Contents

[Prerequisites 2](#_Toc185511948)

[Windows Media 3](#_Toc185511949)

[Recommendation on media type to use 3](#_Toc185511950)

[Obtaining media via the script (New June 2023) 3](#_Toc185511951)

[Downloading media yourself 4](#_Toc185511952)

[Configuration 4](#_Toc185511953)

[Adding Applications and M365 Apps 4](#_Toc185511954)

[Adding 3rd party apps 4](#_Toc185511955)

[Customizing M365 Apps 5](#_Toc185511956)

[Drivers 6](#_Toc185511957)

[Automatically adding drivers (New June 2024 – Release v2406.1) 6](#_Toc185511958)

[Microsoft 6](#_Toc185511959)

[Dell 8](#_Toc185511960)

[Lenovo 9](#_Toc185511961)

[HP 10](#_Toc185511962)

[Manually adding drivers 13](#_Toc185511963)

[Preparing the command line 13](#_Toc185511964)

[Script parameters 13](#_Toc185511965)

[Preparing a Configuration File (New in 2412.1) 19](#_Toc185511966)

[Creating your own Coniguration Json file 19](#_Toc185511967)

[Command Lines 20](#_Toc185511968)

[Running the Script 21](#_Toc185511969)

[Customizing the USB drive (optional) 23](#_Toc185511970)

[Driver Support (optional but recommended) 23](#_Toc185511971)

[Having DISM inject drivers on the fly (recommended if managing multiple models) 24](#_Toc185511972)

[How to inject drivers into FFU 24](#_Toc185511973)

[Device Naming (optional but recommended) 25](#_Toc185511974)

[Prompt to enter name 25](#_Toc185511975)

[Specifying Multiple Name Prefixes 26](#_Toc185511976)

[Device Enrollment 27](#_Toc185511977)

[Optional - Copy provisioning package to the USB drive 28](#_Toc185511978)

[Optional - Using Autopilot for Existing Devices 29](#_Toc185511979)

[Create User Driven Autopilot Profile 29](#_Toc185511980)

[Install Azure AD, WindowsAutopilotIntune, Intune Graph Powershell modules 31](#_Toc185511981)

[Get the User Driven Autopilot profile and convert to JSON 32](#_Toc185511982)

[Multiple Autopilot Configuration Files 32](#_Toc185511983)

[Optional - Copying a custom WinRE.wim 32](#_Toc185511984)

[Optional – Adding Multiple Components using unattend.xml 34](#_Toc185511985)

[Troubleshooting 34](#_Toc185511986)

[Deployment 34](#_Toc185511987)

[Creation 35](#_Toc185511988)

[Need help? 35](#_Toc185511989)

[Windows Language Support 35](#_Toc185511990)

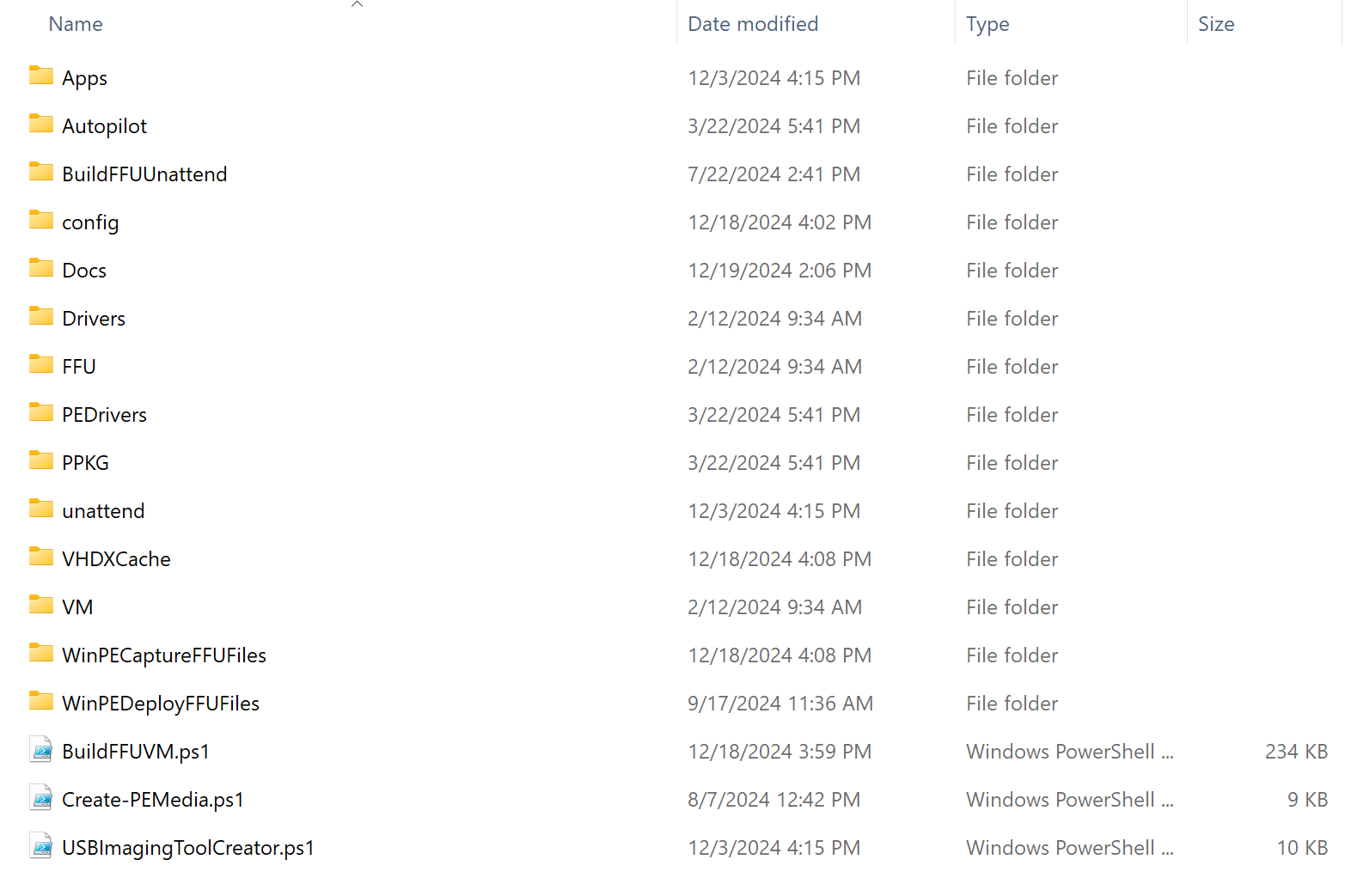
# Prerequisites

1. Enable Hyper-V – this can be done on either Windows 10/11 or Windows Server  
     
   [Enable Hyper-V on Windows 10 | Microsoft Learn](https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v)  
   [Get started with Hyper-V on Windows Server | Microsoft Learn](https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/get-started/get-started-with-hyper-v-on-windows)
2. Once Hyper-V has been enabled and you have rebooted, create either an external or internal switch. An external switch is preferred, but an internal switch can be used.   
     
   Once created, you’ll want to get the IP address of the switch using ipconfig /all . The switch name will be used in the -VMSwitchName parameter (e.g. -VMSwitchName ‘external’) and the IP address will be used in the -VMHostIPAddress (e.g. -VMHostIPAddress 192.168.1.158).

If using an internal switch, the switch IP address won’t be that of the VM Host physical machine. It could be a link-local IP address like 169.254.x.x. IPconfig /all will tell you.

While this could easily be automated, it’s a design decision to not automate this process due to the difficulty of picking the right VMSwitch and IP address.

1. Extract the ZIP file from the Github repo to C:\FFUDevelopment. If you must use another location, you’ll need to specify the -FFUDevelopmentPath parameter when running the PowerShell script to the new location (e.g. BuildFFUVM.ps1 -FFUDevelopment ‘D:\FFUDevelopment’)

The folder structure is as follows  
  


# Windows Media

## Recommendation on media type to use

Windows media comes in two types: business or consumer.

Windows media can be obtained from a few different sources – Volume Licensing Service Center (VLSC – now available at admin.microsoft.com), Microsoft Visual Studio Downloads, or the Windows Media Creation Tool.

The BuildFFUVM.ps1 script will allow you to pass whichever type of media you want from whatever source you want using the -ISOPath parameter; however, **it’s recommended that you use consumer media**, not business/VL. This is because Subscription Activation will fail if the media is mismatched from the key in the firmware.

If you plan on using a MAK or KMS to activate, you can use media from VLSC, but if you expect the device to activate automatically and upgrade to Enterprise or Education SKUs via Subscription Activation, you must use consumer media. To use a MAK/KMS key to activate, you must provide a the -ProductKey XXXXX-XXXXX-XXXXX-XXXXX-XXXXX parameter.

## Obtaining media via the script (New June 2023)

By default, the script will download the Windows 11 English x64 media from the media creation tool. If you would like to download Windows 10, x86, or another language, you can use the -WindowsRelease, -WindowsArch, -WindowsLanguage, -MediaType parameters to download different media.

## Downloading media yourself

If you would like to download media yourself, you can pass the -ISOPath parameter to tell the script the location of the ISO you have downloaded.

