

# Contents

Manufacture

    Desktop manufacturing

        Deployment guides and walkthroughs

            OEM deployment of Windows 10 for desktop editions

            Plan your Windows deployment

            Get the tools you need for this lab

            OEM deployment lab

            Sample scripts

        System builder deployment of Windows 10 for desktop editions

            Manufacturing Windows Engineering Guide

            Windows 10 in S mode

                Planning a Windows 10 in S mode deployment

                Manufacturing environment

                Manufacturing mode

                Enable S mode

                Windows 10 in S mode deployment lab

            Windows 10 IoT Enterprise

                Get the tools you need to create an IoT Enterprise image

                Lab 1a: Create a basic image

                Lab 1b: Customize a reference device in Audit mode

                Lab 2: Device lockdown features

                Lab 3: Configure policy settings on IoT Enterprise devices

                Lab 4: Sysprep, capture, and deploy

                Lab 5: Configure Shell Launcher or Assigned Access

    Boot and install Windows

        Install Windows from a USB Flash Drive

        Install Windows from a DVD

        Boot to WinPE

            Download WinPE (Windows PE)

- [Create bootable WinPE media](#)
- [WinPE: Install on a hard drive \(Flat Boot or Non-RAM\)](#)
- [Boot to UEFI Mode or legacy BIOS mode](#)
- [Deploy a custom image](#)
- [Deploy Windows with a VHD \(Native Boot\)](#)
- [Windows Setup installation process](#)
  - [Install Windows 10 using a previous version of Windows PE](#)
  - [Windows Setup automation overview](#)
  - [Automate Windows Setup](#)
  - [Windows Setup: Installing using the MBR or GPT partition style](#)
  - [Add languages to Windows Setup](#)
  - [Answer files \(unattend.xml\)](#)
  - [Run custom actions during a feature update](#)
  - [Add a custom script to Windows Setup](#)
- [Modify an image](#)
  - [Audit Mode](#)
    - [Run Audit Mode in the factory](#)
    - [Boot Windows to Audit Mode or OOB](#)
    - [Enable and disable the built-in administrator account](#)
  - [Sysprep](#)
    - [Sysprep process overview](#)
    - [Sysprep \(Generalize\) a Windows installation](#)
    - [Use answer files with sysprep](#)
    - [Sysprep command-line options](#)
    - [Sysprep support for server roles](#)
  - [Mount and modify a Windows image Using DISM](#)
    - [Repair a Windows image](#)
    - [Configure a Windows repair source](#)
  - [Capture and apply an image](#)
    - [WIM vs. VHD vs. FFU: comparing image file formats](#)
    - [Capture and apply a Full Flash Update \(FFU\) image](#)
    - [Capture and apply a WIM image](#)

Capture and apply Windows using a single WIM file

Capture and apply Windows, system, and recovery partitions

Create a WIM for multiple architecture types

Split a Windows image file (.wim) to span across multiple DVDs

Append a volume image to an existing image

Create a WIM data image

Boot to VHD (Native Boot): Add a Virtual Hard Disk to the Boot Menu

Hard drives and partitions

UEFI/GPT-based hard drive partitions

BIOS/MBR-based hard drive partitions

Configure more than four partitions on a BIOS/MBR-Based hard disk

Configure multiple hard drives

BitLocker drive encryption

Hard disk location path format

Windows and GPT FAQ

Customize

Apps

Siloed provisioning packages (SPPs)

