

Chapter 6. Drivers

In the previous chapter, we discussed the `CustomSettings.ini` file as it relates to `Unattend.xml`, what it does, how it works, and how to use it for your environment. We discussed that not only can we define global rules that apply to all task sequences, but we can also take advantage of variables to perform condition-based actions based on things such as hardware type, model, default gateway, or any variables that MDT defines during the gather phase, or even your own custom variables that you have created. These concepts are very important in our next topic of driver management.

In this chapter, we will discuss how to utilize MDT to make the complex world of device drivers into a much more manageable experience. We will focus on how drivers get installed via MDT, how to specifically control which drivers get installed, and general best practices around proper driver management.

We will cover the following topics in this chapter:

- Understanding offline servicing
- The MDT method of driver detection and injection
- Populating the Out-of-Box Drivers node of MDT
- Utilizing model variable to control what drivers are installed
- Drivers as applications
- Win PE drivers

Understanding offline servicing

Those of us who created images for deployment of Windows XP were often met with an enormous challenge of dealing with drivers for many different models of hardware. We were already forced to create separate images for different HAL families. Additionally, in order to deal with different hardware models within the same HAL family, the standard practice way was usually to have a `C:\Drivers` folder, which contained a copy of every possible driver that could be required by this image for all the different hardware models it would be installed to. There was an `OemPnPDriversPath` entry in the registry that individually listed each of the driver paths (subfolders under the `C:\Drivers` directory) for the Windows **Plug and Play (PnP)** process to locate and install the driver. As you can imagine, this was not a very efficient way to manage drivers. One reason is that every driver for every machine was staged in the image, causing the image size to grow; another reason being that we were relying on PnP to figure out the right driver to install, which gives

us less control of what driver actually gets installed, based on a driver ranking process.

Fast forward to Windows Vista and the current versions of Windows, and we can now utilize the magic of offline servicing to inject drivers into our WIM as it gets deployed. With this in mind, consider the concept of having your customized Windows image created through your reference image build process, but it contains no drivers. Now, when we go to deploy this image, we could utilize a process to detect all of the hardware in the target machine and then grab only the correct drivers that we need for this particular machine. Then, we can utilize DISM to inject them into our WIM before the WIM actually gets installed, thereby making the drivers available to be installed as Windows is installed on this machine. MDT is doing just that.

The MDT method of driver detection and injection

When we boot a target machine via our LiteTouch media, one of the initial task sequence steps will enumerate (via [PnPEnum.exe](#)) all of the PnP IDs for every device in the machine. Then, as part of the Inject Drivers task sequence step, we will search all of our Out-of-Box Driver INF files to find the matching driver, then MDT will utilize DISM to inject these drivers offline into the applied WIM.

NOTE

Note that, by default, we will be searching our entire Out-of-Box drivers repository and letting PnP figure things out.

We will later discuss how to force MDT to only choose from the drivers that we specify, thereby gaining strict control over which drivers actually get installed.

The preceding scenario indicates that this whole process hinges on the fact that we are searching through driver INF files to find the matching PnP IDs in order to correctly detect and install the correct driver. This brings up a concern; what if the driver does not contain an INF file, but rather it simply has to be installed via an EXE program? In this scenario, we cannot utilize the driver injection process; instead, we would treat this driver as an **application** in MDT, meaning that we would add a new application using the EXE program, and its accompanied files if present, as the source files, specifying the command-line syntax to launch the driver install program and install silently, and then adding this application as a task sequence step. I will

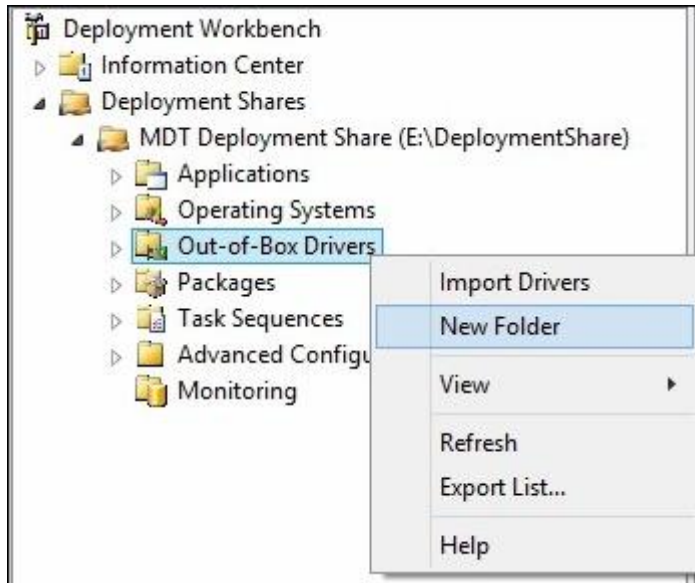
later demonstrate how to utilize conditional statements in your task sequence to only install this driver program on the model that it applies to, thereby keeping our task sequence flexible in order to be able to install correctly on any hardware.