Windows consumer media – download the Windows ISO from Visual Studio downloads, or the Windows Media Creation Tool. If you don’t have access to Visual Studio downloads, download the Windows Media Creation Tool from either of these links:  
  
[Windows Media Creation Tool for Windows 10](https://www.microsoft.com/en-us/software-download/windows10)

[Windows Media Creation Tool for Windows 11](https://www.microsoft.com/en-ca/software-download/windows11)  
  
Visual Studio downloads seems to have more up to date media (updates every 3rd Monday/Tuesday) while the Media Creation Tool is behind.

# Adding Applications and M365 Apps

The FFUDevelopment\Apps folder contains the following files:

* Office folder
  + DeployFFU.xml
  + DownloadFFU.xml
* InstallAppsandSysprep.cmd
* AppList\_InboxAppsSample.json
* AppList\_Sample.json

The Office folder contains a DeployFFU.xml and DownloadFFU.xml file that are used by the Office Deployment Toolkit to handle the download and deployment of Office.

The InstallAppsandSysprep.cmd file is a command file that will run during audit mode after Windows is installed to the VM which will run whatever commands are in that file.

The AppList files are sample JSON files used with winget, a package manager for Windows. By using the command `winget search ` in the command line, you can find the Name, ID, and source information for the applications you want to include in your FFU. Once you have this information, you can add it to an AppList.json file, which will then download and install the specified applications to the VM.

## Adding 3rd party apps

If you want to add additional applications to the FFU, create a new folder in the Apps folder and put the installation files in that folder.

For example, if you want to include Chrome, you would create a Chrome folder in c:\FFUDevelopment\Apps and copy in the GoogleChromeStandaloneEnterprise64.msi  
  
Graphical user interface, text, application

Description automatically generated

Next, modify the InstallAppsandSysprep.cmd file and include the silent command line to install Chrome. It’s recommended that for each app or command that you add to the file that you put a **REM App Name (e.g. REM Google Chrome)** on the line before the command so it’s easy for you to remember what each command is doing.

**Do not remove the commands that delete the unattend.xml files or the sysprep related commands.**

Graphical user interface, text, application

Description automatically generated

## Customizing M365 Apps

If the -InstallOffice and -InstallApps parameters are both set to $true, the script will download M365 Apps/Office Deployment Toolkit and download M365 Apps to c:\FFUDevelopmentPath\Apps\Office.

It shouldn’t be necessary to modify the DownloadFFU.xml file in the Office folder.

If you want to modify which Office apps, client edition, channel, or any other properties, you’ll want to modify the **DeployFFU.xml** file.

The DeployFFU.xml file will install Word, Excel, and PowerPoint. Outlook and Teams are excluded. If you want to include Outlook and/or Teams, comment out the lines in the DeployFFU.xml file and save it.

<Configuration ID="efa6df21-a106-428e-8eaa-d89a5dda6030">

  <Add OfficeClientEdition="64" Channel="Current">

    <Product ID="O365ProPlusRetail">

      <Language ID="MatchOS" />

      <ExcludeApp ID="Access" />

      <ExcludeApp ID="Lync" />

      <ExcludeApp ID="Publisher" />

      <ExcludeApp ID="Bing" />

      <ExcludeApp ID="Teams" />

      <ExcludeApp ID="Outlook" />

    </Product>

  </Add>

  <Property Name="SharedComputerLicensing" Value="0" />

  <Property Name="FORCEAPPSHUTDOWN" Value="FALSE" />

  <Property Name="DeviceBasedLicensing" Value="0" />

  <Property Name="SCLCacheOverride" Value="0" />

  <Updates Enabled="TRUE" />

  <Display Level="None" AcceptEULA="TRUE" />

</Configuration>

If you have an XML that you normally use for Office, it’s suggested to replace DeployFFU.xml. In other words, delete the existing DeployFFU.xml file, copy your XML file to the c:\FFUDevelopment\Apps\Office directory, and rename it to DeployFFU.xml.

If you want to change the name of the DeployFFU.xml file for whatever reason, you’ll also need to change the path in the InstallAppsandSysprep.cmd file as well as the BuildFFUVM.ps1 file.

## Adding Winget apps

The script will check for the presence of the Winget PowerShell module and if not found, will install it.

In the C:\FFUDevelopment\Apps folder, if an AppList.json file is found, the script will attempt to install apps that have been downloaded via Winget to the Apps folder. The AppList\_Sample.json file contains the following apps:

{

    "apps": [

        {

            "name": "Company Portal",

            "id": "9WZDNCRFJ3PZ",

            "source": "msstore"

        },

        {

            "name": "Microsoft Teams",

            "id": "Microsoft.Teams",

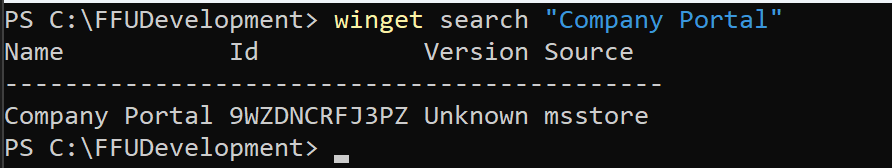
            "source": "winget"

        }

    ]

}

The Company Portal and Microsoft Teams apps show two different applications from different sources. Using **Winget search “Company Portal”** we can see the Name, ID, and source information about the Company Portal app that we can use to construct the json file.

****

# Drivers

## Automatically adding drivers (New June 2024 – Release v2406.1)

The script can now automatically download drivers from the following OEMs and either add them to the FFU, or copy them to a Drivers folder on your USB drive’s deploy partition to use at deployment time:

* Microsoft
* Dell
* Lenovo
* HP

To automatically download drivers, you need to specify the -Make and -Model parameters and either the -InstallDrivers or -CopyDrivers parameters. For more information about these parameters, see the [Script parameters](#_Script_parameters) section.

The script will only download the drivers for a single model. If you need multiple models, you’ll be better off copying the drivers to the USB drive manually. See [Having DISM inject drivers on the fly (recommended if managing multiple models)](#_Having_DISM_inject) for more info.

The script attempts to download drivers early in the build process since it’s likely that you’ll need to provide some additional information.

Each OEM has a different way of how we go about getting drivers and what the user experience is like.

### Microsoft

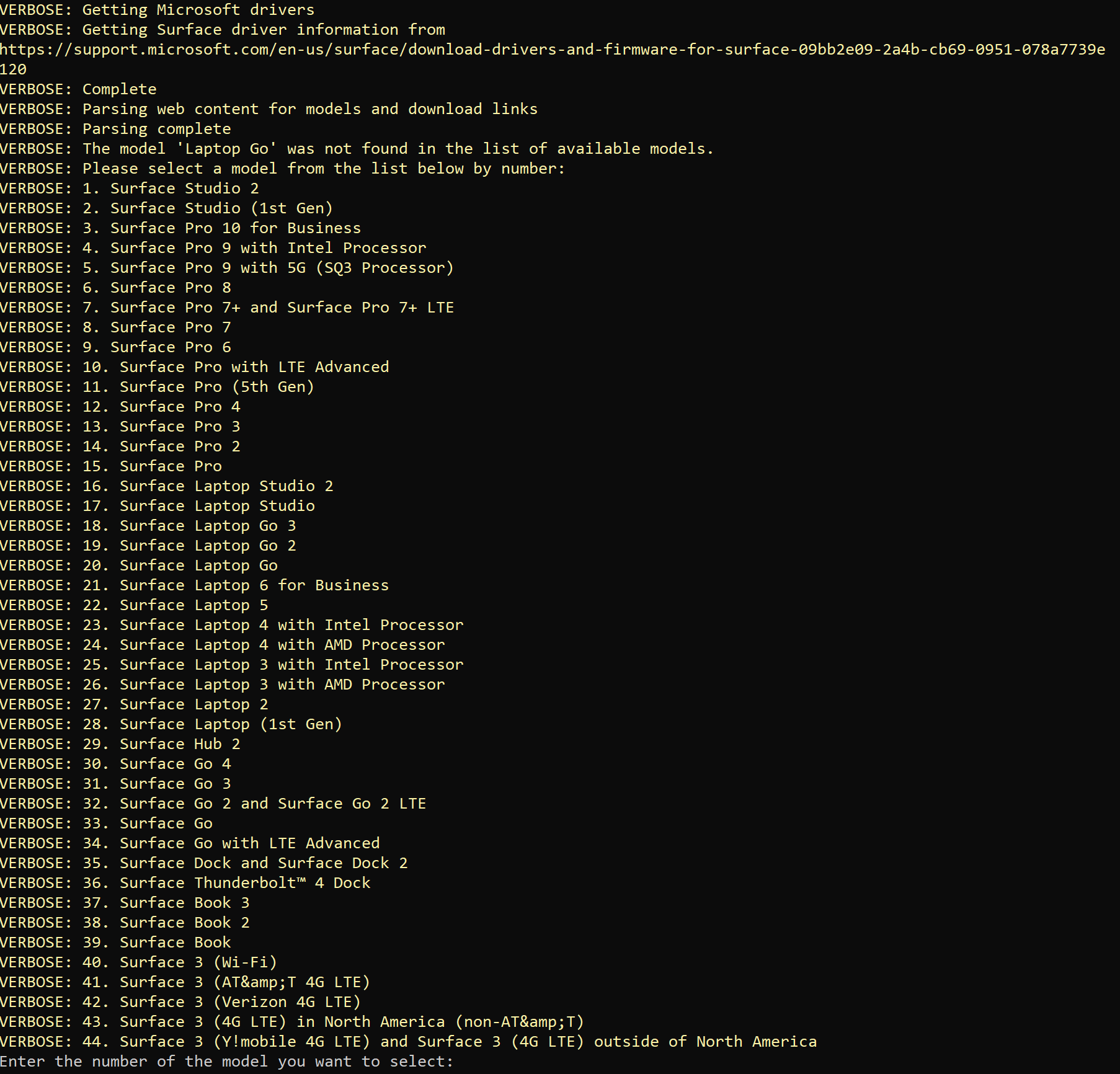
For Microsoft Surface, the driver files are hosted on the Microsoft download center. The script will scrape and parse the [Download Surface Drivers and Firmware](https://support.microsoft.com/en-us/surface/download-drivers-and-firmware-for-surface-09bb2e09-2a4b-cb69-0951-078a7739e120) page to get the latest list of Surface devices.

