policies and technical limitation in bandwidth, internet or vpn. Policies may for example prohibit split-tunneling, thus forcing all traffic through corporate firewalls and various other infrastructural do-hickeys.

Consequently, old trues are no longer valid and we have had to reconsider the way we approach the challenge of keeping Windows up to date. A Task sequence is not always the best approach.

5. CONTENT

All content needed to implement this solution can be downloaded from:

Onevinn - Featured (schrewelius.it)

6. CONCEPT

This concept utilizes fully patched Windows media; most used tool for constructing and maintaining such media (.iso) would be OSDBuilder Overview - OSDBuilder (osdeploy.com) by @SeguraOSD, but there are others. Or you might want to build your own solution based on for example this article: Update Windows 10 media with Dynamic Update - Windows Deployment | Microsoft Docs

Even thou it might appear slightly unorthodox an Application has many useful properties when it comes to running In Place Upgrades. It is designed to download and run on the client, it is stateful in that if the detection method requirements are not met (the correct OS is not installed), it will try again.

The solution combines fully patched media with the usage of the not known enough possibility to use an ini-file (SetupConfig.ini) to control the upgrade process. All logic has been wrapped in a .Net installer called <<**IPUInstaller.exe**>>, that also gathers SetupDiag results and overall status and writes it to common location in the registry. This registry location is then inventoried and sent to the site DB where it can be used for creating membership rule(s) for Collection and reports.

By extending the hardware inventory we can keep good track of the upgrade process even if the computer should perform the upgrade completely offline, for example on an employees' kitchen table. Once the computer gets back online and post a hardware inventory we will learn about its status.

The rest of this document will describe how to set this up, we will:

- Extend the Hardware Inventory
- Create Collections.
- Create the IPU Application.
- Create a Drivers Task Sequence.
- Deploy the Application in two different ways (with and without the Deployment Scheduler).

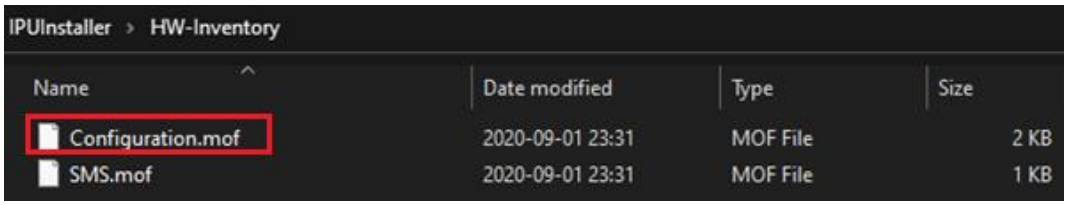
When all things are done, we will have an Applications that upgrades Windows and various collections to store the computers as they change state during the progress. We will also have setup driver management and learned about Deployment scheduling, with and without 3rd party assistance.

Reference: Windows Setup Automation Overview | Microsoft Docs

7. HARDWARE INVENTORY

Extending the hardware inventory for the IPU involves modifying the configuration.mof file which is a site-wide file in ConfigMgr. Good news is that these days ConfigMgr automatically creates a backup every time you make an edit, so it is easy to restore if you accidentally make a typo. The backups are stored in the <ConfigMgrInstallDir>\data\hinvarchive.

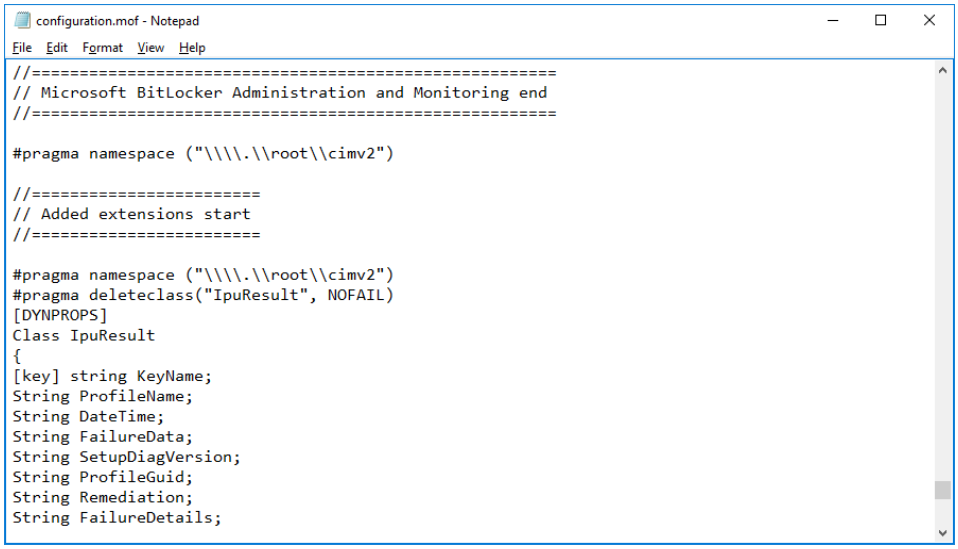
1. From the extracted IPU Installer archive, open the HW-Inventory\Configuration.mof file.



2. Copy all the content, and then open the <ConfigMgrInstallDir>\Inboxes\clifiles.src\hinv\Configuration.mof file.

3. Locate the below section in the end of the file and paste the copied content between the start / end comments.

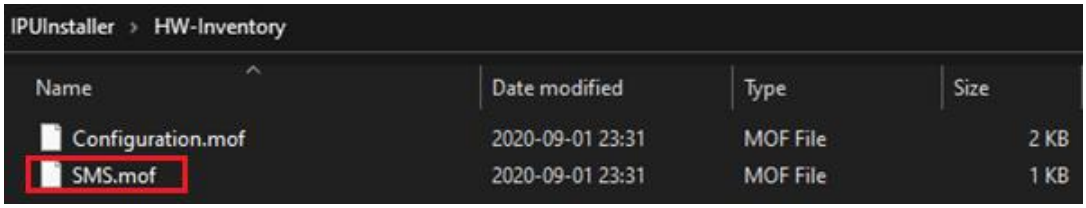
```
//=========================
// Added extensions start
//=========================

Here goes the content of the provided file (configuration.mof).

//=========================
// Added extensions end
//=========================
```



The updated Configuration.mof file.

4. Next step is to import the <<**SMS.mof>>** to the <<**Default Client Settings**>>. Navigate to the Hardware tab, click <<Set Classes>> and Import the file.

5. In the **Hardware Inventory** node, click **Set Classes**. Then click **Import**, browse to the **HW-Inventory\SMS.mof** file, and import it with the default settings.