Populating the Out-of-Box Drivers node of MDT

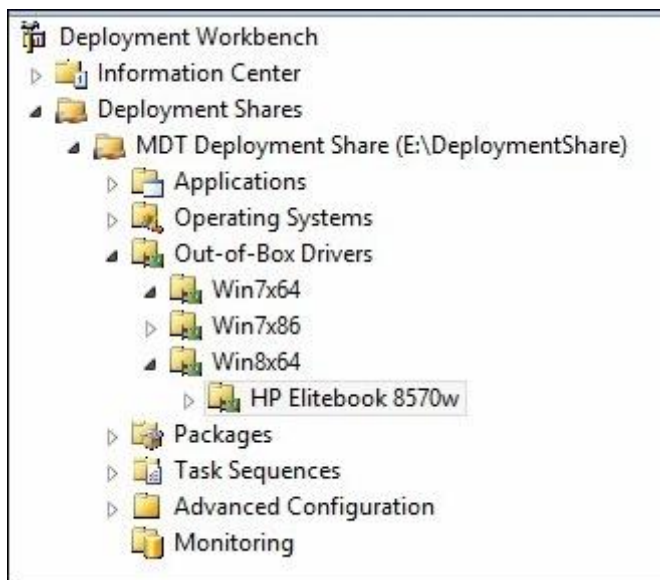
The first step will be to visit the OEM Manufacturer's website and download all device drivers for each model machine that we will be deploying to. Note that many OEMs now offer a deployment-specific download or CAB file that has all of the drivers for a particular model compressed into one single CAB file. This benefits you as you will not have to go through the hassle of downloading and extracting each individual driver for each device separately (NIC, video, audio, and so on). Once you have downloaded the necessary drivers, to store them in a folder for each specific model, you will need to extract the drivers within your folder before importing them into MDT.

Next, we want to create a folder structure under the **Out-of-Box Drivers** node in MDT to organize our drivers. This will not only allow easy manageability of drivers as new drivers are released by the OEM; but if we name the folders to match the model names exactly, we can later also introduce logic to limit our PnP search to the exact folder that contains the correct drivers for our particular hardware model. As we will have different drivers for x86 and x64, as well as for different operating systems, a general best practice would be to create the first hierarchy of your folder structure. Perform the following steps:

1. In order to create the folder structure, simply click on **Out-of-Box Drivers** and choose **New Folder**, as shown in the following screenshot:



2. Next, we will want to create a folder for each model that we will be deploying to:



3. In order to ensure that you are using the correct model name, you can use the following WMI query to see what the hardware returns as the model name:

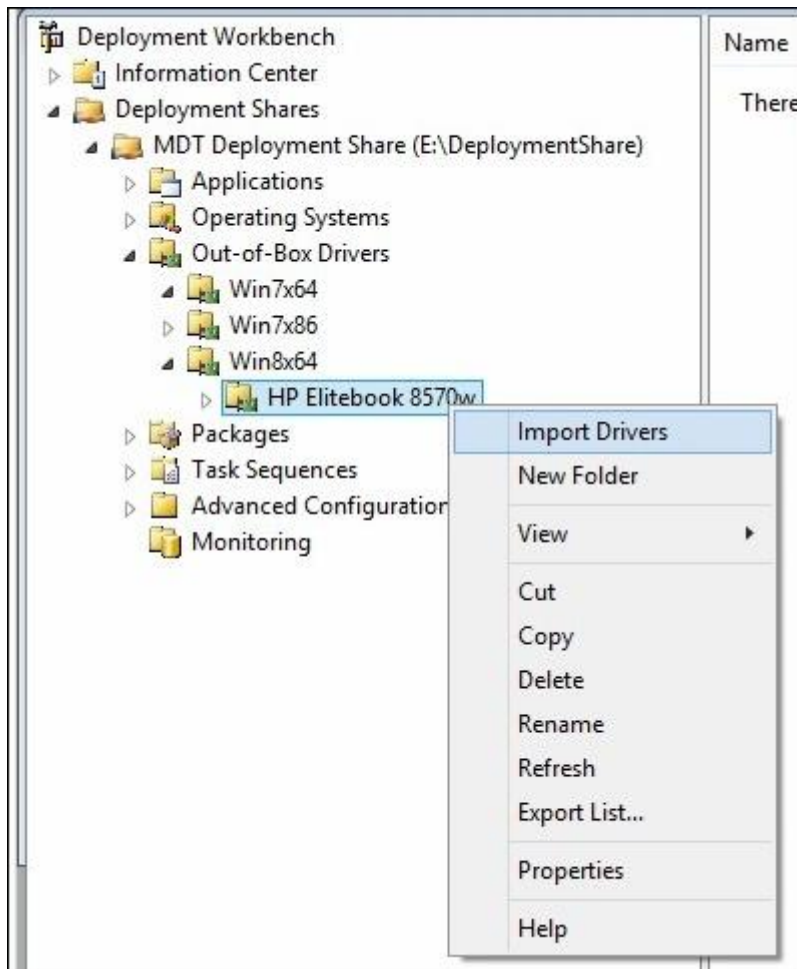
```
Administrator: Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>wmic computersystem get model
Model
HP EliteBook 8570w

C:\WINDOWS\system32>_
```

4. Once you have your folder structure created, you are ready to inject the drivers. Right-click on the model folder and choose **Import Drivers**. Point the

driver source directory to the folder where you have downloaded and extracted the OEM drivers:



NOTE

There is a checkbox stating **Import drivers even if they are duplicates of an existing driver**. This is because MDT is utilizing the **Single Instance Storage (SIS)** technology to store the drivers in the actual deployment share. If you are importing multiple copies of a driver to different folders, MDT only stores one copy of the file in the actual filesystem by default, and the folder structure you see within the MDT Workbench will be pointing duplicates to the same file in order to not waste any space.

As new drivers are released from the OEM, you can simply replace the drivers by going to the particular folder for this model, removing the old drivers, and importing the new drivers. Then, next time you install your WIM to this model, you will be using the new drivers, and you don't have to make any modifications or updates to your WIM.