The script uses the Get-MicrosoftDrivers function which accepts -Make, -Model, and -WindowsRelease parameters. Make and Model are both string parameters and WindowsRelease is an integer parameter. If the model parameter doesn’t contain an exact match of a known Surface model, it’ll give you a list of Surface model’s to pick from.

The following command line says that we want to download the drivers for a Microsoft Laptop Go for Windows 10.

.\BuildFFUVM.ps1 **-make 'Microsoft' -model 'Laptop Go'** -WindowsSKU 'Pro' -Installapps $true -InstallOffice $true -InstallDrivers $true -VMSwitchName 'external' -VMHostIPAddress '192.168.1.158' -CreateDeploymentMedia $true -BuildUSBDrive $true -UpdateLatestCU $true -UpdateLatestNet $true -UpdateLatestDefender $true -UpdateEdge $true -UpdateOneDrive $true -verbose -RemoveFFU $true **-WindowsRelease 10**

The issue with this command line is that there isn’t a model named exactly Laptop Go. Since there aren’t that many Surface models (compared to hundreds from other OEMs), the script will give you a list of Surface models to pick from.



If you want to build an FFU for Surface Laptop Go 3, enter 18 and it’ll download the MSI and extract the drivers to the .\FFUDevelopment\Drivers\Microsoft\Surface Laptop Go 3 folder.

If you would have provided the exact model string instead of just Laptop Go (e.g. -Model ‘Surface Laptop Go 3’), the script wouldn’t prompt you to enter a valid model.

### Dell

For Dell, the script uses the [Dell CatalogPC Cab file](http://downloads.dell.com/catalog/CatalogPC.cab) which is used in Dell Support Assist and possibly other Dell tools to download drivers. The cab consists of an XML file that the script parses to search for drivers applicable for the model you wish to create a FFU for.

The script calls the Get-DellDrivers function which accepts the -Model and -WindowsArch parameters.

Unlike Microsoft Surface drivers, Dell doesn’t give a list to pick from when the -model parameter isn’t an exact match. This is due to how the CatalogPC XML file lists drivers. It treats the driver as the primary element and lists what models that driver can be installed on.

The following command line says that we want to download Dell 3190 drivers for Windows 10.

.\BuildFFUVM.ps1 **-make 'Dell' -model '3190'** -WindowsSKU 'Pro' -Installapps $true -InstallOffice $true -InstallDrivers $true -VMSwitchName 'external' -VMHostIPAddress '192.168.1.158' -CreateDeploymentMedia $true -CreateCaptureMedia $true -BuildUSBDrive $true -UpdateLatestCU $true -UpdateLatestNet $true -UpdateLatestDefender $true -UpdateEdge $true -UpdateOneDrive $true -RemoveFFU $true **-WindowsRelease 10** -Verbose

The script will find every driver that is tagged with 3190 and download the latest available version. It strips out any firmware or other non-driver file types. You may notice that it will download multiple video or audio drivers. This is due to each model having variants with different video cards or other hardware. This would make the FFU a bit larger, but not excessively so.

Below is a screenshot of what the verbose output of the script looks like when downloading the drivers for a Dell 3190.



### Lenovo

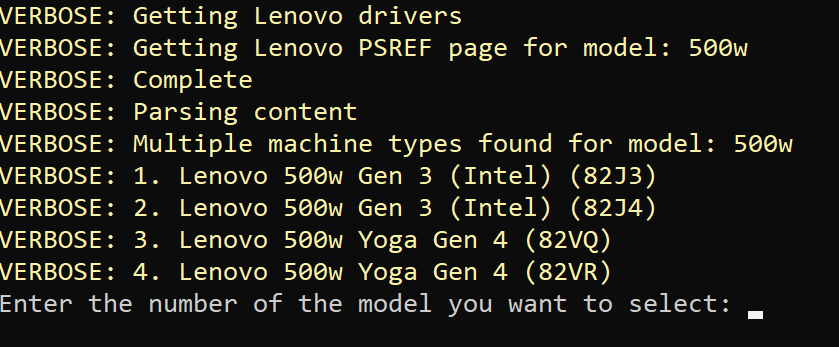
For Lenovo, the script uses the same process Lenovo System Update/Update Retriever use. It uses the Get-LenovoDrivers function which accepts -Model, -WindowsArch, -WindowsRelease parameters.

Lenovo as a company doesn’t use model like other companies do. Lenovo prefers to use a Machine Type value instead of Model number. The Machine Type value can be found on the bottom of your device as the first four characters of the MTM: value. Since most people don’t know what the machine type value is, when passing the -model parameter, you can pass either the machine type or the “friendly” model number.

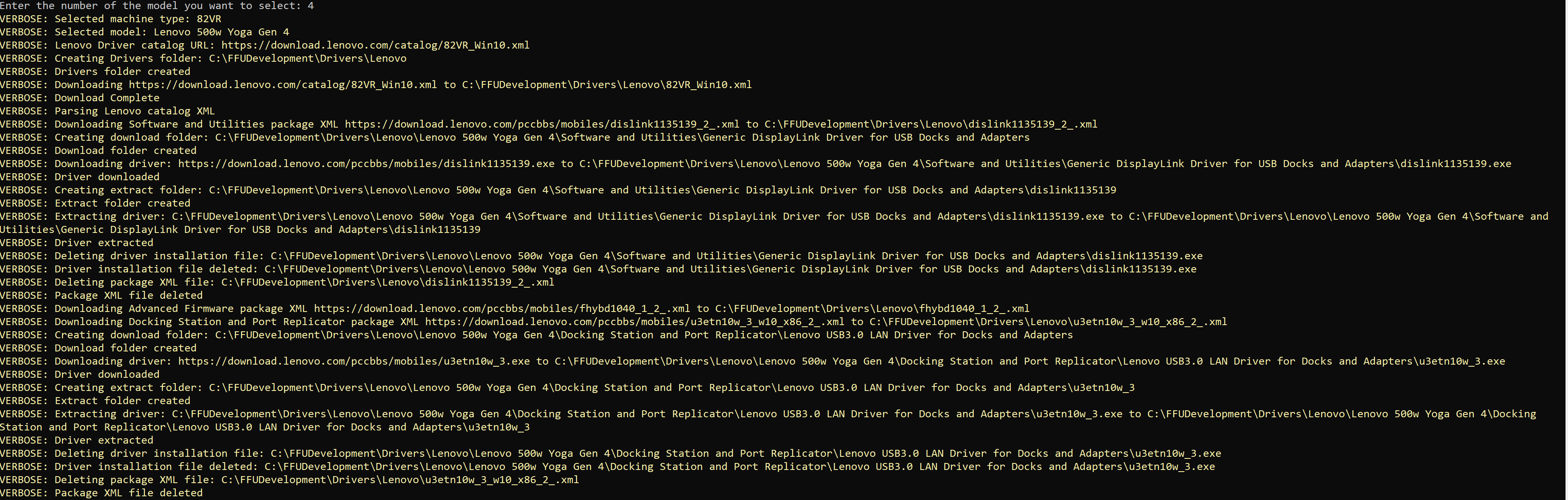
The following command line says that we want to download Lenovo 500w drivers for Windows 10.

.\BuildFFUVM.ps1 **-make 'Lenovo' -model '500w'** -WindowsSKU 'Pro' -Installapps $true -InstallOffice $true -InstallDrivers $true -VMSwitchName 'external' -VMHostIPAddress '192.168.1.158' -CreateDeploymentMedia $true -BuildUSBDrive $true -UpdateLatestCU $true -UpdateLatestNet $true -UpdateLatestDefender $true -UpdateEdge $true -UpdateOneDrive $true -RemoveFFU $true **-WindowsRelease 10** -Verbose

The script will go out to the [Lenovo PSREF](https://psref.lenovo.com/search) page to figure out the Machine Type value and if multiple Machine Types are found (there are usually multiples found for different configuration types).



The Machine Type is the value in parenthesis. On the bottom of my device, the MTM value is MTM:**82VR**ZAKXXX. I would want to pick number 4 from the list since it includes (82VR). The below screenshot shows the script downloading the appropriate drivers for a Lenovo 500w.