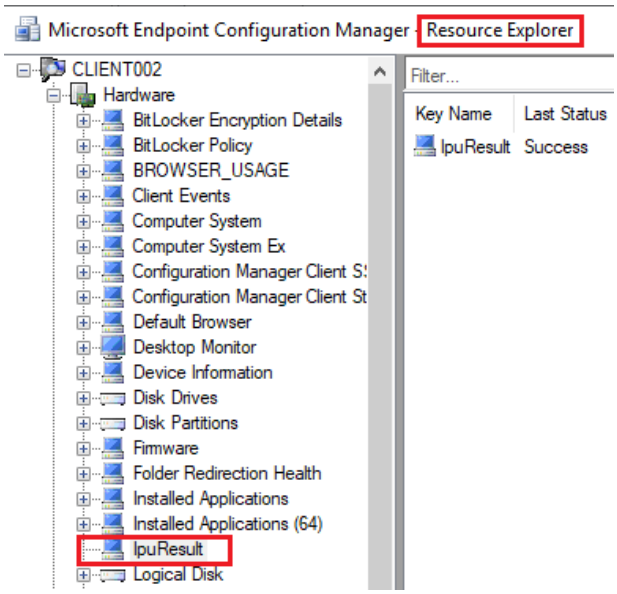


Reference: https://bit.ly/2LF8WwL

7.1. Verify extended hw inventory.

1. After extending the hardware inventory in the preceding steps, pick one of your clients, force a machine policy update, and then force a hardware inventory.

2. Once the hardware inventory is complete, open Resource Explorer and verify that you have a new node named IpuResult.



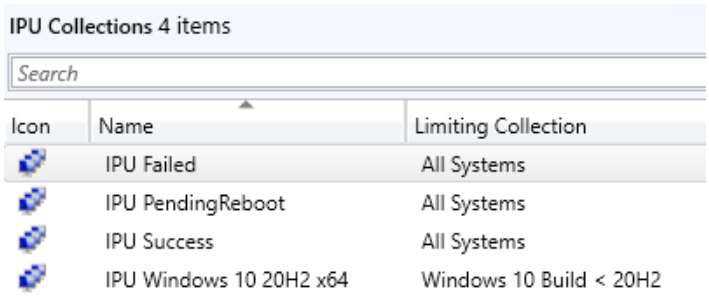
Note: <<Last Status>> will be empty until the upgrade has run.

8. COLLECTIONS

To keep track of the IPU's a few collections will have to be created and configured. We recommend using <<Incremental Update>> on these collections.

1) We will need a Collection with candidates.

2) Status Collections (top 3) and deployment Collection (bottom).



8.1. Collection specifications

8.1.1. Windows 10 Build < 20H2

Limit: All Systems

Query:

```
select
SMS_R_SYSTEM.ResourceID,SMS_R_SYSTEM.ResourceType,SMS_R_SYSTEM.Name,SMS_R_SYSTEM.
SMSUniqueIdentifier,SMS_R_SYSTEM.ResourceDomainORWorkgroup,SMS_R_SYSTEM.Client from
SMS_R_System inner join SMS_G_System_OPERATING_SYSTEM on
SMS_G_System_OPERATING_SYSTEM.ResourceId = SMS_R_System.ResourceId where
SMS_G_System_OPERATING_SYSTEM.BuildNumber < "19042" and
SMS_G_System_OPERATING_SYSTEM.Caption = "Microsoft Windows 10 Enterprise"
```

8.1.2. IPU Failed

Limit: All Systems

Query:

select SMS_R_SYSTEM.ResourceID,SMS_R_SYSTEM.ResourceType,SMS_R_SYSTEM.Name,SMS_R_SYSTEM.SMSUniqueIdentifier,SMS_R_SYSTEM.ResourceDomainORWorkgroup,SMS_R_SYSTEM.Client from SMS_R_System inner join SMS_G_System_IpuResult on SMS_G_System_IpuResult.ResourceId = SMS_R_System.ResourceId where SMS_G_System_IpuResult.LastStatus = "Failure"

8.1.3. IPU PendingReboot

Limit: All Systems

Query:

select SMS_R_SYSTEM.ResourceID,SMS_R_SYSTEM.ResourceType,SMS_R_SYSTEM.Name,SMS_R_SYSTEM.SMSUniqueIdentifier,SMS_R_SYSTEM.ResourceDomainORWorkgroup,SMS_R_SYSTEM.Client from SMS_R_System inner join SMS_G_System_IpuResult on SMS_G_System_IpuResult.ResourceId = SMS_R_System.ResourceId where SMS_G_System_IpuResult.LastStatus = "PendingReboot"

8.1.4. IPU Success

Limit: All Systems

Query:

select SMS_R_SYSTEM.ResourceID,SMS_R_SYSTEM.ResourceType,SMS_R_SYSTEM.Name,SMS_R_SYSTEM.SMSUniqueIdentifier,SMS_R_SYSTEM.ResourceDomainORWorkgroup,SMS_R_SYSTEM.Client from SMS_R_System inner join SMS_G_System_IpuResult on SMS_G_System_IpuResult.ResourceId = SMS_R_System.ResourceId where SMS_G_System_IpuResult.LastStatus = "Success"

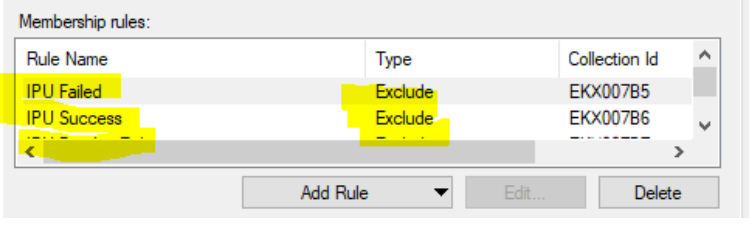
8.1.5. IPU Windows 10 20H2 x64

Limit: <<**Windows 10 Build < 20H2>>**

Query: None

Exclude collections:

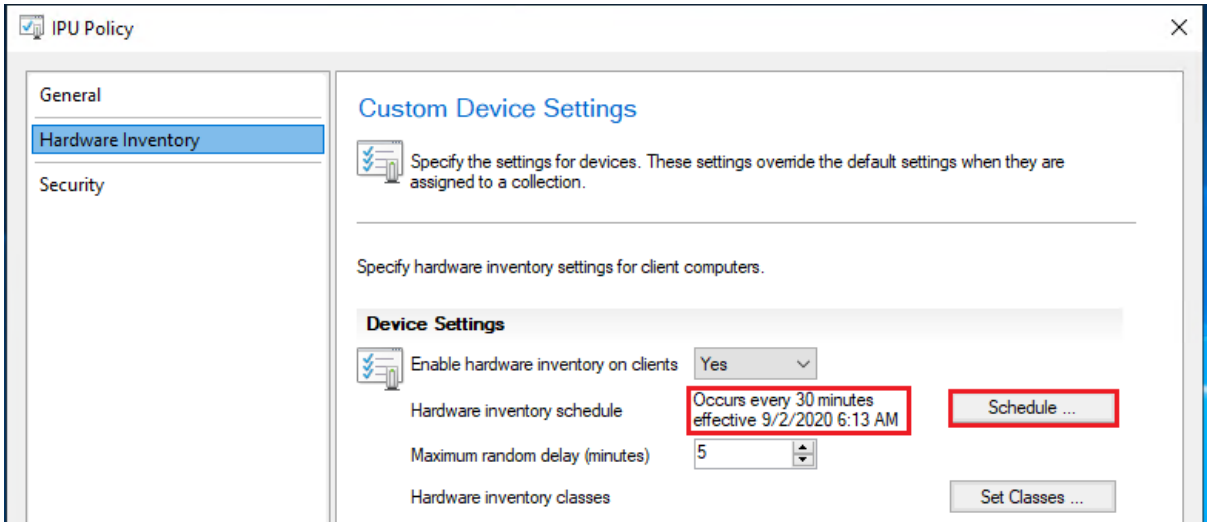
All three status collections (Failed, Success and Pending reboot):



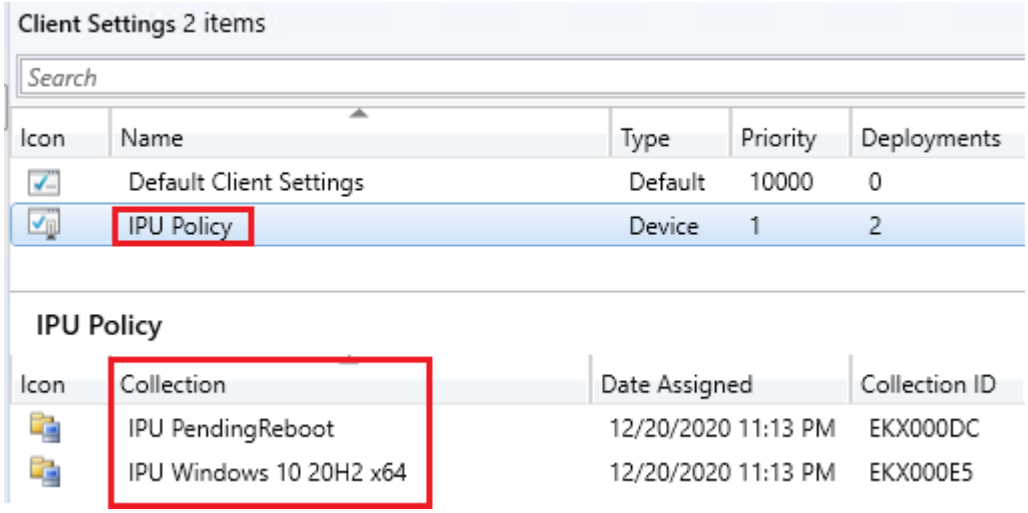
9. CUSTOM CLIENT SETTINGS

Even though it is not necessary we recommend creating a custom client setting to push hardware inventory a little bit on a couple of collection.

Navigate to the Client Settings node in the console and create a new Custom Client Device Settings.



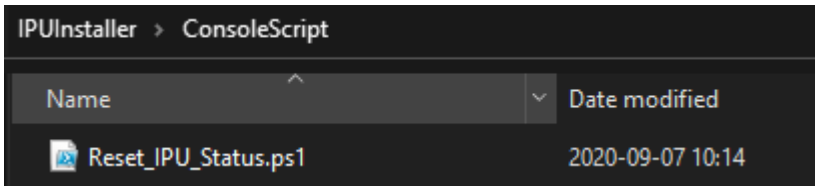
Deploy the custom settings as indicated:



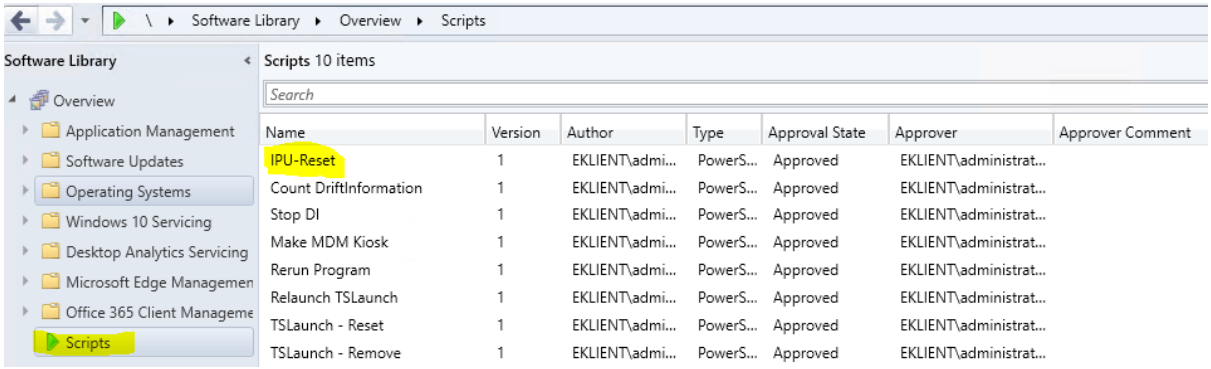
This will improve the performance of the solution by delivering faster results.

10. CONSOLE POWERSHELL SCRIPT

Locate the Reset_IPU_Status.ps1 script:



Add it to the Scripts node in the Console, ensure it gets approved by a second admin.



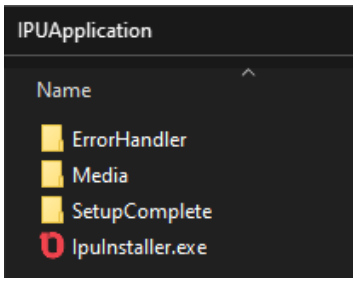
The actual purpose of this script will be clarified later on in the document, in short its used to reset a status flag in the registry of a machine that has failed to upgrade, giving it a second chance.

11. IPU APPLICATION

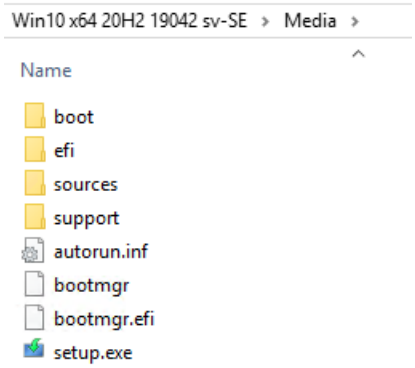
Time to create the actual Application.