Utilizing model variable to control what drivers are installed

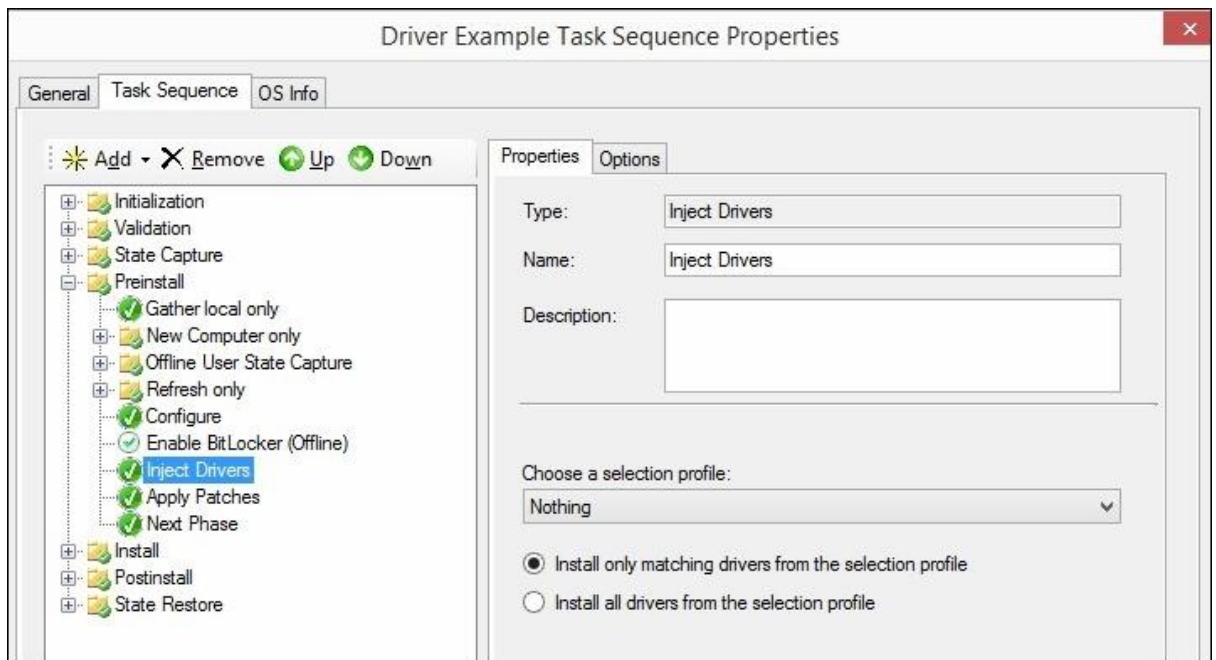
As mentioned earlier, while using a default MDT task sequence, the Inject Drivers task sequence step will search your entire Out-of-Box drivers repository to find a matching driver. If you only have a few hardware models, and all models are from the same manufacturer, then this could very well suit your needs without problem; but let's consider the following scenario.

Let's say you have a Dell model that has a rebranded Broadcom Network Adapter. You also have an HP model that has a rebranded Broadcom Network Adapter of the same chipset. Now, Broadcom, Dell, and HP each have a driver. Based on what we discussed earlier about how PnP finds a matching driver, it would be possible for any of these three drivers to be a match, if both devices reported the same PnP ID, then Windows' driver ranking process would determine which driver is installed based on signed versus unsigned, version number, inbox versus Out-of-Box, and so on. So, we could get in a situation where perhaps the HP driver was installed on the Dell. For us to have more control over which driver gets installed, while also keeping the task sequence flexible to work on any model, we can take advantage of variables and conditional statements.

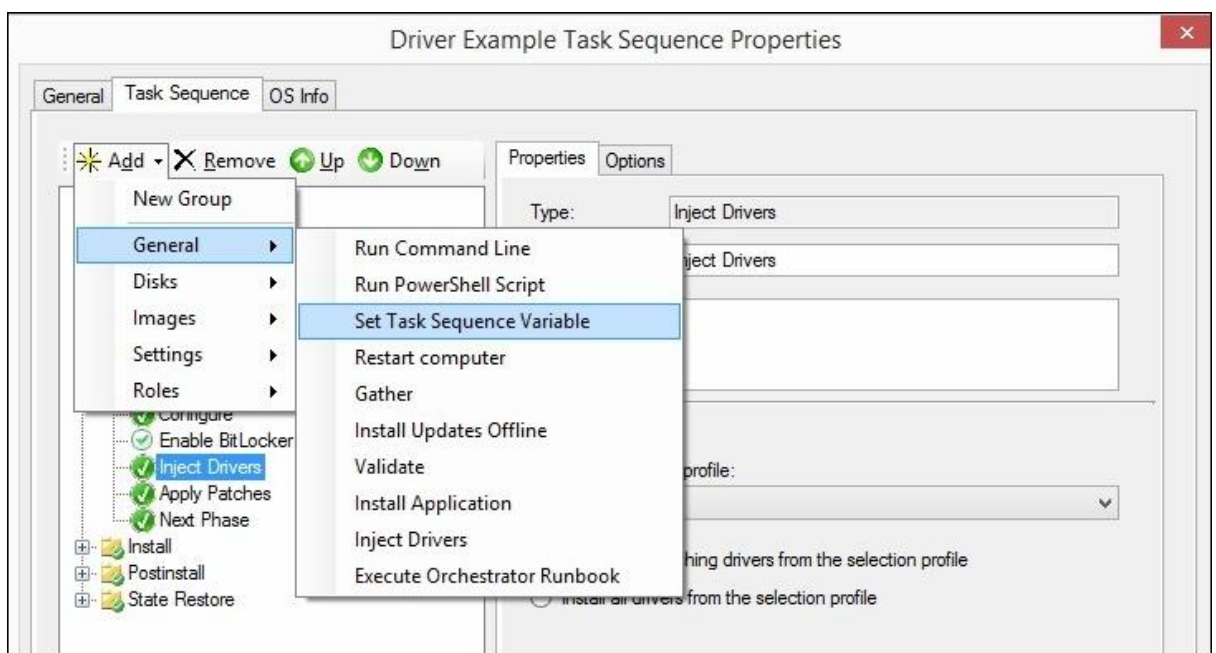
During the MDT Gather phase, MDT has already determined the exact model name of the machine we are installing to and has stored this value in a `%model%` variable. MDT also utilizes another built-in `DriverGroup00x` variable in order to set the path of where to look for the drivers. So, we can place a step in the task sequence to set `DriverGroup001` to point to a dynamic path that gets filled in with the exact model name by use of the `%model%` variable, as shown in the following screenshot example:

1. In the task sequence that you are using to deploy your image, under the **Preinstall** section, we will modify the **Inject Drivers** step to use the selection

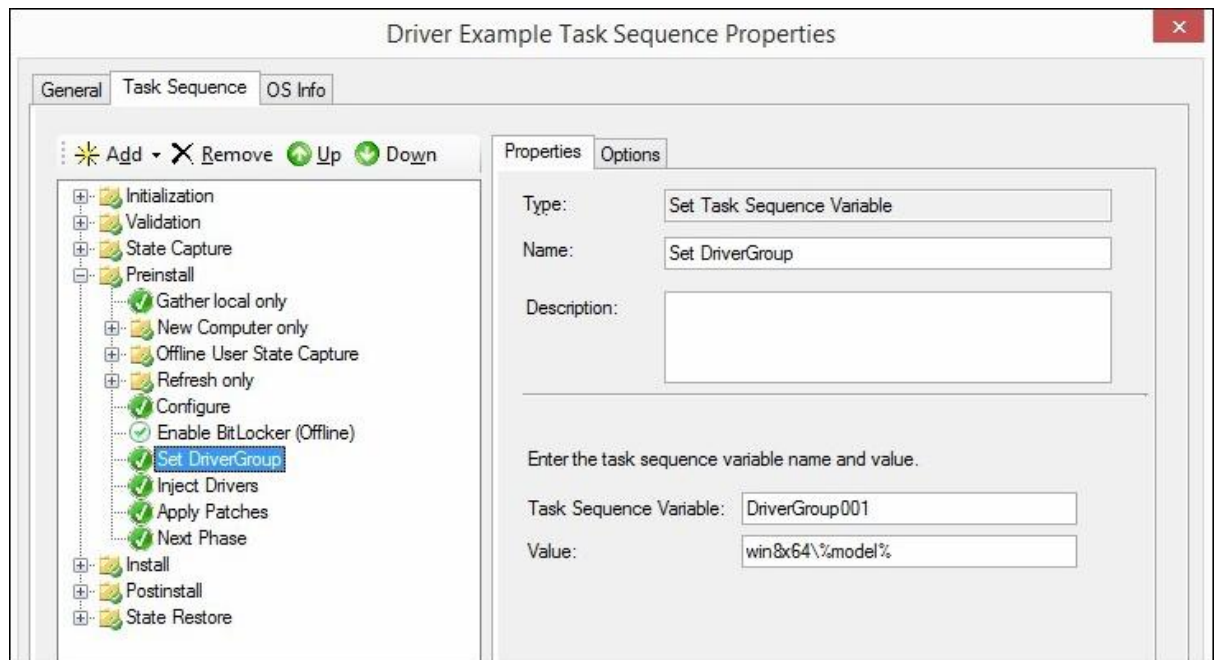
profile of **Nothing**, as shown in the following screenshot:



2. Next, we will add a step above the **Inject Drivers** step to set our `DriverGroup001` variable to point to the path of the model we are installing to, which will have the `%model%` variable filled in with the correct information, as follows:



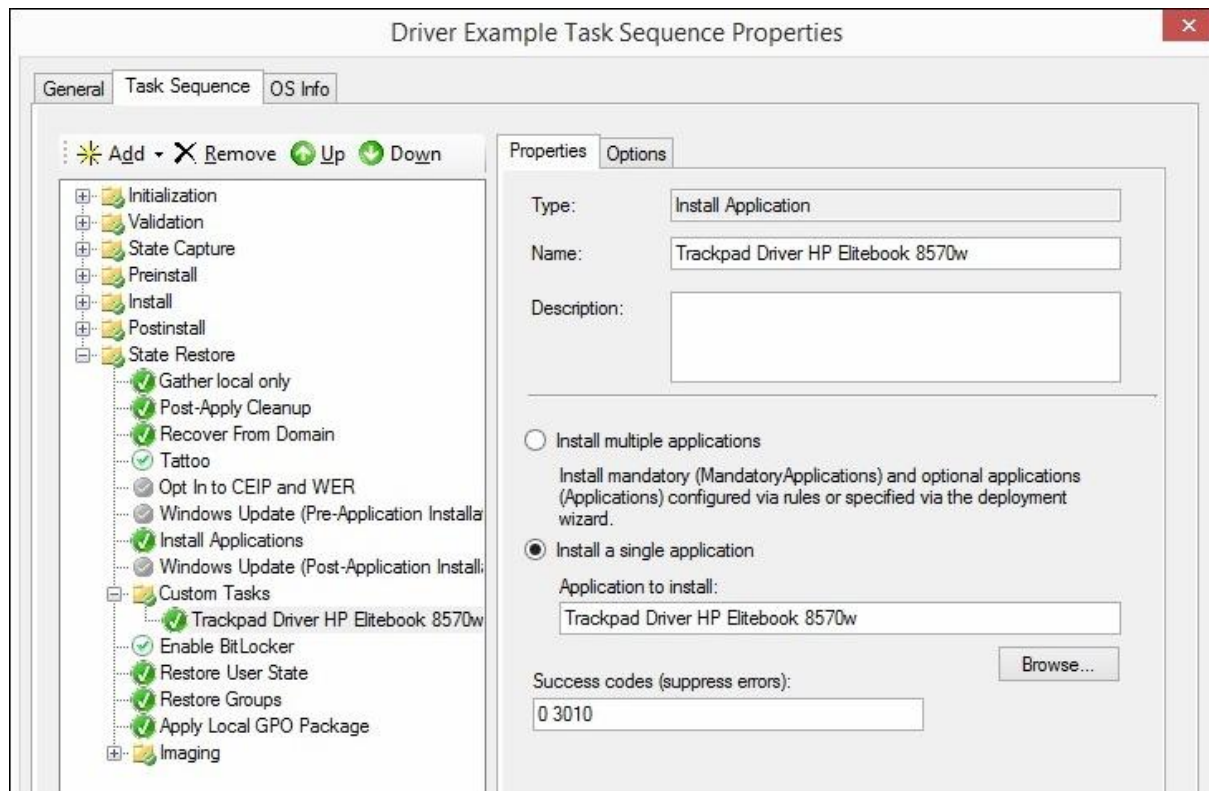
3. Configure the **Properties** page as outlined in the following screenshot:



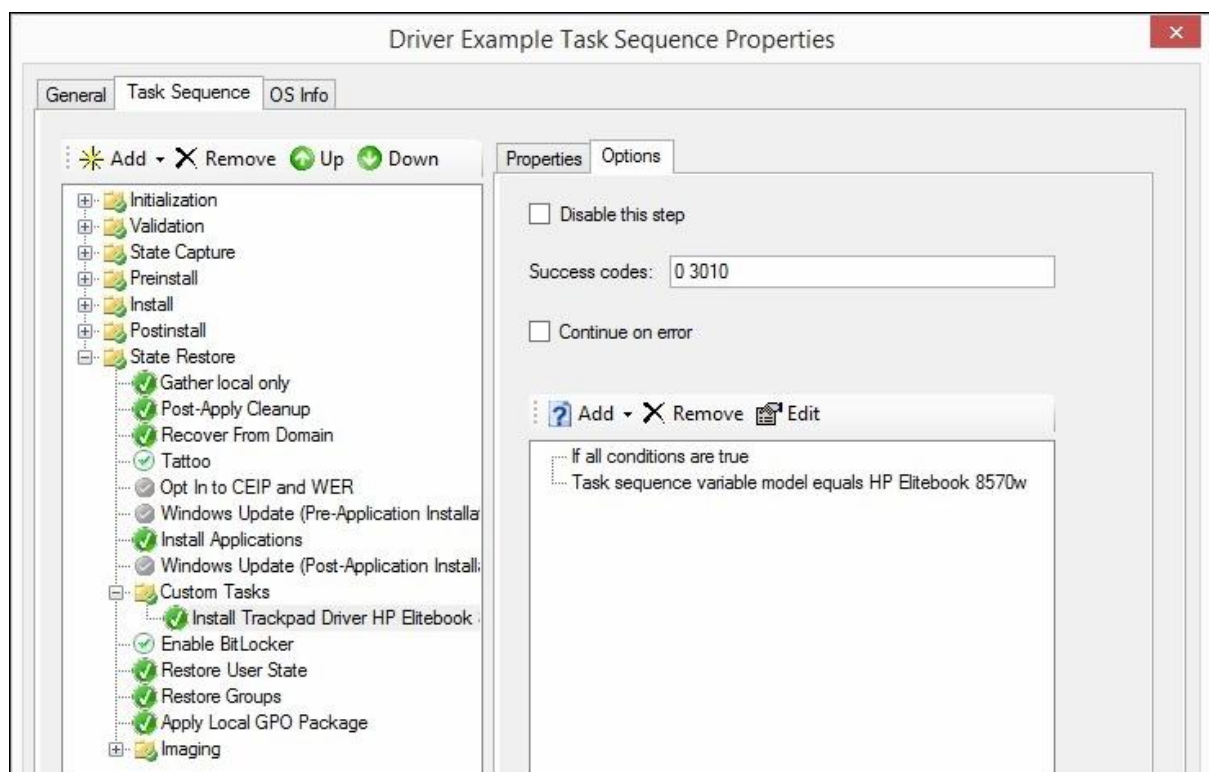
Now, you can be sure that when PnP is searching for and installing drivers, the drivers are only coming from the correct folder for this model and you have full control over the process instead of hoping things just work out on their own.