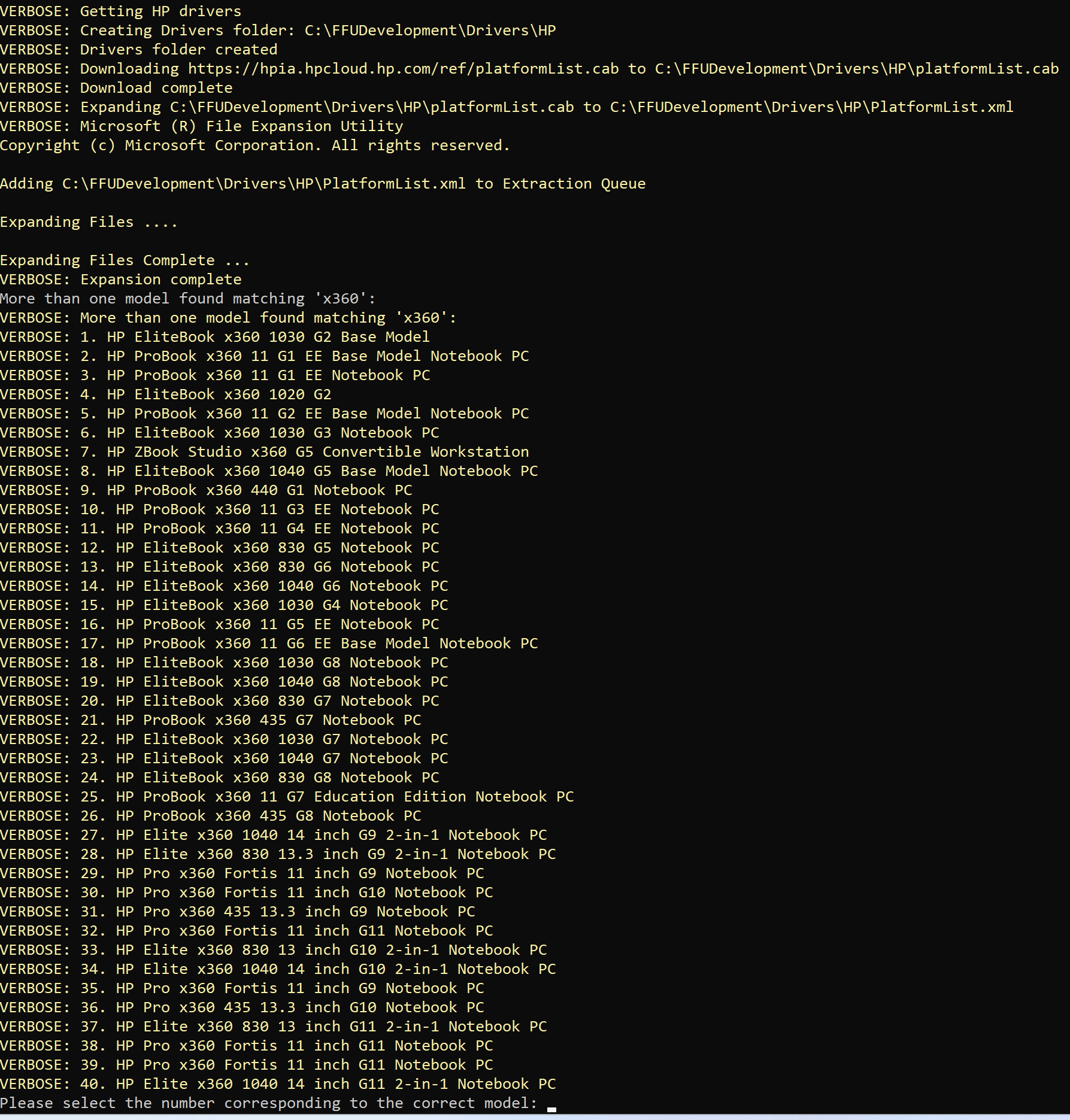
If you use the Machine Type value for the -Model parameter (e.g. -model ‘82VR’) the script will automatically download the drivers without prompting you to select the model.

### HP

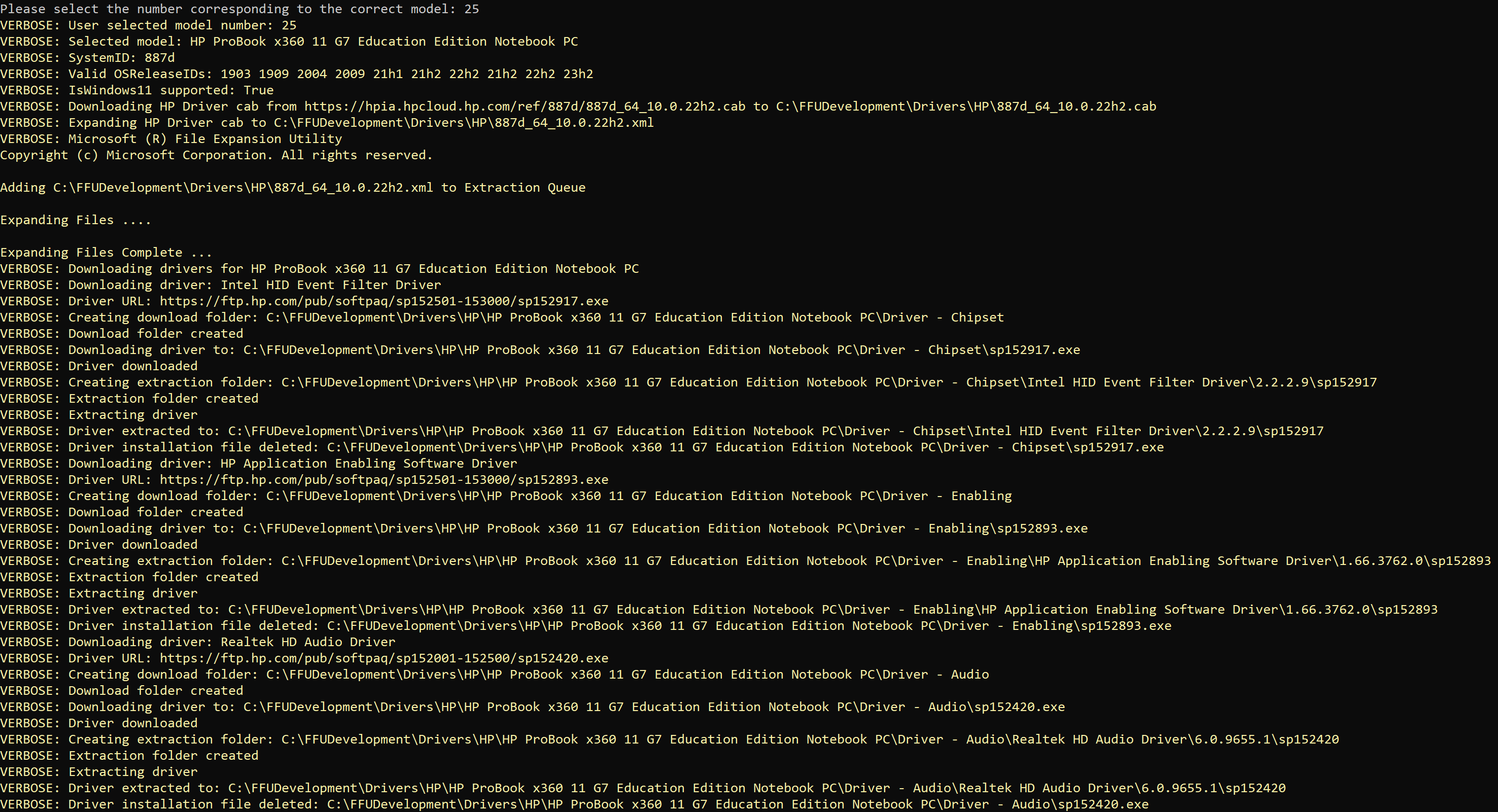
For HP, the script uses the same process as the HP Image Assistant tool to automate the downloading of drivers. It’s using the Get-HPDrivers function which accepts the -Make, -Model, -WindowsArch, -WindowsRelease, and -WindowsVersion parameters. HP is the only vendor that uses -WindowsVersion (e.g. 23h2) for its drivers. This is because their XML files contain the -WindowsVersion value in the file name. By default, the script uses 23h2 for the -WindowsVersion parameter. You can override that for whatever WindowsVersion you wish to use.

The following command line says that we want to download HP x360 drivers for Windows 10 version 22h2.

.\BuildFFUVM.ps1 **-make 'HP' -model 'x360'** -WindowsSKU 'Pro' -Installapps $true -InstallOffice $true -InstallDrivers $true -VMSwitchName 'external' -VMHostIPAddress '192.168.1.158' -CreateDeploymentMedia $true -CreateCaptureMedia $true -BuildUSBDrive $true -UpdateLatestCU $true -UpdateLatestNet $true -UpdateLatestDefender $true -UpdateEdge $true -UpdateOneDrive $true -RemoveFFU $true **-WindowsRelease 10 -WindowsVersion '22h2'** -Verbose



HP has 40 models that contain the string x360 in the model name. This is because HP uses generations for most models and its XML contains all of the generations that are supported for the corresponding WindowsRelease and WindowsVersion (e.g. Windows 10 22h2). I want to select the HP ProBook x360 11 G7 Education Edition Notebook PC which is number 25. The below screenshot shows the output of selecting the HP ProBook x360 11 G7 Education Edition Notebook PC



If you were to enter the exact model name (e.g. -model ‘HP ProBook x360 11 G7 Education Edition Notebook PC’), the script wouldn’t prompt you to select from a list of models.