11.1. Content

Locate the <<**IPUApplication**>> folder:

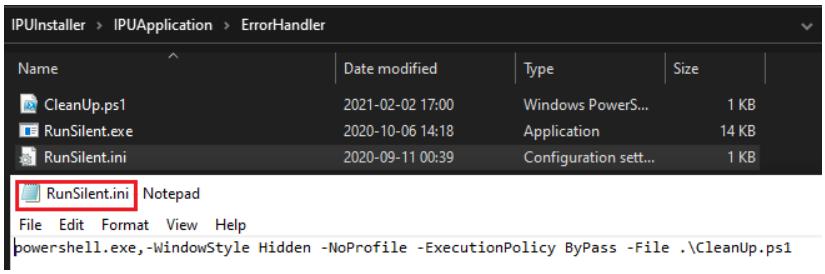


Copy the complete folder to your content share (\\server.domain.com\\Share$) and give it a more meaningful name, indicating version and language. Then copy the entire content of your media (.iso) to the <<Media>> folder:



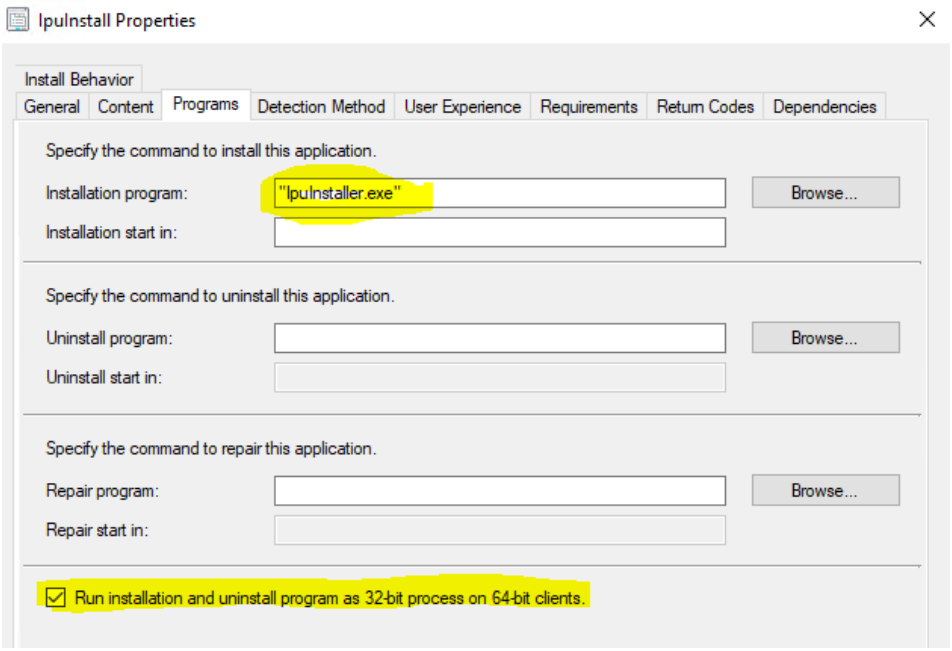
SetupComplete, ErrorHandler

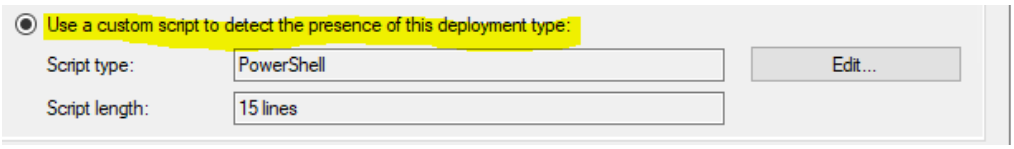
Both folders contain a utility called <<RunSilent.exe>> which basically is a command line wrapper for executing multiple commands sequentially. Put one command per line in the .ini file, separate File and Arguments with a comma, no blanks around that comma, please!



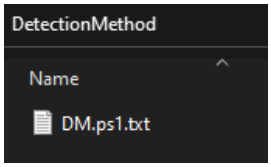
11.2. Application settings

Continue by creating a ConfigMgr Application (manual/scripted), we will focus on all non-default settings, a complete walkthrough of all dialogs raised by the wizard would simply fill too many pages.





The custom script is in the <<**DetectionMethod**>> folder.

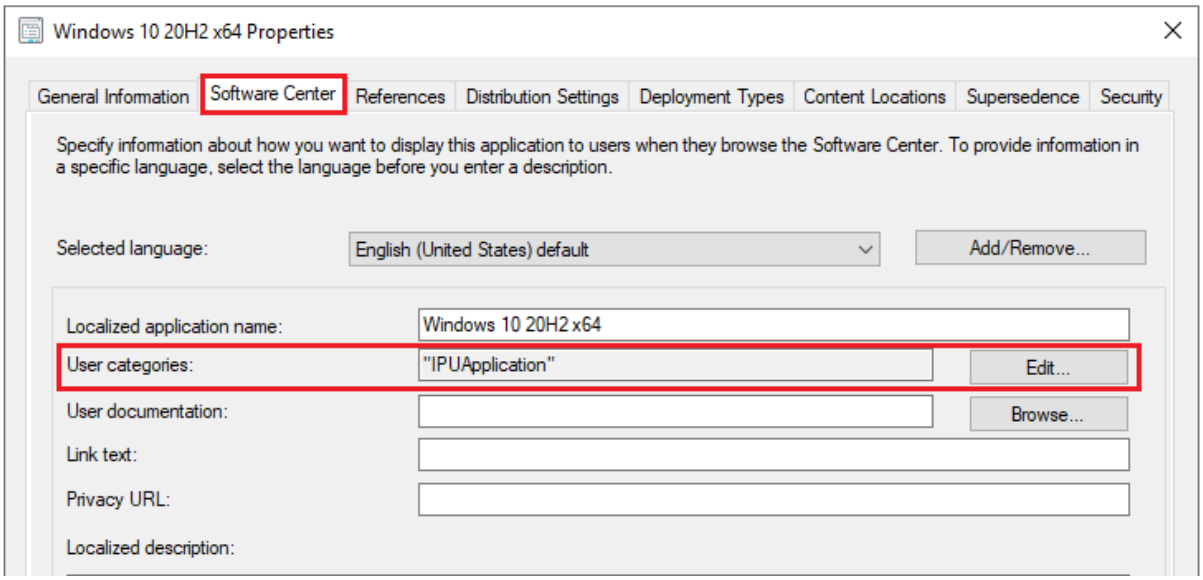


Remark: The detection script is not signed; consequently, the client policy setting in effect must be configured with Execution Policy <<**Bypass**>>.