Create a provisioning package with Windows desktop applications

Sideload apps with DISM

Preinstall apps Using DISM

Export or import default application associations

Microsoft .NET Framework 3.5 deployment considerations

    Deploy .NET Framework 3.5 by using Group Policy Feature on Demand setting

    Deploy .NET Framework 3.5 by using DISM

    Enable .NET Framework 3.5 by using Windows PowerShell

    Enable .NET Framework 3.5 by using the Add Roles and Features Wizard

    .NET Framework 3.5 deployment errors and resolution steps

Battery life

Managing battery life and power consumption overview

Set the default power plan

Create a custom power plan

Fine-tune a custom power plan

Test battery life and power consumption

## Drivers

Maintain driver configurations when capturing a Windows image

Add a driver online in Audit Mode

Add and remove drivers to an offline Windows image

Add device drivers to Windows during Windows Setup

## Configuration and settings

Add updates to a Windows image

Customize the default user profile by using CopyProfile

Work with product keys and activation

Change the Windows image to a higher edition

High DPI Support for IT Professionals

High DPI and Windows 8.1

Fixing blurry text in Windows for IT Professionals

High DPI projection and multi-monitor configurations

DPI-related APIs and registry settings

## Features

Features On Demand

Available Features on Demand

Enable or disable windows features Using DISM

Configure a Trusted Image Identifier for Windows Defender

Configure Windows System Assessment Test scores

Add or remove packages offline using DISM

## OOBE

Oobe.xml settings

How Oobe.xml works

## Secure Boot

Windows Secure Boot key creation and management guidance

Secure Boot key generation and signing using HSM (example)

UEFI validation Option ROM validation guidance

Disabling Secure Boot

[Secure Boot isn't configured correctly: troubleshooting](#)

[BCD system store settings for UEFI](#)

[Validating Windows UEFI firmware update platform functionality](#)

## Localize

[Add languages to Windows images](#)

[Available languages](#)

[Language and region Features on Demand](#)

## International settings

[Default input profiles \(input locales\)](#)

[Default time zones](#)

[Keyboard identifiers and input method editors](#)

## Optimize

[Compact OS, single-instancing, and image optimization](#)

[Manage the component store](#)

[Determine the actual size of the WinSxS folder](#)

[Clean up the WinSxS folder](#)

[Reduce the size of the component store in an offline Windows image](#)

[Take inventory of an image or component using DISM](#)

## Prepare recovery tools

[Windows Recovery Environment \(Windows RE\)](#)

[Customize Windows RE](#)

[Add a custom tool to the Windows RE boot options menu](#)

[Add a hardware recovery button to start Windows RE](#)

[Deploy Windows RE](#)

## Push-button reset

[How push-button reset features work](#)

[Recovery components](#)

[Deploy push-button reset features](#)

[Deploy push-button reset features using auto-apply folders](#)

[Add extensibility scripts to push-button reset](#)

[Frequently-asked questions \(FAQ\)](#)

[Validation scenarios](#)

## Bare metal reset/recovery

Create recovery media while deploying new devices

Enable your users to create recovery media

## REAgentC command-line options

ResetConfig XML reference

WinREConfig XML reference

Windows RE troubleshooting features

## Deployment Tools Reference

### DISM - Deployment Image Servicing and Management

What is DISM?

Use DISM in Windows PowerShell

Command-line options

Image management (.wim, .ffu, .vhdx)

Global options for command-line syntax

Operating system packages (.cab or .msu)

Provisioning packages (.ppkg)

App packages (.appx or .appxbundle)

App patches (.msp)

Default application associations

Languages and international

Capabilities packages

Windows editions

Drivers (.inf)

Unattended answer files (unattend.xml)

Windows PE

Operating system uninstall

DISM reference

Configuration list and WimScript.ini files

Best practices

Supported platforms

DISM API

Windows PE (WinPE)

## What's new in Windows PE

- WinPE: Add packages (Optional Components Reference)
- WinPE: Mount and customize
- WinPE: Adding PowerShell support to Windows PE
- WinPE: Store or split images to deploy Windows using a single USB key
- WinPE: Identify drive letters with a script
- WinPE: Storage Area Network (SAN) policy
- WinPE Network drivers: Initializing and adding drivers
- WinPE: Create apps
- WinPE: Debug apps
- CopyPE command-line options
- MakeWinPEMedia command-line options
- DrvLoad command-line options
- Winpeshl.ini reference: Launching an app when WinPE starts
- WPEInit and startnet.cmd: Using WinPE Startup Scripts
- WPEUtil command-line options