## Manually adding drivers

There is an empty folder named Drivers where you can add the drivers you want to include in the FFU. This is typically used if the FFU you are building is for a specific model (e.g. Surface Laptop Go 2). You wouldn’t want to include drivers for multiple models as that can bloat the drivers store and waste space on the device, and even cause plug and play matching to match older or unexpected drivers. If you have multiple models you want to use this FFU with, you can dynamically choose which drivers you want on the deployment USB drive by creating a drivers folder on the deployment USB drive and creating folders for each model and the associated drivers. For more information on how to modify the USB drive to handle drivers, see the [Drivers support](#_Having_DISM_inject) section.

In 2402.1, the ability to add WinPE drivers was added. There is now a .\FFUDevelopment\PEDrivers folder where you can copy your PE drivers and they will be added to the USB boot media if you provide the -CopyPEDrivers $true parameter.

# Preparing the command line

The following table is a list of all parameters the script accepts.

## Script parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| AllowExternalHardDiskMedia | Bool | If $true, will allow the use of media identified as External Hard Disk media via WMI class Win32\_DiskDrive. Default is not defined. |
| AllowVHDXCaching | Bool | When set to $true, will cache the VHDX file to the $FFUDevelopmentPath\VHDXCache folder and create a config json file that will keep track of the Windows build information, the updates installed, and the logical sector byte size information. Default is $false. |
| AppsScriptVariables | Hashtable | When passed a hashtable, the script will alter the $FFUDevelopmentPath\Apps\InstallAppsandSysprep.cmd file to set variables with the hashtable keys as variable names and the hashtable values their content.   e.g. .\BuildFFUVM.ps1 […] -AppsScriptVariables @{InstallVMwareTools = "true"} […]  InstallAppsandSysprep.cmd contains:  REM START Batch variables placeholder  set InstallVMwareTools=true  REM END Batch variables placeholder  And later, we can use it to branch our code in the InstallAppsandSysprep.cmd file:  IF "%InstallVMwareTools%"=="true" (  […]  ) |
| BuildUSBDrive | Bool | When set to $true, will partition and format a USB drive and copy the captured FFU to the drive. |
| CleanupAppsISO | Bool | When set to $true, will remove the Apps ISO after the FFU has been captured. Default is $true. |
| CleanupCaptureISO | Bool | When set to $true, will remove the WinPE capture ISO after the FFU has been captured. Default is $true. |
| CleanupDeployISO | Bool | When set to $true, will remove the WinPE deployment ISO after the FFU has been captured. Default is $true. |
| CleanupDrivers | Bool | Used to delete the drivers folders underneath the $DriversFolder path (e.g. C:\FFUDevelopmentPath\Drivers\HP) after the FFU has been built. Default is $true. |
| CompactOS | Bool | When set to $true, will compact the OS when building the FFU. Default is $true. |
| ConfigFile | String | Path to a JSON file containing parameters to use for the script. |
| CopyAutopilot | Bool | When set to $true, will copy the $FFUDevelopmentPath\Autopilot folder to the Deployment partition of the USB drive. Default is $false. |
| CopyDrivers | Bool | When set to $true, will copy the drivers from the $FFUDevelopmentPath\Drivers folder to the Drivers folder on the deploy partition of the USB drive. Default is $false. |
| CopyPEDrivers | Bool | When set to $true, will copy the drivers from the $FFUDevelopmentPath\PEDrivers folder to the WinPE deployment media. Default is $false. |
| CopyPPKG | Bool | When set to $true, will copy the provisioning package from the $FFUDevelopmentPath\PPKG folder to the Deployment partition of the USB drive. Default is $false. |
| CopyUnattend | Bool | When set to $true, will copy the $FFUDevelopmentPath\Unattend folder to the Deployment partition of the USB drive. Default is $false. |
| CreateCaptureMedia | Bool | When set to $true, this will create WinPE capture media for use when $InstallApps is set to $true. This capture media will be automatically attached to the VM and the boot order will be changed to automate the capture of the FFU. |
| CreateDeploymentMedia | Bool | When set to $true, this will create WinPE deployment media for use when deploying to a physical device. |
| CustomFFUNameTemplate | String | Sets a custom FFU output name with placeholders. Allowed placeholders are: {WindowsRelease}, {WindowsVersion}, {SKU}, {BuildDate}, {yyyy}, {MM}, {dd}, {H}, {hh}, {mm}, {tt}.  {WindowsRelease} = 10, 11, 2016, 2019, 2022, 2025  {WindowsVersion} = 1607, 1809, 21h2, 22h2, 23h2, 24h2, etc  {SKU} = Home, Home N, Home Single Language, Education, Education N, Pro, Pro N, Pro Education, Pro Education N, Pro for Workstations, Pro N for Workstations, Enterprise, Enterprise N, Standard, Standard (Desktop Experience), Datacenter, Datacenter (Desktop Experience) {BuildDate} = e.g. Dec2024 {yyyy} = e.g. 2024 {MM} = 2 digit month format (e.g. 12 for December) {dd} = Day of the month in 2 digit format (19) {HH} = Current hour in 24-hour format (e.g., 14 for 2 PM)  {hh} = Current hour in 12-hour format (e.g., 02 for 2 PM)  {mm} = Current minute in 2-digit format (e.g., 09)  {tt} = Current AM/PM designator (e.g., AM or PM) |
| Disksize | Uint64 | Size of the virtual hard disk for the virtual machine. Default is a 30GB dynamic disk. |
| DriversFolder | String | Location where Drivers will either be downloaded, and/or the location of the drivers you wish to be added to the FFU, or copied to the deploy partition of the USB drive. The default location is $FFUDevelopmentPath\Drivers (e.g. C:\FFUDevelopmentPath\Drivers). |
| ExportConfigFile | String | Path to a JSON file to export the parameters used for the script. |
| FFUCaptureLocation | String | Path to the folder where the captured FFU will be stored. Default is $FFUDevelopmentPath\FFU. |
| FFUDevelopmentPath | String | Path to the FFU development folder. Default is C:\FFUDevelopment. |
| FFUPrefix | String | Prefix for the generated FFU file. Default is \_FFU. |
| Headers | Hashtable | This hash table is used in conjunction with the Useragent when invoking Invoke-Webrequest or Invoke-RestMethod. This has been helpful when interacting with the Microsoft Download Center and preventing intermittent errors. If interested in the default value, reference the script itself. |
| InstallApps | Bool | When set to $true, the script will create an Apps.iso file from the $FFUDevelopmentPath\Apps folder. It will also create a VM, mount the Apps.ISO, install the Apps, sysprep, and capture the VM. When set to $False, the FFU is created from a VHDX file. No VM is created. |
| InstallDrivers | Bool | Install device drivers from the specified $FFUDevelopmentPath\Drivers folder if set to $true. You will need to the drivers and put them in the Drivers folder. The script will recurse the drivers folder and add the drivers to the FFU. |
| InstallOffice | Bool | Install Microsoft Office if set to $true. The script will download the latest ODT and Office files in the $FFUDevelopmentPath\Apps\Office folder and install Office in the FFU via VM. |
| ISOPath | String | Path to the Windows 10/11 ISO file. If not defined, the script will download Windows 11 x64 English consumer media by default. |
| LogicalSectorBytes | Unit32 | unit32 value of 512 or 4096. Most devices shouldn’t need to change from 512. 4kn drives, which include UFS, will need to change this value to 4096. Default is 512. |
| Make | String | Used for automatically downloading drivers. Valid values are 'Microsoft', 'Dell', 'HP', 'Lenovo'. The script will throw an error if any other string value is used. |
| MediaType | String | String value of either 'business' or 'consumer'. This is used to identify which media type to download. Default is 'consumer'. |
| Memory | Uint64 | Amount of memory to allocate for the virtual machine. Recommended to use 8GB if possible, especially for Windows 11. Use 4GB if necessary. Default value 4GB. |
| Model | String | Used for automatically downloading drivers with the Make parameter. |
| Optimize | Bool | When set to $true, will optimize the FFU file. Optimized FFU files allow for the target device to use all available disk space. Default is $true. |
| OptionalFeatures | String | Provide a semi-colon separated list of Windows optional features you want to include in the FFU (e.g. netfx3;TFTP). |
| Processors | Int | Number of virtual processors for the virtual machine. Recommend at least 4. Default value 4. |
| ProductKey | String | Product key for the Windows 10/11 edition specified in WindowsSKU. This will overwrite whatever SKU is entered for WindowsSKU. Recommended to use if you want to use a MAK or KMS key to activate Enterprise or Education. If using VL media instead of consumer media, you'll want to enter a MAK or KMS key here. |
| PromptExternalHardDiskMedia | Bool | If $true and AllowExternalHardDiskMedia is $true, the script will prompt to select which drive to use. When set to $true, only a single drive will be created. If $false, the script won't prompt for which external hard disk to use and can use multiple external hard disks, similar to how removable USB drives function. |
| RemoveFFU | Bool | When set to $true, will remove the FFU file from the $FFUDevelopmentPath\FFU folder after it has been copied to the USB drive. Default is $false. |
| ShareName | String | Name of the shared folder for FFU capture. The default is FFUCaptureShare. This share will be created with rights for the user account. When finished, the share will be removed. |
| UpdateEdge | Bool | When set to $true, will download and install the latest Microsoft Edge for Windows 10/11. Default is $false. |
| UpdateLatestCU | Bool | When set to $true, will download and install the latest cumulative update for Windows 10/11. Default is $false. |
| UpdateLatestDefender | Bool | When set to $true, will download and install the latest Windows Defender definitions and Defender platform update. Default is $false. |
| UpdateLatestMSRT | Bool | When set to $true, will download and install the latest Windows Malicious Software Removal Tool. Default is $false. |
| UpdateLatestNet | Bool | When set to $true, will download and install the latest .NET Framework for Windows 10/11. Default is $false. |
| UpdateOneDrive | Bool | When set to $true, will download and install the latest OneDrive for Windows 10/11 and install it as a per machine installation instead of per user. Default is $false. |
| UpdatePreviewCU | Bool | When set to $true, will download and install the cumulative update ‘Preview’ for Windows 10/11. Note, if set, UpdateLastestCU is ignored and UpdatePreviewCU will take Precedence. Default is $false. |
| UserAgent | String | The useragent string is used when invoking Invoke-Webrequest or Invoke-RestMethod. This has been helpful when interacting with the Microsoft Download Center and preventing intermittent errors. Default is Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36 Edg/125.0.0.0. |
| Username | String | Username for accessing the shared folder. The default is ffu\_user. The script will auto-create the account and password. When finished, it will remove the account. |
| VMLocation | String | Default is $FFUDevelopmentPath\VM. This is the location of the VHDX that gets created where Windows will be installed to. |
| VMHostIPAddress | String | IP address of the Hyper-V host for FFU capture. If $InstallApps is set to $true, this parameter must be configured. You must manually configure this. The script will not auto detect your IP (depending on your network adapters, it may not find the correct IP). |
| VMSwitchName | String | Name of the Hyper-V virtual switch. If $InstallApps is set to $true, this must be set. This is required to capture the FFU from the VM. The default is \*external\*, but you will likely need to change this. |
| WindowsArch | String | String value of x86 or x64. This is used to identify which architecture of Windows to download. Default is x64. |
| WindowsLang | String | String value in language-region format (e.g. en-us). This is used to identify which language of media to download. Default is en-us. Windows Language Support. |
| WindowsRelease | Int | Integer value of 10 or 11. This is used to identify which release of Windows to download. Default is 11. |
| WindowsSKU | String | Edition of Windows 10/11 to be installed, e.g., 'Home', 'Home\_N', 'Home\_SL', 'EDU', 'EDU\_N', 'Pro', 'Pro\_N', 'Pro\_EDU', 'Pro\_Edu\_N', 'Pro\_WKS', 'Pro\_WKS\_N'. |
| WindowsVersion | String | String value of the Windows version to download. This is used to identify which version of Windows to download. Default is 23h2. |