The first line in the script contains the build-no of the new OS, change as necessary – this manual is all about 20H2, in which case 19042 is correct.

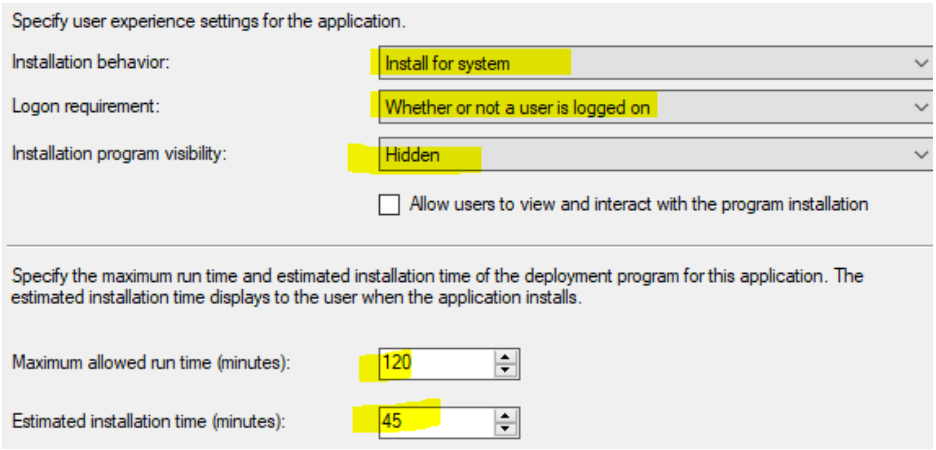
```
$BuildNumber = "19042"
```

On the **Software Center Tab,** add User Category (you will have to create it) <<**IPUApplication**>>



The category is used by the Deployment Scheduler, that we will invoke later in this document.

Install silent for System, the estimated install time is approximately right.



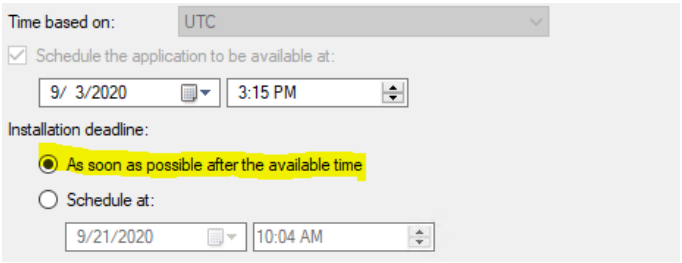
Remember to distribute the Application.

12. DEPLOYMENT SIMPLE

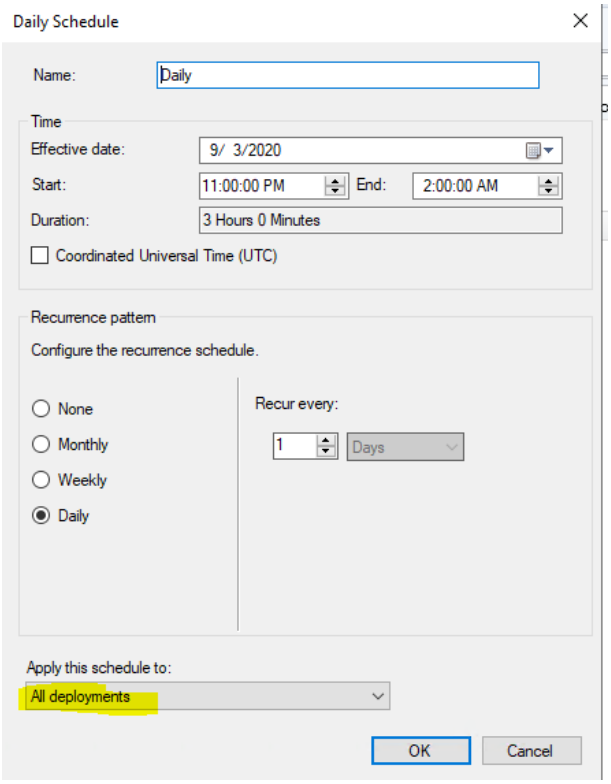
In this scenario we are not using any additional software or tools to schedule or otherwise control the upgrade.

Deploy the Application as **Required** to the Collection name <<**IPU Windows 10 20H2 x64**>>

Instruct ConfigMgr to install it as soon as possible.



As you can understand this does not give much control over the deployment and when it is run. To improve the situation, we recommend applying a <<Maintenance Window>> to the Collection.



Remark: <<All deployments>> means exactly that, this will halt all deployments until the time indicated in the schedule is reached. It is possible to overrule this on a per application / update basis.

Nothing else must be done; put a candite **computer in** the <<**IPU Windows 10 20H2 x64**>> Collection, once the maintenance window occurs, **it will be upgraded**.

13. DEPLOYMENT ADVANCED

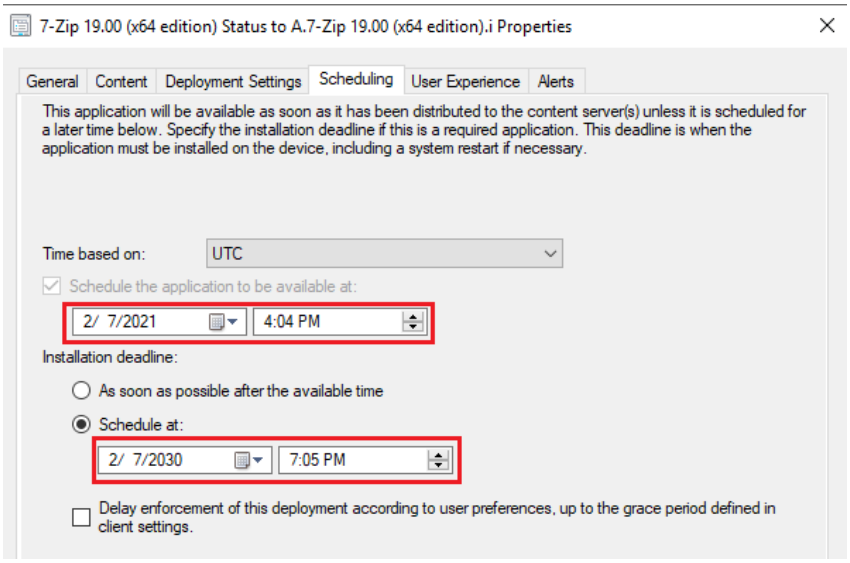
So far, we have only used built-in functionality, apart from the <<IPUInstaller.exe>> that only wraps standard commands and functionality. To gain better control we need to add a new <<Tool>> to the equation – The **Onevinn Deployment Scheduler**.