## Windows Setup

- Supported platforms and cross-platform deployments
- Add languages to Windows Setup
- Scenarios and best practices
- Windows Setup command-line options
- States
  - Edition configuration and product ID files (EI.cfg and PID.txt)
  - Log files and event logs
  - Configuration passes
    - How Configuration Passes Work
    - auditSystem
    - auditUser
    - generalize
    - offlineServicing
    - oobeSystem
    - specialize

windowsPE

Deployment troubleshooting and log files

Command-Line Tools

BCDBoot command-line options

Repair the boot menu on a dual-boot PC

BCDEdit command-line options

Bootsect command-line options

Oscdimg command-line options

Mobile manufacturing

Mobile deployment and imaging

Prepare for Windows mobile development

Create mobile packages

Adding mobile packages

Primary elements and attributes of a package project file

Specifying components in a package project file

Specifying files and registry entries in a package project file

Command-line arguments for package generator

Merging packages before imaging

Merging packages using FeatureMerger

Windows Standard Packaging Configuration (WSPC) requirements for retail images

Configure the Start layout

Part 1: Classic mobile deployment

Configure customization settings

Add a package to an OEM manifest file

Configure the OEMInput file

Build a mobile image using ImgGen

Sign a mobile image

Flash an image to a mobile device

Part 2: Mobile deployment using Windows Provisioning

Use the Windows ICD UI to customize and build a mobile image

Use the Windows ICD CLI to customize and build a mobile image

Manufacturing Mode

[Create a custom manufacturing profile package](#)

[Create a custom manufacturing profile package with USBFN settings](#)

[Define a service that only runs in Manufacturing Mode](#)

[Create a full operating system manufacturing profile](#)

[Detect Manufacturing Mode](#)

[Enable or Disable Manufacturing Mode](#)

[Optional features for Manufacturing Mode](#)

[Boot mode management UEFI protocol](#)

[EFI\\_BOOT\\_MODE\\_INFO enumeration](#)

[EFI\\_BOOT\\_MODE\\_MGMT\\_PROTOCOL.GetBootModeInfo](#)

[EFI\\_BOOT\\_MODE\\_MGMT\\_PROTOCOL.SetBootModeInfo](#)

[Microsoft Manufacturing OS](#)

[MMOS image definition](#)

[Flash MMOS to the device](#)

[Working with WIM MMOS images](#)

[Creating a secure MMOS WIM image](#)

[Develop MMOS test applications](#)

[Deploy and test a user-mode test application in MMOS](#)

[Determine if MMOS is running](#)

[Manufacturing test environment supported APIs](#)

[Manufacturing Mode Phone Call Testing APIs](#)

[MfgPhoneDial](#)

[MfgPhoneEndCall](#)

[MfgPhoneGetSimLineCount](#)

[MfgPhoneGetSimLineDetail](#)

[MfgPhoneGetSpeaker](#)

[MfgPhoneInitialize](#)

[MfgPhoneSetSimLineEventNotifyCallback](#)

[MfgPhoneSetSpeaker](#)

[MfgPhoneUninitialize](#)

[MFGPHONE\\_CALLSTATUS](#)

[MFGPHONE\\_LINESYSTEMTYPE](#)