When running the script, the **InstallApps** parameter is what dictates whether a Hyper-V VM gets created. To install Office or any 3rd party app, Windows needs to be deployed so we can install the app and then capture the FFU. To capture, the script will mount a WinPE ISO, boot from that, and capture the FFU over the network to your VM host.

This means that when **InstallApps** is set to $true, you also need to set **VMSwitchName** to the name of your VMSwitch in Hyper-V, as well as your **VMHostIPAddress.** While it’s generally easy to find the IP address of the machine, it’s more difficult if you have different network adapters or VMSwitches. So, you’ll need to specify them when running the script.

Note: If -ISOPath is not defined and the script is downloading the Windows media, $InstallApps will always be set to true, regardless of what you define in your command line. This is due to an issue with ESD files being applied to vhdx and then captured into a FFU. When deploying, the device ends up with an older-style OOBE and in a bad state. Deploying the ESD to VM, sysprepping, then capturing fixes this.

# Preparing a Configuration File (New in 2412.1)

A configuration file can now be used to configure the parameters in lieu of, or in conjunction with, parameters specified on the command line. Configuration files are especially helpful for those making FFUs for different models, Windows releases, application sets, and more.

To use, run:

**.\BuildFFUVM.ps1 -ConfigFile ‘C:\FFUDevelopment\config\Sample\_default.json’ -verbose**

## Creating your own Coniguration Json file

If you have a command line that you’ve been using for awhile and would like to convert it to a json file automatically, run your command line like normal, adding

**-exportConfigFile ‘C:\FFUDevelopment\config\YourConfigFile.json’**

to the end of the command. Doing this will generate a well-formatted json file with your configuration settings.

You can also temporarily overwrite parameters while using a config file. Using the following sample command:

**.\BuildFFUVM.ps1 -ConfigFile ‘C:\FFUDevelopment\config\Sample\_default.json’ -verbose**

If you’d like to not include Office (the Sample\_default.json file installs Office), you’d add -InstallOffice $False to the command line

**.\BuildFFUVM.ps1 -ConfigFile ‘C:\FFUDevelopment\config\Sample\_default.json’ -verbose -InstallOffice $False**

Doing this will temporarily overwrite whatever is in the json for the InstallOffice parameter. It will not modify the json file. If you would like to change the json file, you can add -exportConfigFile ‘C:\FFUDevelopment\config\Sample\_default.json’ and that will overwrite the json file with the new parameter.

**.\BuildFFUVM.ps1 -ConfigFile ‘C:\FFUDevelopment\config\Sample\_default.json’ -verbose -InstallOffice $False -exportConfigFile ‘C:\FFUDevelopment\config\Sample\_default.json’**

# Command Lines

With the number of parameters, there are quite a few ways to build your FFU. Below are some of the more common. The first in the list is a good starting point to create your first FFU with everything up to date if you want the latest Windows, Office, etc.

**Command line for most people who want to download Windows 11 Pro x64 media in English (US) with the latest Windows Cumulative Update, .NET Framework, Defender platform and definition updates, Edge, OneDrive, and Office/M365 Apps. It will also copy drivers to the FFU. This can take about 40 minutes to create the FFU due to the time it takes to download and install the updates.**

.\BuildFFUVM.ps1 -WindowsSKU 'Pro' -Installapps $true -InstallOffice $true -InstallDrivers $true -VMSwitchName 'Name of your VM Switch in Hyper-V' -VMHostIPAddress 'Your IP Address' -CreateDeploymentMedia $true -BuildUSBDrive $true -UpdateLatestCU $true -UpdateLatestNet $true -UpdateLatestDefender $true -UpdateEdge $true -UpdateOneDrive $true -verbose

**Command line for those who want to download the Windows 11 Pro x64 media in English (US) and install the latest version of Office and drivers and build a USB drive**

.\BuildFFUVM.ps1 -WindowsSKU 'Pro' -Installapps $true -InstallOffice $true -InstallDrivers $true -VMSwitchName 'Name of your VM Switch in Hyper-V' -VMHostIPAddress 'Your IP Address' -CreateCaptureMedia $true -CreateDeploymentMedia $true -BuildUSBDrive $true -verbose

**Command line for those who want to download the latest Windows 11 Pro x64 media in French (CA) and install the latest version of Office and drivers and build a USB drive**

.\BuildFFUVM.ps1 -WindowsSKU 'Pro' -Installapps $true -InstallOffice $true -InstallDrivers $true -VMSwitchName 'Name of your VM Switch in Hyper-V' -VMHostIPAddress 'Your IP Address' -CreateDeploymentMedia $true -BuildUSBDrive $true -WindowsRelease 11 -WindowsArch 'x64' -WindowsLang 'fr-ca' -verbose

**Create FFU with your own Windows Pro ISO, Install Office (and/or additional apps), Install Drivers, Create Deployment Media, Build USB drive, with Product Key (get this from VLSC)**

If providing your own ISO from VLSC, you’ll also want to pass the -ProductKey parameter with your VLSC key. If you’re providing an ISO from MSDN or consumer media obtained elsewhere, ProductKey isn’t needed if the SKU you’re installing matches the SKU of the key in the firmware of the device.

.\BuildFFUVM.ps1 -ISOPath 'C:\path\_to\_iso\Windows.iso' -WindowsSKU 'Pro' -Installapps $true -InstallOffice $true -InstallDrivers $true -VMSwitchName 'Name of your VM Switch in Hyper-V' -VMHostIPAddress 'Your IP Address' -CreateDeploymentMedia $true -BuildUSBDrive $true -ProductKey ‘XXXXX-XXXXX-XXXXX-XXXXX-XXXXX’ -verbose

**Create FFU Only with your own ISO, Create Deployment Media, Build USB drive**

The command line below is for those who just want a FFU with no drivers, apps, or Office. You might use this if you have different models and want to dynamically add the drivers during deployment. See the [Driver Support](#_Toc134205535) section later in this document.