13.1. Deployment Scheduler description

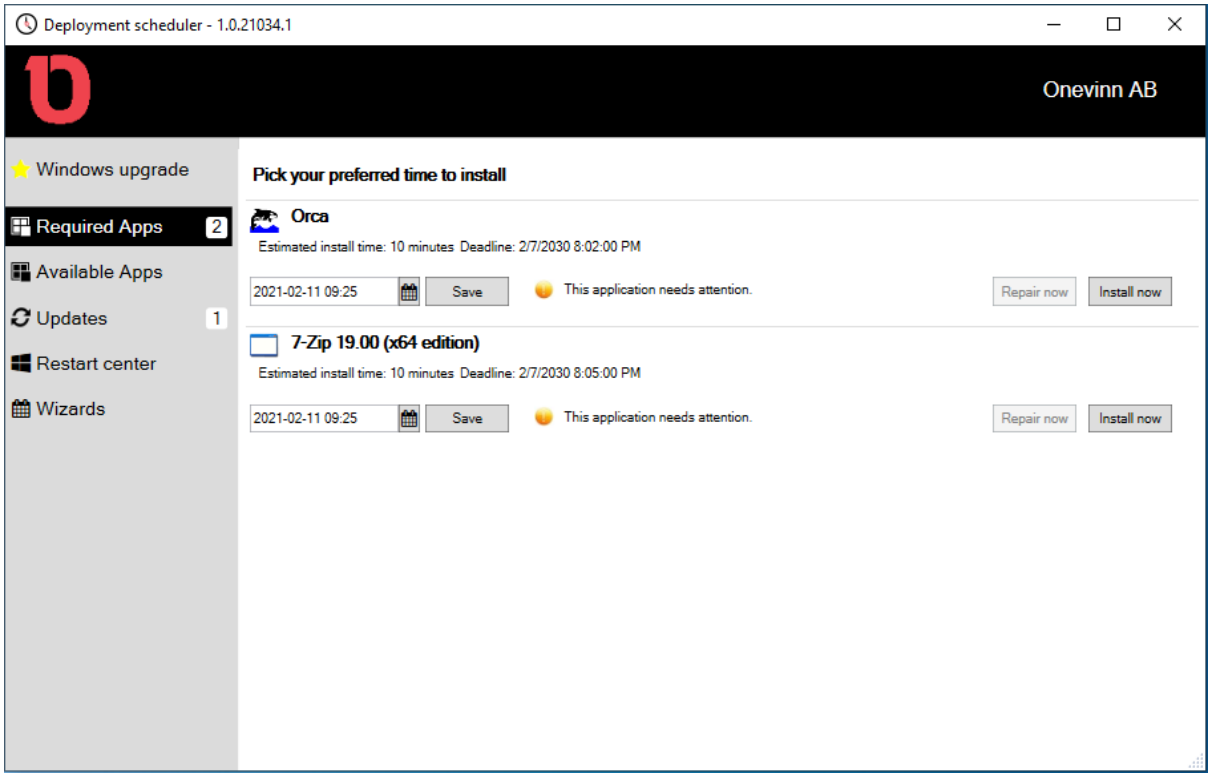
This tool is designed to allow end users to freely schedule installations and updates. It currently supports Applications, Software Updates and <<IPUApplications>> (as described in this document). It also offers advanced reboot control with file operations exclusions and scheduling.

- It has the same look and feel as Software Center – the end user will feel <<At home>>.
- All Configuration is done from a CM Console extension per Collection.
- It supports running / scheduling a Maintenance cycle when it installs all required / pending applications and updates, during such a cycle it will reboot, as necessary.
- **The Deployment Scheduler requires DotNet 4.7.2 or higher**.

To engage in an Application or Software Update deployment the deployments scheduling must allow for some elasticity. There must be a significant gap between start time and deadline and the application must also be deployed as **REQUIRED**. In this rather brutal example, the installation can be scheduled freely for ~9 years.

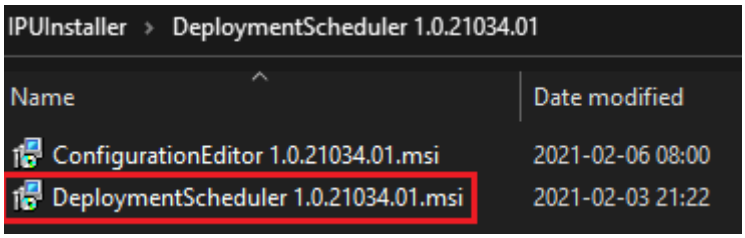


Appearance



Installation

Locate the installer.



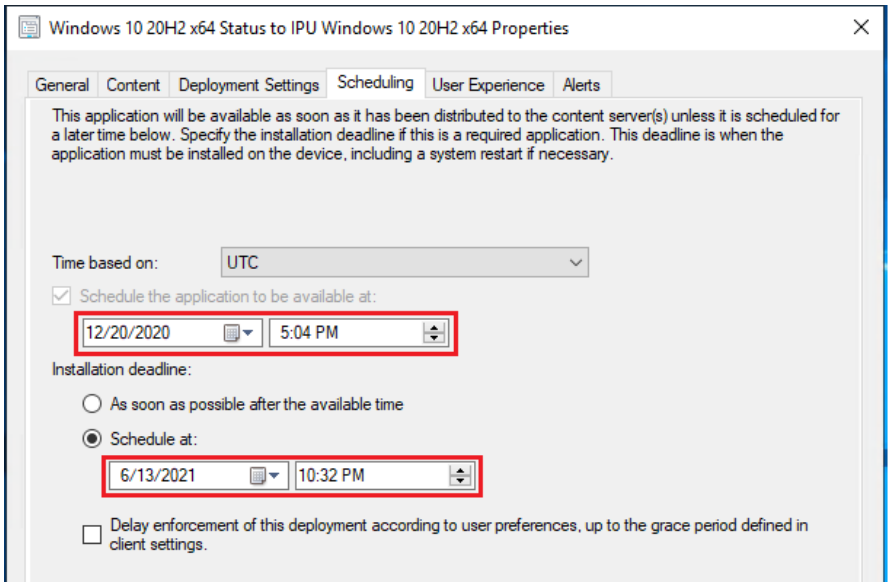
Deploy like any other msi using the silent switch:

msiexec /i “DeploymentScheduler 1.0.21034.01.msi” /qn

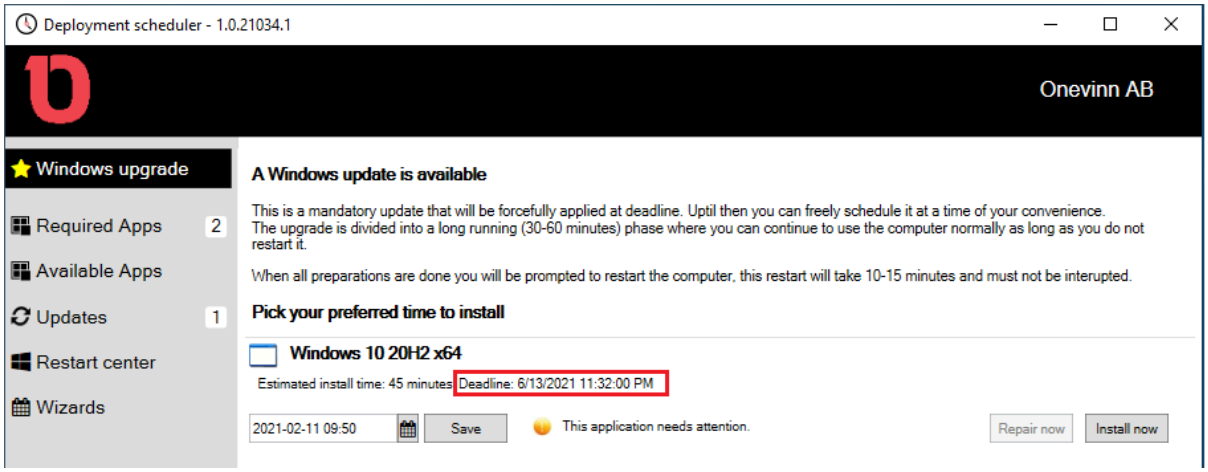
13.2. IPUApplication deployment

If you have adhered to the instructions above the only modification necessary is to adjust the <<IPUApplication>> scheduling, we need to create a <<Time Gap>> by moving the **deadline into the future**.

In this example we allow the end user until June 13th, 2020 to at will schedule and run the upgrade to Windows 10 20H2.



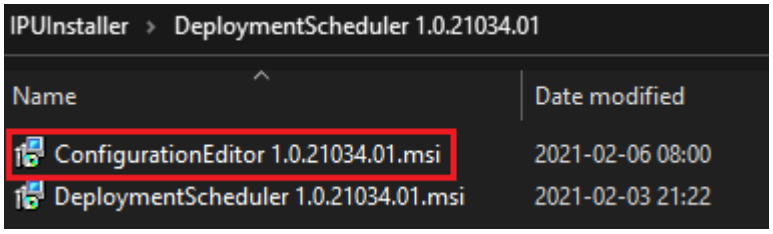
This will be reflected in the Deployment Scheduler as shown here:



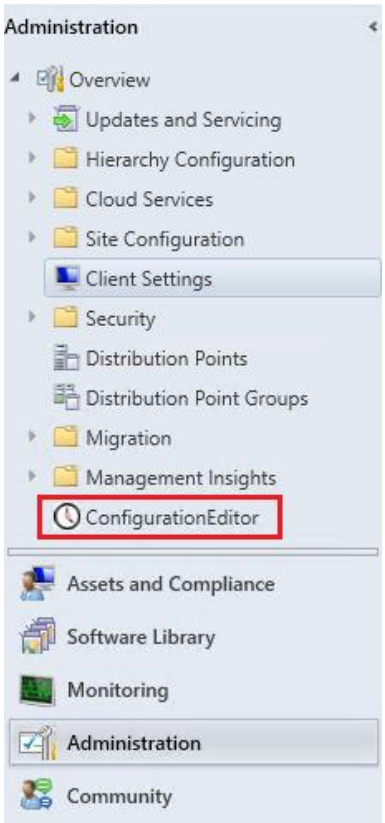
13.3. Configuration Deployment Scheduler

Requires DotNet 4.7.2 or higher

Install the ConfigurationEditor CM Console extension on site server or any computer with the CM Console previously installed.

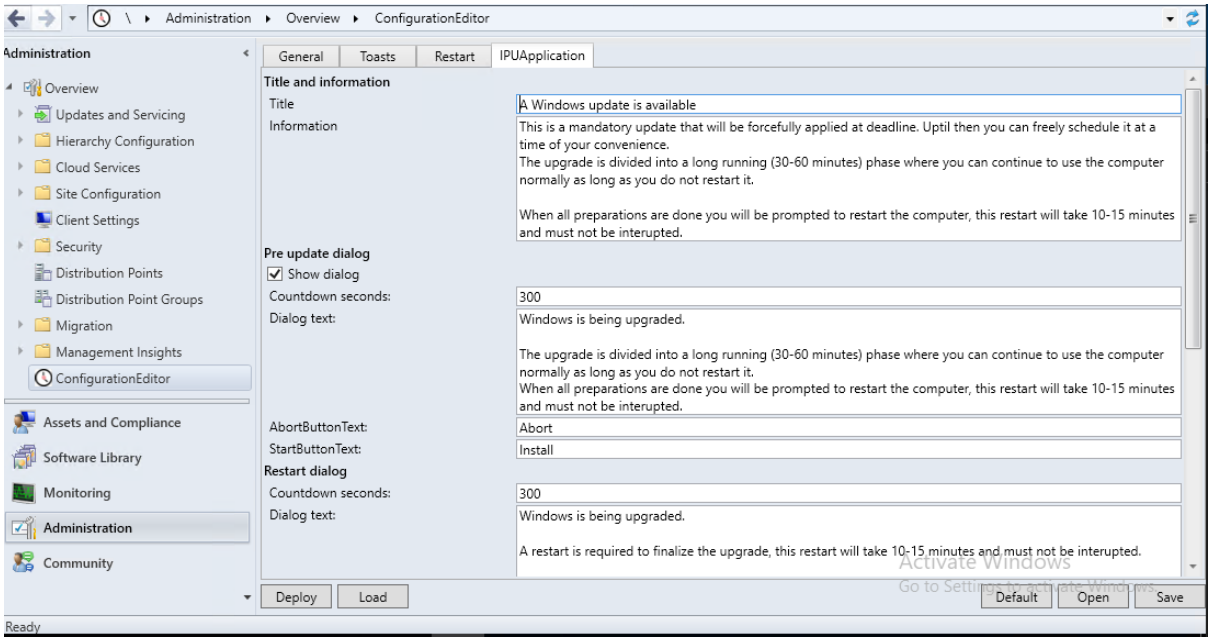


Next time you open the console you will find a new node on the Administration tab:



Settings

This document focusses on IPU, there will be a complete manual for the Deployment Scheduler available soon, possibly also a short film. But this should get you going (perhaps after some trial and error)



Button	Function
Deploy	Deploy all settings to a Collection
Load	Read back all settings from a Collection
Default	Load default settings, us-EN only so far
Open	Open settings from file.
Save	Save settings to file - Backup.