MFGPHONE\_REGISTRATIONSTATE

MFGPHONE\_SIMLINEDETAIL

MFGPHONE\_SIMSTATE

Access the touch interface in MMOS

Calling SetScreenOff to enter connected standby

Resetting a device during manufacturing

Wi-Fi manufacturing API

    WlanMTEEnumAdapters

    WlanMTEOpenHandle

    WlanMTECloseHandle

    WlanMTERegisterCallbackHandler

    WlanMTEDeRegisterCallbackHandler

    WlanMTEGetVendorInfo

    WlanMTEResetAdapter

    WlanMTEQueryMacAddress

    WlanMTEQueryPhyTypes

    WlanMTEStartSelfTest

    WlanMTEQuerySelfTestResult

    WlanMTERxSignal

    WlanMTETxSignal

    WlanMTEQueryADC

    WlanMTESetData

    WlanMTEQueryData

    WlanMTESleep

    WlanMTEAwake

Adding Wi-Fi manufacturing test support to the OID interface

    Reporting operating mode capabilities

    Supporting updated OID behavior in manufacturing mode

    Supporting existing OID commands in manufacturing mode

    Supporting new OID commands for manufacturing mode

    Supporting new callbacks for manufacturing mode

Flashing tools

Developing custom OEM flashing tools

- - Flashing security requirements
  - FFU image format
  - Implementing image integrity validation in custom flashing tools
  - Field service scenarios
- Using a host PC to reboot a device to flashing mode and get version information
- Disabling the initial setup process
- Reset protection
- Building and flashing mobile images
  - Build a mobile image using Windows ICD
  - Build a mobile image using ImgGen.cmd
  - Build a mobile image using a hybrid method
  - Define the image using OEMInput and feature manifest files
    - OEMInput file contents
    - Optional features for building mobile images
    - Feature manifest file contents
    - Create a feature and include it in an image
    - Adding a driver to a test image
    - Feature groupings and constraints
    - Set device platform information
  - Sign a full flash update (FFU) image
  - Use the flashing tools provided by Microsoft
    - IUTool.exe: Update packages on a device
      - IUTool error codes
      - Update packages on a device and get package update logs
      - Update packages in an .FFU image file
  - IoT Core manufacturing
    - OEM license requirements
    - Create and install packages
    - Windows 10 IoT Core services
    - IoT Core manufacturing guide
      - Get the tools needed to customize Windows IoT Core
      - Lab 1a: Create a basic image

[Lab 1b: Add an app to your image](#)

[Lab 1c: Add file\(s\) and registry settings to an image](#)

[Lab 1d: Add a provisioning package to an image](#)

[Lab 1e: Add a driver to an image](#)

[Lab 1f: Add Win32 services to an image](#)

[Lab 1g: Build a retail image](#)

[Lab 2: Create a board support package](#)

[BSP for hardware](#)

[IoT device layout](#)

[OS customizations](#)

[IoT Core feature list](#)

[IoT Core Add-ons](#)

[IoT Core Add-ons Powershell commands](#)

[IoT Core image wizard](#)

[Update the time server](#)

[Add a recovery mechanism](#)

[Frequently Asked Questions](#)

[Troubleshooting](#)

[Create Windows Universal OEM Packages](#)

[Windows Universal OEM Package Schema](#)

# Manufacture

1/18/2019 • 2 minutes to read

## Purpose

Use the manufacturing tools to deploy your Windows [customizations](#) to new Windows 10 devices. Learn how to:

- Combine your customizations, plus languages, drivers, apps and more, into new Windows images.
- Modify these images either from a manufacturing mode for a familiar Windows experience, or from a command line for quicker changes that can be automated and scripted.
- Install images onto new devices. Choose whether to use compression to balance disk space versus device performance. Use flashing tools to speed up the final manufacturing processes
- Capture your customizations into the recovery tools, helping your customers get back up to speed quickly.

TOPIC	DESCRIPTION
<a href="#">OEM deployment of Windows 10 for desktop editions</a>	This guide is intended for OEMs, and applies to Windows 10 for desktop editions (Home, Pro, Enterprise, and Education). IT professionals using this guide should have prior knowledge of Windows basic administration and troubleshooting.
<a href="#">System builder deployment of Windows 10 for desktop editions</a>	Learn how to deploy Windows 10 desktop, including online and offline customizations, and optional steps for specific scenarios. This guide is intended to help system builders with both 64-bit and 32-bit configurations.
<a href="#">What's new in Windows manufacturing</a>	Learn what new features are available.
<a href="#">Desktop manufacturing</a>	Technical reference for Windows 10 for desktop editions
<a href="#">Mobile manufacturing</a>	Technical reference for Windows 10 Mobile
<a href="#">IoT Core manufacturing</a>	Technical reference for Windows 10 IoT Core

# Desktop manufacturing

5/15/2019 • 2 minutes to read

After you've learned how to design, develop, and customize Windows images, you can use the tools in the Windows ADK to manufacture and deploy Windows images to new PCs and devices.