.\BuildFFUVM.ps1 -ISOPath 'C:\path\_to\_iso\Windows.iso' -WindowsSKU 'Pro' -CreateDeploymentMedia $true -BuildUSBDrive $true -ProductKey ‘XXXXX-XXXXX-XXXXX-XXXXX-XXXXX’ -verbose

**Create FFU, Install Apps, Install Drivers, no Office, Create Deployment Media, Build USB drive**

Command line for those who just want a FFU with Apps and drivers, no Office with ISO

.\BuildFFUVM.ps1 -ISOPath 'C:\path\_to\_iso\Windows.iso' -WindowsSKU 'Pro' -Installapps $true -InstallOffice $false -InstallDrivers $true -VMSwitchName 'Name of your VM Switch in Hyper-V' -VMHostIPAddress 'Your IP Address' -CreateCaptureMedia $true -CreateDeploymentMedia $true -BuildUSBDrive $true -verbose

# Running the Script

Once you’ve downloaded the FFUDevelopment.zip and extracted it to C:\FFUDevelopment

**Click Start** – type **Powershell** – **Right Click Windows PowerShell and select Run as Administrator**

Use one of the command lines from above. If this is your first time running the script, use this (make sure to modify it)  
  
c:\FFUDevelopment\BuildFFUVM.ps1 -WindowsSKU 'Pro' -Installapps $true -InstallOffice $true -InstallDrivers $true -VMSwitchName 'Name of your VM Switch in Hyper-V' -VMHostIPAddress 'Your IP Address' -CreateDeploymentMedia $true -BuildUSBDrive $true -UpdateLatestCU $true -UpdateLatestNet $true -UpdateLatestDefender $true -UpdateEdge $true -UpdateOneDrive $true -verbose

Note – if you get an error that the script cannot be loaded run the following command:  
  
**Set-ExecutionPolicy -executionpolicy Bypass**  
  
Then try running the script again  
  
You should see output like the following  
A screen shot of a computer program

Description automatically generated with low confidence

When the script is complete, you should see the following  
  
A screenshot of a computer program

Description automatically generated with medium confidence

Depending on how fast your machine and network connection is and the options you’ve selected, it could take 20 minutes or more for the whole process to complete. Once it’s finished, your USB drive can be used to image a machine. The USB drive should have two partitions: Boot and Deploy. The deploy partition will have the FFU file.  
  
A screenshot of a computer

Description automatically generated with medium confidence

# Customizing the USB drive (optional)

The following customizations can be done to your USB drive

* Adding additional drivers
* Device naming
* Adding Provisioning package(s)
* Adding Autopilot for Existing Devices (used for devices not registered to Autopilot)

None of these customizations are required. If you’ve included drivers in your FFU during the building of the FFU, then it’s not necessary to add additional drivers. However, if you didn’t include drivers, you should follow the below process for adding drivers if you plan to use this FFU on multiple models.

The most common customizations are drivers, device naming, and provisioning packages (if not using Autopilot). Naming the device in the FFU process can save you a reboot during Autopilot or using a Provisioning package.

# Driver Support (optional but recommended)

Adding drivers is not required in most cases, however it will result in a better experience on the device. We are seeing more instances where mouse/keyboard drivers aren’t inbox in WinPE or full Windows, which makes it challenging to get through OOBE. If you do not add drivers, Windows Update will update the device. The first logged on user will see driver installation pop ups as WU installs the drivers. If IT will be signing on first, they can deal with the driver prompts and validation before delivering to the end user.

If you would like to add drivers, there are two ways to do it.

1. Do it during FFU capture setting the -InstallDrivers parameter to $true.
2. Create a drivers folder on your Deploy USB partition and copy your drivers there (e:\drivers\Lenovo 300e) (recommended if managing multiple models and want a single FFU)

The first option will result in faster imaging. This option is good if imaging the same model in an environment many times (hundreds or thousands of devices before the school year starts). It is not recommended to inject drivers of many models in a single FFU. Doing so will increase the driver store considerably. When dealing with devices that have limited storage, you run the risk of running out of disk space. You could create your own FFU per model with only the drivers injected for that specific model.

The second option is easier; however, it will take longer to image. This is due to dism injecting the drivers on the fly. The time it takes will depend on how many drivers are added and the size of the drivers.

## Having DISM inject drivers on the fly (recommended if managing multiple models)

Simply create a folder on the deploy partition called drivers. Inside that folder, create sub-folders for each model you will be imaging. If imaging multiple models, a pick list will show up asking which set of drivers to install.

Graphical user interface, application

Description automatically generated

## How to inject drivers into FFU

This step is not necessary if you are adding drivers by setting $InstallDrivers to $true when creating your FFU.

If you have a FFU that doesn’t have drivers added and would like to add drivers, you can manually do it with dism.

1. If you haven’t already, download the drivers you wish to install for the model you are deploying. Normally, OEMs will provide an enterprise SCCM cab/zip of the drivers. Some, however, do not, especially for the low-cost EDU devices (Lenovo doesn’t for the 14W, 100e, 300e. They require individual downloads for each component, which are all exes). Don’t forget to extract the drivers.   
     
   The 3189 driver cab looks like this once extracted  
     
   A screenshot of a cell phone

   Description automatically generated

For Lenovo devices, each exe can be extracted via the command line using:  
  
Filename.exe /VERYSILENT /Extract="Yes" /DIR=<PathToWhereYouWantToExtractTo>

1. **Right-click Start Menu** icon and select **Windows Powershell (Admin)**
2. Run **mkdir c:\mount**
3. Run **Dism /Mount-Image /ImageFile:"<PathToFFUFile>" /Index:1 /MountDir:"C:\mount"**  
   (where <PathToFFUFile> is the location of your FFU e.g. e:\Win10\_1903\_Pro\_Edu.ffu)
4. Run **dism /image:C:\mount /Add-Driver /Driver:<PathToDriverFiles> /Recurse**   
   (Replace the c:\temp\drivers with the location of your extracted driver files)
5. Run **Dism /Unmount-Image /MountDir:C:\mount /commit**

# Device Naming (optional but recommended)

**New as of June 2020**

Device naming can be done from PE. The way this works is by leveraging an unattend.xml file to either take input from the user at imaging time or read a list of prefix values and append the serial number of the device. There are some major benefits to doing this:

1. Total deployment time is reduced if naming is set at FFU deployment time since there is no additional reboot done during OOBE.
2. Reduces the need for multiple provisioning packages or autopilot profiles. This means you can use a single PPKG or autopilot profile.
3. Allow for future considerations on how to customize/automate device naming in the future

**IMPORTANT – If using a provisioning package or autopilot json file, DO NOT specify a name in either of these. They will overwrite the name you have specified in the unattend.xml.**

By leveraging the unattend.xml, you can also add additional steps during the specialize phase to customize the experience even further.

## Prompt to enter name

1. Create a folder on the deploy partition of the USB drive named **unattend**
2. In the unattend folder, create a file named **unattend.xml** (this file has already been created if you downloaded this from aka.ms/ffu or the internal Microsoft FFU deployment share . It is in the unattend folder in that location).
3. The contents of the unattend.xml file should be the following  
     
   <?xml version="1.0" encoding="utf-8"?>

<unattend xmlns="urn:schemas-microsoft-com:unattend">

<settings pass="specialize">

<component name="Microsoft-Windows-Shell-Setup" processorArchitecture="amd64" publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS" xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<ComputerName>MyComputer</ComputerName>

</component>

</settings>

</unattend>

Graphical user interface, text

Description automatically generated

If this file is located during imaging, the user will be prompted to enter a name before the FFU is applied to the device.

## Specifying Multiple Name Prefixes

If you have multiple device name prefixes for different locations or device use cases, or even a single prefix, you can specify a prefixes.txt file in the unattend folder. If the prefixes.txt file is detected and a single prefix is listed, the device will just use that prefix and append the serial number of the device. If there are multiple prefixes listed in the prefixes.txt file, you will be prompted to select which prefix you want to name the device and the serial number will be appended to that prefix. If you want a dash in the name, include the dash in the prefix (e.g. if ABCD- is in the prefixes.txt file, the device name will be ABCD-*SerialNumber*).

1. Follow the previous steps used to create the unattend folder and the unattend.xml file.
2. Create a file named **prefixes.txt** in the **unattend** folder
3. In the prefixes.txt file, enter one prefix per line and save the file

Your folder should look like the following

Graphical user interface, text

Description automatically generated

Your prefixes.txt file should look similar to the following image

Graphical user interface, text, application

Description automatically generated

# Device Enrollment

There are three common ways to enroll new devices using this method to Azure AD and Intune

1. Autopilot
2. Provisioning packages
3. Autopilot for Existing Devices

If using Autopilot, your devices should already be registered, and an Autopilot profile should be assigned to them. At this point, you can Insert the USB drive to your device and boot from it.

However, if your devices aren’t enrolled into Autopilot, you can either use a Provisioning Package, or Autopilot for Existing devices.

Generally, AP or Provisioning Packages are the two common methods of deployment.