When settings are deployed to a Collection it will require a machine policy request before the Client adheres to them. The Deployment Scheduler itself checks in wmi (on client) every 30 minutes, so it can take a while.

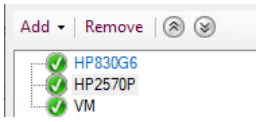
14. DRIVERS - STANDARD

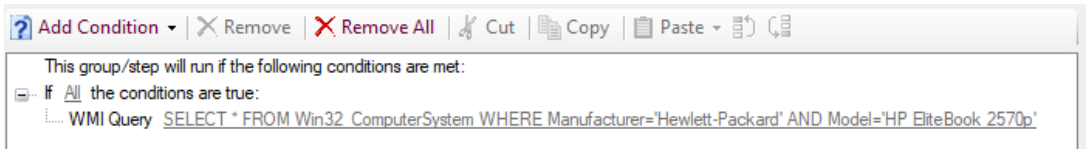
The <<IPUInstaller>> will add drivers to the auto generated SetupConfig.ini file if present in the following location when the upgrade starts:

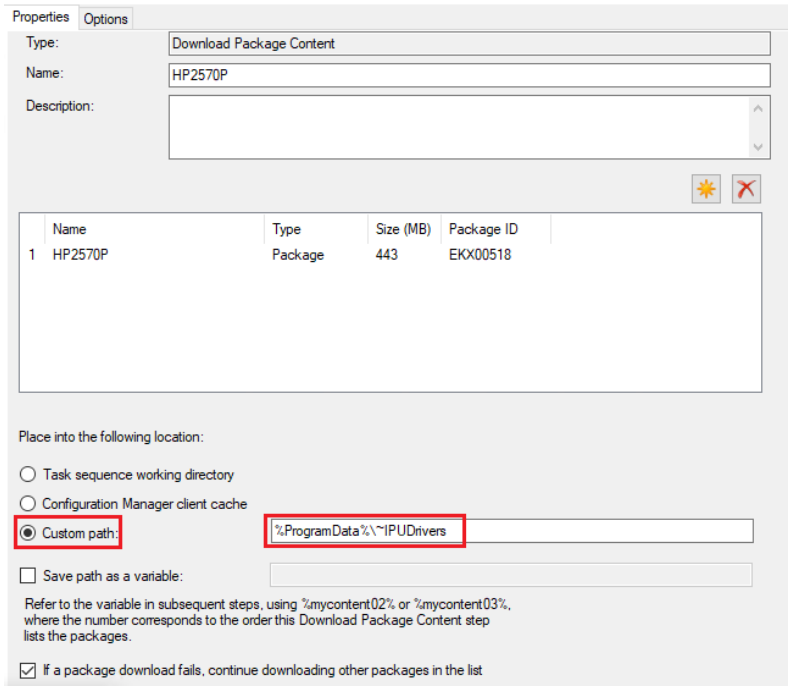
%ProgramData%\~IPUDrivers

One way is to create a Driver Task Sequence and execute it in advance of the IPU Application. This is actually very simple.

Create a custom Task Sequence, add one “Download Package Content” per model with appropriate Wmi query, for example (‘HP EliteBook 2570p’):







This way pre-existing packages can be re-used. When the <<IPUInstaller>> starts it will detect the driver folder and invoke them in the IPU.

15. DRIVERS – ADVANCED

It is possible to use a compressed driver package as an alternative to re-using pre-existing packages.

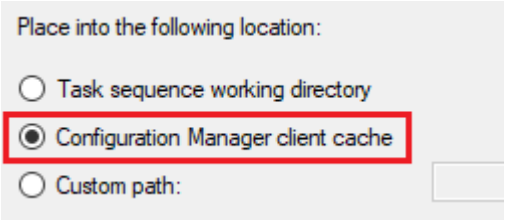
<<IPUInstaller>> can expand drivers compressed as a .wim file. We will not dive in to how to create the wim but you can use several tools available on the internet or dism.exe

Dism /Capture-Image /ImageFile:"<TargetLocation>\IPUDrivers.wim" /CaptureDir:"<SourceFolder>" /Name:IPUDrivers

The key here is that the resulting file **must have the exact name** <<**IPUDrivers.wim**>>

Create a package from the .wim file and add it to the custom TS exactly as you would add a standard package.

The only other difference is that you must choose to download to CCMCACHE.



The <<IPUInstaller>> will with this approach sniff the ccmcache for files named <<**IPUDrivers.wim**>>, if more than one is found (possible update) it will pick the one will the latest file date, expand it to %ProgramData%\~IPUDrivers and add the path to the SetupConfig.ini file.

16. PROCESS

1. A candidate computer is put in the <<**IPU Windows 10 20H2 x64**>> collection.
2. The computer attempts the upgrade, scheduled or not.
3. If the upgrade succeeds, the hardware inventory in combination with our collection queries will move it to the succeeded collection.
4. In case the upgrade fails, the same process will instead move the computer to the failed collection. At this point we can examine the extended <<**IPUStatus**>> node in Resource Explorer, this will not always be enough, but will in many cases point us in the right direction. Once the cause of failure, be it an outdated driver or worse, has been remediated, we can reset the failure flag on the machine by running the <<**IPU-Reset**>> PowerShell script remotely on it. This will result in the hw inventory and collection queries moving the computer back to the deployment collection for a second attempt.
5. Ultimately all computers put in the deployment collection will magically migrate to the succeeded collection.

17. ISSUES

Non so far 😊

18. FINALLY

18.1. Support us supporting you.

All tools published are free to use, some of them could however be further developed and better maintained, this is not always an easy task, we must always focus on paying customers which means all work is done on spare or private hours.

Based on this we expect well doing, commercially driven companies to sign up for a support deal if you decide to use a tool like the Deployment Scheduler; this will not only grant you faster and dedicated support it will also allow us to spend more time maintaining these tools, something we all can benefit from.

That said, the Tools are free, support is not.

Please contact us for an offer at: [About us (onevinn.com)]

18.2. Code

If you have made it this far, you might be interested in checking out the code behind the DeploymentScheduler:

Josch62/DeploymentScheduler (github.com)