## In This Section

CONTENT TYPE	REFERENCES
<b>Getting started</b>	<ul style="list-style-type: none"><li>• <a href="#">Download the Windows ADK</a></li><li>• <a href="#">Desktop manufacturing guide</a></li><li>• <a href="#">System builder deployment of Windows 10 for desktop editions</a></li><li>• <a href="#">Manufacturing Windows Engineering Guide (WEG)</a></li></ul>
<b>Deployment options</b>	<ul style="list-style-type: none"><li>• <a href="#">UEFI Firmware</a></li><li>• <a href="#">Hard Drives and Partitions</a></li><li>• <a href="#">VHD (Native Boot)</a></li><li>• <a href="#">Secure Boot</a></li><li>• <a href="#">Device Drivers</a></li><li>• <a href="#">Language Packs</a></li><li>• <a href="#">Features On Demand</a></li><li>• <a href="#">More deployment options</a></li></ul>
<b>Tools</b>	<ul style="list-style-type: none"><li>• <a href="#">Deployment Image Servicing and Management (DISM)</a></li><li>• <a href="#">System Preparation (SysPrep)</a></li><li>• <a href="#">Windows PE (WinPE)</a></li><li>• <a href="#">Windows Recovery Environment (Windows RE)</a></li><li>• <a href="#">Windows Setup</a></li><li>• <a href="#">More command-line tools</a></li></ul>

# OEM deployment guides and walkthroughs for Windows 10

1/18/2019 • 2 minutes to read

## In this section

GUIDE	DESCRIPTION
<a href="#">OEM deployment of Windows 10 for desktop editions</a>	End-to-end desktop manufacturing lab for OEMs
<a href="#">System builder deployment of Windows 10 for desktop editions</a>	End-to-end lab for system builders
<a href="#">OEM Windows Desktop Deployment and Imaging Lab</a>	Collection of walkthroughs for OEM deployments
<a href="#">Manufacturing Windows Engineering Guide</a>	Roadmap for OEMs and ODMs of the ideal manufacturing process for Windows 10 devices, with guidance for potential pitfalls and opportunities to streamline the process.

# OEM deployment of Windows 10 overview

1/18/2019 • 2 minutes to read

## In this section

GUIDE	DESCRIPTION
<a href="#">Plan your Windows deployment</a>	Information to consider when planning your Windows deployment
<a href="#">Get the tools you need to complete the Windows Desktop Deployment and Imaging Lab</a>	Gather the tools you need to complete the lab
<a href="#">OEM deployment of Windows 10 for desktop editions</a>	End-to-end desktop manufacturing lab for OEMs
<a href="#">Sample scripts</a>	Get the sample scripts that enable faster deployment of Windows 10

# Planning: Customizing reference images for different audiences

1/18/2019 • 2 minutes to read

Instead of having one device design that tries to fit everyone, Windows image management tools help you tailor device designs to meet the specific needs of various customers.

To get started, choose a hardware design that targets a specific audience, market, or price point. Build base images for this design and test it. Next, modify the base images to create designs for different audiences, including branding, logos, languages, and apps.

## Device types

Consider creating separate designs for different device types, such as low-cost or performance laptops, or low-cost or performance desktops. Each of these styles has different sets of critical differentiators, such as battery life or graphics performance.

Although Windows includes base drivers for many common devices, some hardware requires specialized device drivers that must be installed and occasionally updated.

Many drivers are designed to be installed offline without booting the Windows image.

Use Windows Assessment tools to make sure that the apps and hardware that you're installing can perform well in a variety of circumstances.

## Architecture

If you plan to build devices with both 64-bit and 32-bit (x86) chipsets and architectures, you'll need separate base images. You'll also need different versions of drivers, packages, and updates.