Drivers as applications

Unfortunately, all drivers don't adhere to the usual `.inf/.sys` format, and in this case, the MDT driver injection method will not apply to these drivers. What I'm referring to is drivers that install via a `.exe` format and cannot be extracted to the usual `.inf/.sys` format that we are used to. In this case, we need to treat these drivers as applications. Let's say, for example, I have a trackpad driver for a particular laptop that I need to install and the driver installation program must be installed via a `.exe` program. I can import a new application into MDT. The command line must be configured to perform a silent install and you should also check the box in the application's **Properties** to hide the application so that it will not show up during the LiteTouch Deployment wizard. We will add the driver to our **Task Sequence** as follows:



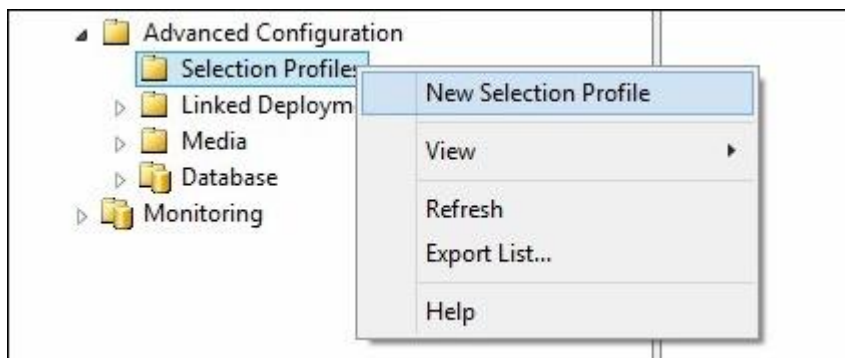
Now, to ensure that this application/driver only gets installed on the machines that we need to install it on, we can take advantage of conditional statements to make this happen. This way, the task sequence will still apply to any hardware, but will only execute this step on the hardware the driver is targeted to. We can accomplish this as outlined in the following screenshot:



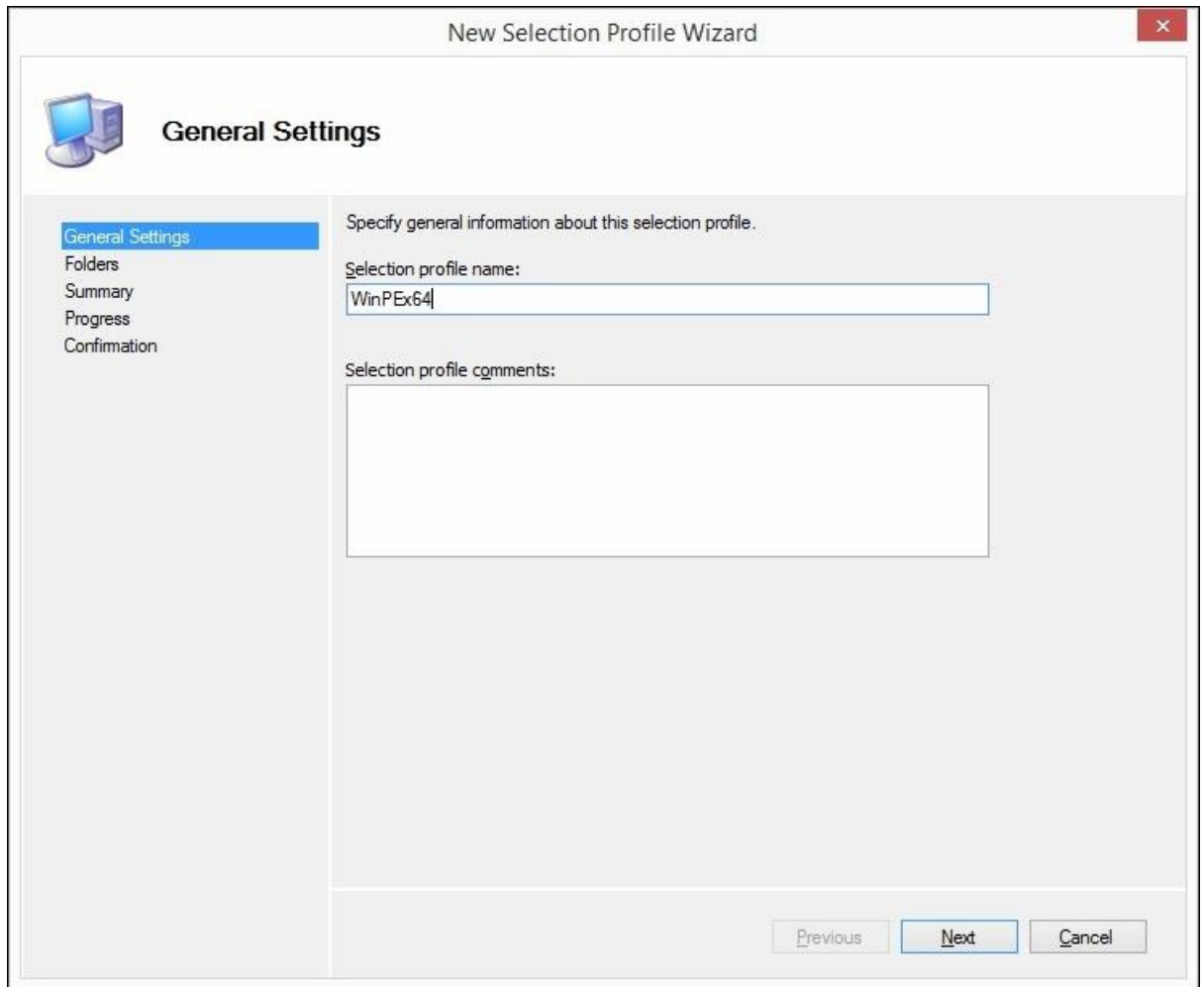
Win PE drivers

By default, MDT will inject all network adapter and mass storage drivers into the LiteTouchPE_x86/LiteTouchPE_x64 ISO/WIM file(s). If you want to specifically control which drivers are injected, you can create separate folders under your Out-of-Box Drivers node for WinPE_x86 and WinPE_x64 drivers, then create selection profiles for each of these folders. Navigate to the **Properties** of your deployment share | **Windows PE** tab | **Drivers and Patches** tab, and select the **Selection profile** for each architecture. The steps to inject specific drivers are as follows:

1. The first step would be to create a selection profile to point to the specific folder where you have imported your Win PE drivers. You can accomplish this as outlined in the following screenshot:




2. In the next step, name the selection profile. In this example, we are naming it [WinPEx64](#), as shown in the following screenshot:



The screenshot shows a Windows-style dialog box titled "New Selection Profile Wizard". The window has a standard title bar with a close button (X) in the top right corner. Inside the window, there is a small icon of a computer monitor and tower on the left, followed by the text "General Settings". Below this, a vertical list of steps is shown: "General Settings" (highlighted with a blue background), "Folders", "Summary", "Progress", and "Confirmation". The main area of the dialog is titled "Specify general information about this selection profile." and contains two input fields. The first is labeled "Selection profile name:" and has a text box containing "WinPEx64". The second is labeled "Selection profile comments:" and has a larger, empty text box. At the bottom right of the dialog, there are three buttons: "Previous" (disabled), "Next" (active/highlighted), and "Cancel".

New Selection Profile Wizard

 **General Settings**

General Settings
Folders
Summary
Progress
Confirmation

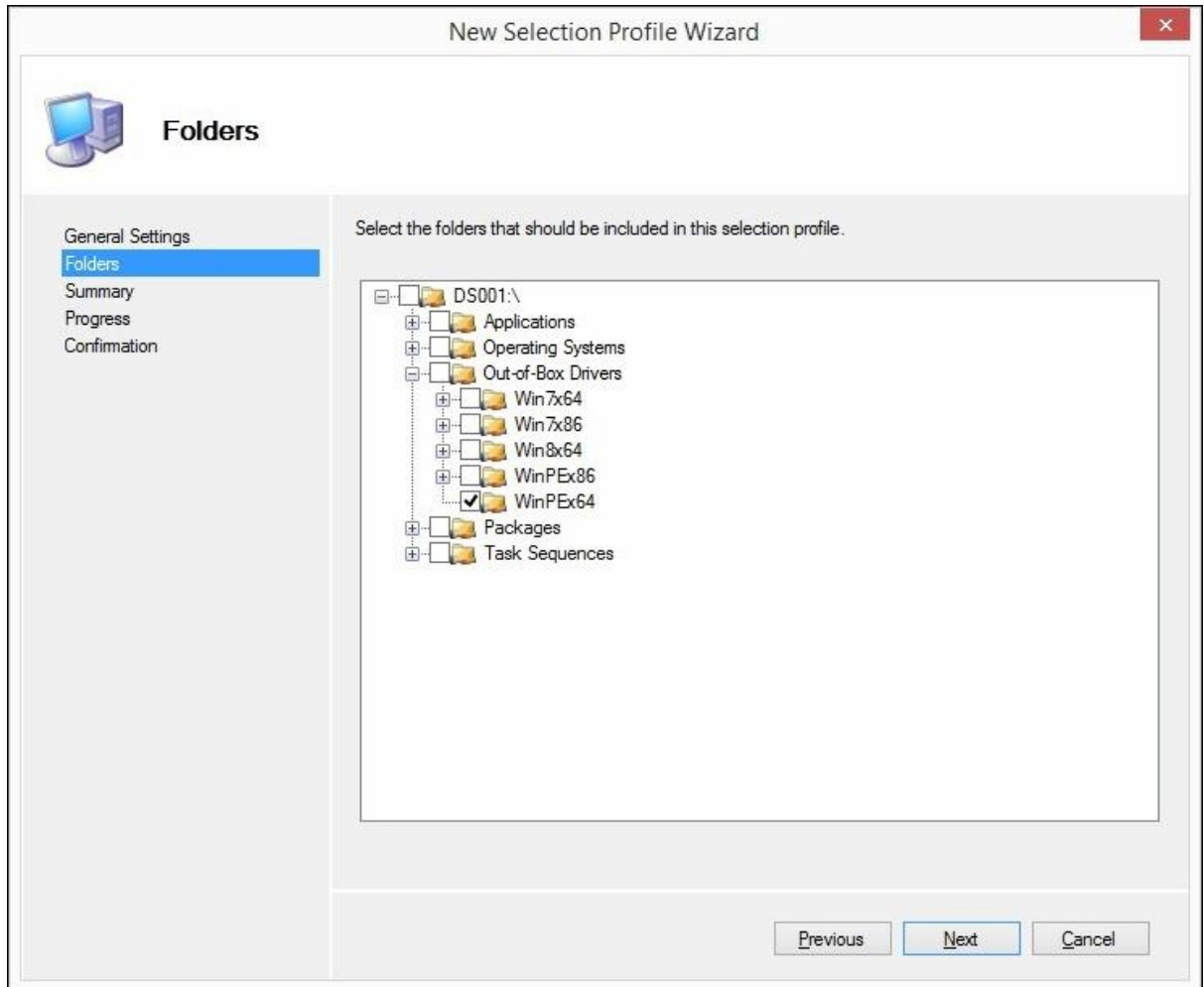
Specify general information about this selection profile.