The reasons you may want to use Autopilot for existing devices vs. using Provisioning are:

1. Speed of USB drive removal – AP for existing devices allows a technician to remove the USB drive after the device applies the FFU and reboots for the first time. This is because there isn’t a provisioning package. This means you can take the USB drive out sooner and image another device. The FFU usually takes about 2-3 minutes to apply. Once the device reboots after applying the FFU, the drive can be removed if you are not using provisioning packages.
2. Setting up a device for 1:1 usage – AP for existing devices will set user device affinity up for the first user that enrolls the device. In this scenario, any applications deployed as available to users/devices in Intune will only show up for that user in the Company Portal. **Do not use this if the devices will be shared.**

If you’re in a rural location, have poor bandwidth, or are deploying shared devices (laptop carts/labs), provisioning packages are recommended. This guide isn’t designed to go in depth on provisioning packages, but there are ways to include complex apps like contain multiple folders and files. At a high level, you can put all of the content into a WIM file, mount the WIM, and install the apps. We’ll provide a guide on this at a later date.

# Optional - Copy provisioning package to the USB drive

**Process changed as of June 2020**

**Recommended for shared devices or devices that may be used 1:1 for now, but could be shared later and won’t be reimaged**

If you plan to use a provisioning package to enroll a device, follow the following steps.

1. On the **deploy partition** of your device, **create a new folder** named **PPKG**
2. Copy one or multiple .PPKG files to the PPKG folder

In June 2020, this process changed to allow for multiple provisioning packages to exist on a single USB drive. If multiple packages are detected, a menu will display each package and ask you to select which package to install. This is useful for organizations that have different packages for different use cases, packages configured for wired vs wireless scenarios, different device names, etc. and do not want to swap packages or have multiple USB drives.

Do NOT place a PPKG file on the BOOT partition of your USB drive AND create a PPKG folder on the deploy partition with one or multiple PPKG files in the folder. This will cause Windows to see two eligible PPKG files to be used during OOBE, which will result in you having to select which package to use at OOBE, slowing the overall process down.

**Important – If you set up the device naming method mentioned earlier, make sure to remove the ComputerName (or DNSComputerName) from your provisioning package. If you do not remove the ComputerName (or DNSComputerName) from the package, the package will overwrite the name you specified via the device naming method earlier. The Accounts node in your provisioning package should look like the following image.**  
  
Graphical user interface, text, application, email

Description automatically generated

# Optional - Using Autopilot for Existing Devices

If you want to configure your media to be able to Autopilot enroll any device, you can use Autopilot for existing devices.

This requires you to create a folder on your **Deploy** drive called **autopilot** and insert an AutopilotConfigurationFile.json file in that folder (E:\autopilot\AutopilotConfigurationFile.json). When the device finishes installing the FFU, it will copy the json file to C:\Windows\Provisioning\Autopilot\AutopilotConfigurationFile.json.

**Note –** AP for existing devices only works for User Driven profiles. It does not work for Self-Deploying or Pre-Provisioning scenarios. It also does not register the device into Autopilot. You can deploy an Autopilot Profile to a group that this device will end up in and select the setting **Convert all targeted devices to Autopilot**.

## Create User Driven Autopilot Profile

Make sure you're using User Driven Profiles (Self Deploying profiles don't work with Offline). In this example, we will query this profile by the display name of the profile. In this case, we'll be using User Driven as the display name.

1. In Intune, go to **Device Enrollment – Windows Enrollment – Deployment Profiles**
2. **Click Create Profile**
3. For **Name,** type in **User Driven** (the name MUST be User Driven for the powershell command to work later)
4. For **Convert all targeted devices to Autopilot** select **Yes**Note – do this in production environments where you want the devices to be autopilot enrolled. This will allow any devices with this profile targeted to go through the Autopilot workflow if the device is reimaged or reset. If you’re working in a lab environment where the devices you use may also be used for demoing provisioning scenarios (SUSPC, WCD) then make sure to set this to NO. You will still get the AP behavior when you use the json file if this is set to No, but the device isn’t really in Autopilot, so it can be used for provisioning and other scenarios later.

A screenshot of a cell phone

Description automatically generated

1. For OOBE settings, configure like below, making sure to select **deployment mode** as **User-Driven** and **Join to Azure AD** as **Azure AD Joined**  
   A screenshot of a cell phone

   Description automatically generated
2. Click **Next** through the rest of the wizard (even on the Assignments tab as you do not have to assign this profile to a group). On the Review + create tab, click **Create.**

## Install Azure AD, WindowsAutopilotIntune, Intune Graph Powershell modules

Open Powershell as an admin

**Run Install-Module AzureAD**

At the untrusted repository message, type Y or A

**Run Install-Module WindowsAutopilotIntune**

At the untrusted repository message, type Y or A  
**Run Install-Module Microsoft.Graph.Intune**

At the untrusted repository message, type Y or A

## Get the User Driven Autopilot profile and convert to JSON

Still in Powershell, run **Connect-MSGraph**

Type in your credentials

The below command will query for our Autopilot Profile with the display name User Driven. It will then convert the profile to the JSON format and save the file to your desktop as **AutoPilotConfigurationFile.json** and encode it using ASCII (must be saved as ASCII and it must use this exact name)

Run the following command

**Get-AutoPilotProfile |? {$\_.displayname -like '\*User Driven\*'} | ConvertTo-AutoPilotConfigurationJSON | Out-File -FilePath $env:userprofile\desktop\AutoPilotConfigurationFile.json -Encoding ASCII**

1. Go to your desktop and copy the AutopilotConfigurationFile.json file
2. On the **Deploy** drive of your USB drive, make a folder called **autopilot**
3. Copy the **AutopilotConfigurationFile.json** file to the **autopilot** folder  
     
   This should look like the following  
     
   A screenshot of a cell phone

   Description automatically generated

## Multiple Autopilot Configuration Files

If imaging many devices that will go to different locations or have different Autopilot settings, you will need multiple Autpilot profiles which means you will also need multiple AutoPilotConfigurationFile.json files.

Give each file a unique name (e.g. ContosoHighSchool.json, FabrakamMiddleSchool.json) and store them in your autopilot folder. If multiple files are detected, you will be prompted for which file you want to use.

# Optional - Copying a custom WinRE.wim

If you are deploying a device that does not have built in drivers in Windows, three things need to be updated

1. WinPE (Storage and possibly keyboard/mouse drivers)
2. Each FFU file (Storage and network drivers)
3. WinRE (Storage and possibly keyboard/mouse drivers)

You can create your own custom WinRE.wim file with the appropriate drivers. Keep in mind that the version of WinRE must match the version of Windows you are deploying.

Your Deploy partition should look like the following:

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

# Optional – Adding Multiple Components using unattend.xml

**New as of June 2022**

In prior releases, only a single component could be used, and it was used primarily to set the device name while in PE to help speed up imaging.

The ApplyFFU.ps1 script has been updated to allow multiple components to be specified in the specialize phase.

In the example below, we’re able to set the device name and run a command to set the date. This can be useful in scenarios where the device date is so old that AzureAD doesn’t allow it to enroll.

See the previous section **Device Naming (optional but recommended)** for how to configure the USB drive to leverage the unattend.xml file.

<?xml version="1.0" encoding="utf-8"?>

<unattend xmlns="urn:schemas-microsoft-com:unattend">

  <settings pass="specialize">

    <component name="Microsoft-Windows-Shell-Setup" processorArchitecture="amd64" publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS" xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

      <ComputerName>A-006259610866</ComputerName>

    </component>

    <component name="Microsoft-Windows-Deployment" processorArchitecture="amd64" publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS" xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

      <RunAsynchronous>

        <RunAsynchronousCommand wcm:action="add">

          <Path>cmd /c date 05-31-2022</Path>

          <Order>1</Order>

        </RunAsynchronousCommand>

      </RunAsynchronous>

    </component>

  </settings>

</unattend>

# Troubleshooting

## Deployment

If there are issues, there is a log on the deploy drive that will be created called Scriptlog.txt. That log can be useful in explaining why something failed. Make sure to only have one USB drive plugged into the device and only one hard disk. This process was not tested or validated with multiple USB drives and/or hard disks.

If you have a device that reboots immediately, chances are there aren’t any storage drivers available for that model in WinPE. Most Intel-based devices should work fine as Windows has inbox drivers. AMD may not have available drivers. You likely will need to modify the WinPE media to include drivers for your model. It’s also possible you may not have keyboard drivers.

## Creation

If you have issues with creating the FFU, the c:\FFUDevelopment\FFUDevelopment.log file may have some clues.

## Need help?

If you have any issues, please [open an Issue](https://github.com/rbalsleyMSFT/FFU/issues) on the Github repo and we’ll take a look as soon as we can.

# Windows Language Support

These are the supported languages that can be used with the -WindowsLang parameter when downloading the Windows ESD file

* ar-sa
* bg-bg
* cs-cz
* da-dk
* de-de
* el-gr
* en-gb
* en-us
* es-es
* es-mx
* et-ee
* fi-fi
* fr-ca
* fr-fr
* he-il
* hr-hr
* hu-hu
* it-it
* ja-jp
* ko-kr
* lt-lt
* lv-lv
* nb-no
* nl-nl
* pl-pl
* pt-br
* pt-pt
* ro-ro
* ru-ru
* sk-sk
* sl-si
* sr-latn-rs
* sv-se
* th-th
* tr-tr
* uk-ua
* zh-cn
* zh-tw