## Retail customers and business customers

If you're building designs for both retail and business customers, you can start with a single base edition such as Windows 10 Home or Windows 10 Pro, and then later upgrade it to a higher edition such as Windows 10 Enterprise, as needed. Once you've built a higher edition, however, you can't downgrade it to the lower edition. For more info, see [Windows Upgrade Paths](#).

If you're building devices to sell to retail customers, you'll need to meet a set of minimum requirements. For info, see the Licensing and Policy guidance on the [Device Partner Center](#).

If you're building devices for businesses, you'll have fewer restrictions. IT professionals can customize their devices in all sorts of ways. However, you should consider the implications of IT policies, as well as customer needs such as migrating data, activating security tools, and managing volume license agreements and product keys.

## Regions

Consider creating different base images for different regions.

The resource files for Windows and other apps like Microsoft Office can be large - some resources like localized handwriting and speech recognition resources are several hundred megabytes.

To save drive space, we've split up the language packs. This can help you preload more languages for your

customers or save space on your image. For example, to target a large region, you may preload the basic language components such as text and user interface files for many areas within the region, but only include the handwriting recognition for devices with pens, or only include voice and speech tools for Cortana on devices with integrated microphones. Users can download these components later as needed.

# Get the tools needed to customize Windows

7/11/2019 • 3 minutes to read

Here's what you'll need to start testing and deploying devices:

## PCs

Here's how we'll refer to them:

- **Technician PC:** Your work PC. This PC should have at least 15GB of free space for installing the [Windows Assessment and Deployment Kit \(Windows ADK\)](#) and working with Windows images.

We recommend using Windows 10 for this PC. The minimum requirement is Windows 7 SP1, though this requires additional tools or workarounds for tasks such as running PowerShell scripts and mounting .ISO images.

For most tasks, you can use either an x86 or x64 PC. If you're creating x86 images, you'll need an x86-based PC (or virtual machine) for a one-time task of [generating a catalog file](#) when you modify your answer file with Windows SIM.

- **Reference PC:** A test PC or tablet that represents all of the devices in a single model line; for example, the *Fabrikam Notebook PC Series 1*. This device must meet the Windows 10 minimum hardware requirements.

You'll reformat this device as part of the walkthrough.

## Storage

- One **USB key** that you'll format with two partitions. The USB key must be at least 16GB. This drive will be reformatted, so save your data off of it first. It shouldn't be a Windows-to-Go key or a key marked as a non-removable drive.

Here's how to format your USB key with two partitions. The first partition will be formatted FAT32 and called *WinPE*. We'll use this partition to boot to WinPE. The second partition will be called *USB-B*. We'll use this drive to store your Windows images and other customizations that we'll use throughout this lab.

From the Command Prompt:

```
diskpart
list disk
select <disk number>
clean
rem === Create the Windows PE partition. ===
create partition primary size=2000
format quick fs=fat32 label="WinPE"
assign letter=P
active
rem === Create a data partition. ===
create partition primary
format fs=ntfs quick label="USB-B"
assign letter=Q
list vol
exit
```

#### **NOTE**

You can use two separate USB drives for this lab. If you decide to use two drives, format one as FAT32 and one as NTFS.

## Software

Create a folder called on the technician PC called `C:\temp\lab`. When working with images, use local storage rather than external sources like network shares or removable drives. This reduces the risk of interrupting the build process from a temporary network issue or from disconnecting the USB device.

To complete this guide, get the recommended downloads in this section from <https://www.microsoft.com>.

The version numbers of the Windows ADK, the Windows image you're deploying, and the languages and features you're adding must match.

This lab only gives examples for building 64-bit systems. If you're working with a 32-bit image, replace the mentions of 64-bit with 32-bit.

Download the following, and place the files into `C:\temp\lab`.