Selection profile name:
WinPEx64

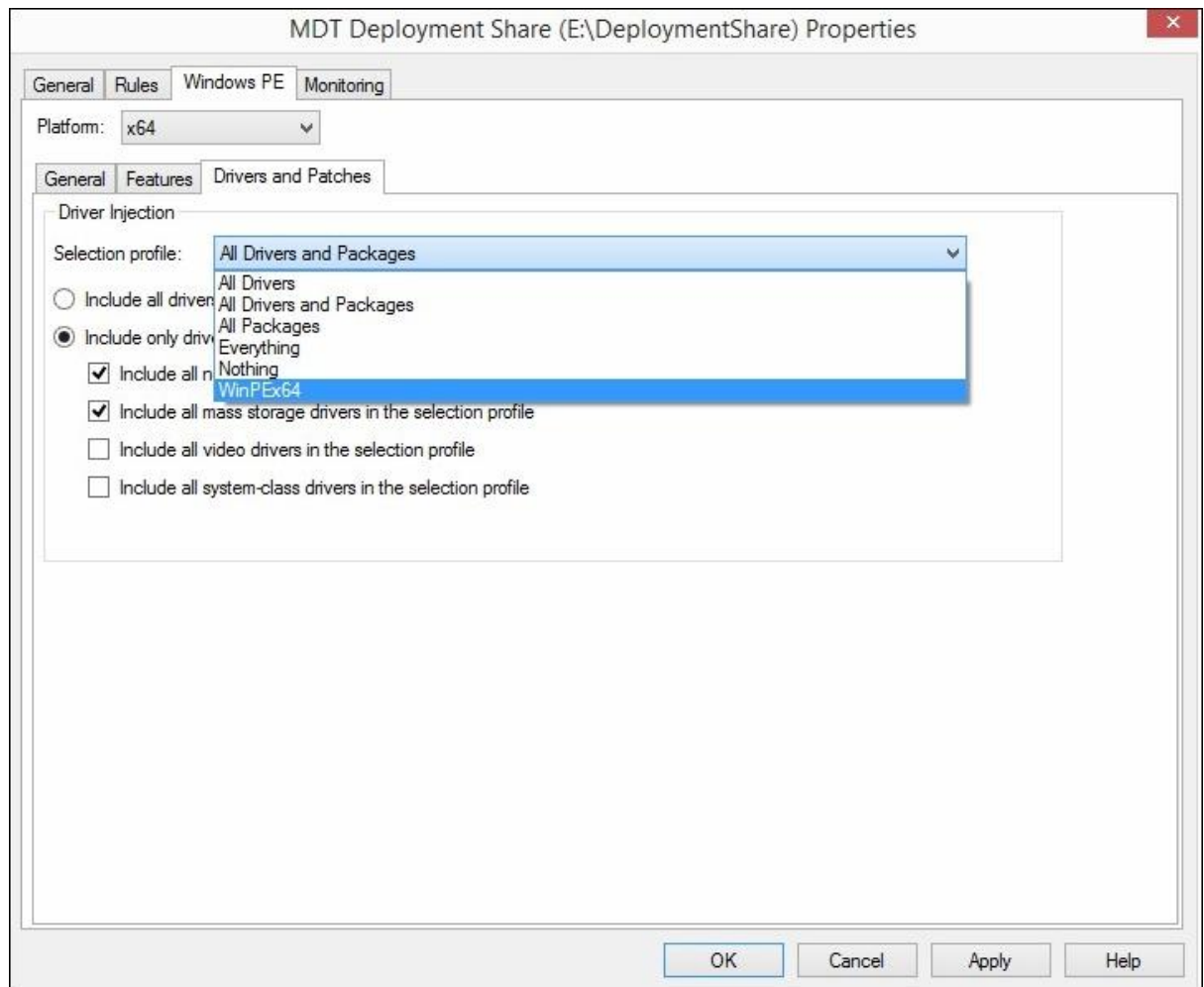
Selection profile comments:

Previous Next Cancel

3. We will then only select the folder that we want to include in the selection profile, as shown in the following screenshot:



4. Once we have created our selection profile, we will then go to the **Properties** of our deployment share and then to the **Windows PE** tab, choose the **Platform (x64)** from the drop-down list, and then go to the **Drivers and Patches** tab. From the **Selection profile** drop-down list, choose the selection profile that you created for WinPEx64, as shown in the following screenshot:



Now, when you update your deployment share to generate new LiteTouch media, it will only inject drivers into Windows PE from the folder that you specified in your selection profile. This will give you more control over which drivers actually get injected into your LiteTouch media.

NOTE

Windows 10

All the concepts shown in this chapter are still valid for Windows 10, but please pay attention to the following points:

- While the plan is definitely to enforce the new driver signing model (signed by Microsoft directly), current builds (1507 and 1511) allow the legacy driver signing model by default.
- This was to work around upgrade challenges that will be removed in the Redstone timeframe.
- Moving forward, drivers will continue to be migrated and loaded successfully (even if not signed *properly*), but all new device drivers installed will require the new model.

- When considering a Windows 10 deployment the same driver considerations apply as any other Windows installation. In some specific device installations such as Surface Pro 4 devices or Surface Book, driver packages need to be installed to perform device firmware updates. Therefore Microsoft and OEM provided drivers are perhaps more important to keep up to date in Windows 10 than in other operating systems.