### **The most recent version of Windows 10**

Windows Home 10, 64-bit English OPK

### **Customizations: Windows updates, languages, features, apps, and Microsoft Office**

Win 10 32/64 MultiLang OPK LangPackAll/LIP

Win 10 32/64 MultiLang OPK Feat on Demand

Win 10 32/64 MultiLang OPK App Update

Latest Office OPK

2 Microsoft Store signed apps, if you have them

### **Windows Assessment and Deployment Kit (ADK) for Windows 10**

Download the version of [Windows ADK for Windows 10](#) that matches the version of Windows 10 you are working with.

If you're using the ADK for Windows 10, version 1809, you'll also have to download and install the [WinPE addon for the ADK](#).

### **Drivers**

We also discuss how to add hardware drivers and other Windows apps in this guide. If you need to add additional drivers to your image, you'll need to contact your hardware or software manufacturers.

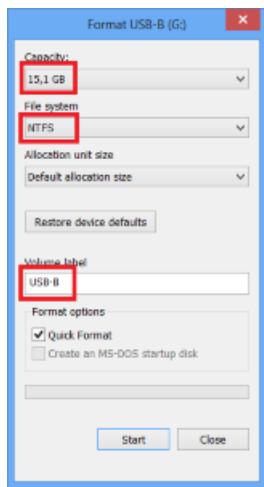
### **Sample files**

Download the lab samples from [USB-B.zip](#), and extract the files to the *USB-B* drive.

- The deployment steps in this guide depend on the sample configuration files included in USB-B. You can download USB-B.zip from the Microsoft Download Center.

- The contents of the configuration files included in USB-B are examples that you may change according to your branding and manufacturing choices. However, file names and hierarchy of the folders and files must be the same as demonstrated below in order to align your deployment procedure with this guide.

**Note:** *USB-B* has to be formatted as NTFS.



## Product keys

Get the default product keys for each Windows version from the Kit Guide Windows 10 Default Manufacturing Key OEM PDF, which is on the ISO with the Windows image.

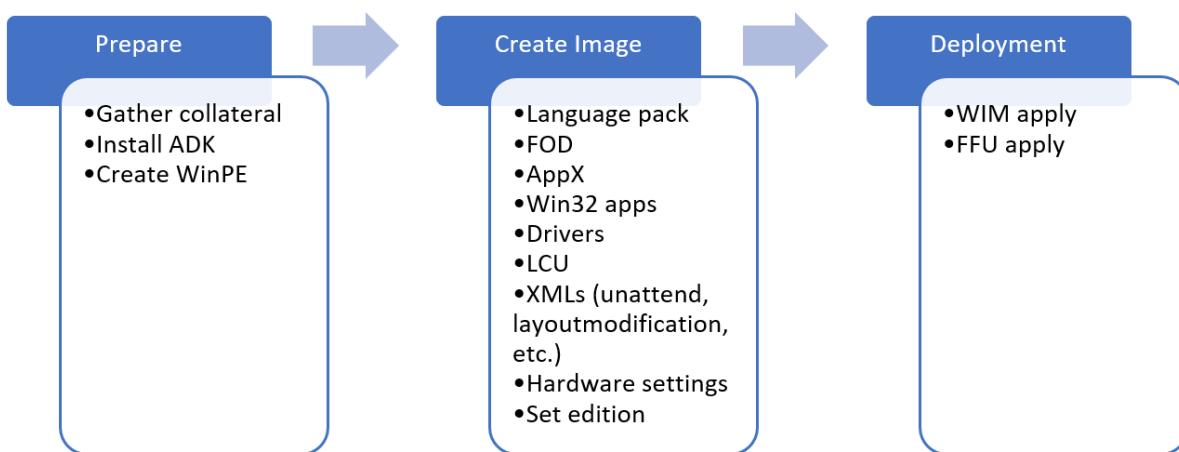
Get the distribution product keys that match the Windows 10 image.

# OEM Deployment of Windows 10 for desktop editions

7/11/2019 • 62 minutes to read

Getting ready to build and test Windows 10 desktop PCs? This lab shows you the steps to make and deploy Windows images. We'll show you how to use the tools and commands to setup an end-to-end deployment. The commands can be scripted, helping you quickly customize new images for specific markets to meet your customers' needs.

We'll walk you through the process of building a customized Windows deployment. Here's what we'll cover:



We'll start by preparing your environment, then we'll cover the steps for:

- Preparing and mounting a WinPE image
- Adding packages
- Adding drivers
- Creating WinPE media

Next we'll move onto customizing your Windows image. We'll start with [offline customizations](#) to a mounted Windows image, where we'll cover:

- Adding Drivers
- Adding Languages
- Adding Updates
- Reinstalling inbox apps
- Preinstalling Microsoft Office
- Adding tiles to the Start Layout
- Setup OOBE to display a custom EULA
- Configuring and using answer files to customize Windows Setup

We'll finish customizing the Windows image by [deploying your image to a PC and then booting into Audit mode](#) and finish making changes, including:

- Making changes in Audit mode
- Preparing Push Button Reset

Finally, we'll [Finalize and Capture your image, verify everything works, and prepare your image for deployment](#).

- Finalizing the image

Let's get started!

## Prepare your lab environment

### NOTE

If you haven't gathered the files you need to go through this lab, see [Get the tools you need for OEM deployment of Windows 10 for desktop editions.](#)

At this point, you should have your tools ready to go. At this point, you should have:

A USB drive, formatted with two partitions, with the following on the NTFS, partition (O):

- The extracted folder structure and files from USB-B

A technician PC that has:

- A folder called `C:\temp\lab` with the following media downloaded:
  - Windows installation media
  - The most recent version of the Microsoft Office OPK
  - Windows Features on Demand ISOs
  - Windows Language pack ISO
  - OPK App Update or Inbox Apps ISO
  - Windows ADK installer
  - WinPE ADK Add-on (if using the ADK for Windows 10 version 1809, or later)
  - Drivers for your image (if needed)

Let's setup your lab.

### Install the Windows ADK for Windows 10

The Windows ADK is a collection of tools that enables you to manage and deploy custom Windows installations to new computers.

### IMPORTANT

Use the matching version of ADK for the images being customized. For example, if you're working with Windows 10, version 1809, use the ADK for Windows 10, version 1809.

On your technician PC:

1. If you have a previous version of the Windows Assessment and Deployment Kit (ADK), uninstall it.
2. Download the version of the [Windows ADK](#) that matches the version of Windows that you're installing.
3. Run the ADK installer to install the ADK with the following options. If you're using the ADK for Windows 10, version 1809, WinPE isn't part of the ADK installer and is a separate add-on package that you have to install after you install the ADK:
  - **Deployment Tools**
  - **User State Migration Tool (USMT)**
  - **Windows Preinstallation Environment (Windows PE)**
4. When installation finishes, close the installer window.

# Create a bootable Windows PE (WinPE) partition

WinPE is a small, command-line based operating system that you can use to capture, update, and optimize Windows images. In this section, we'll show you how to prepare a basic WinPE image on a bootable USB flash drive and try it out.

## Prepare WinPE files

1. On your technician PC, start the **Deployment and Imaging Tools Environment** as an administrator:

- Click **Start**, type **Deployment and Imaging Tools Environment**. Right-click **Deployment and Imaging Tools Environment** and select **Run as administrator**.

2. Use `copype` to create a working directory that has the base WinPE files:

```
copype amd64 C:\winpe_amd64
```

### TIP

If this doesn't work, make sure you're in the Deployment and Imaging Tools Environment, and not the standard command prompt.

## Customize WinPE

You can customize a WinPE image (boot.wim) in several ways by adding files and components to a mounted WinPE image.

Here are some examples of how you can modify your WinPE image:

- **Add an optional component.** [WinPE Optional components](#) ship in the ADK. These are packages that you can add to your WinPE image to add functionality to WinPE.
- **Add a graphics or network driver.** (WinPE includes generic video and network drivers, but in some cases, additional drivers are needed to show the screen or connect to the network.). To learn more, see [WinPE: Add drivers](#).
- **Set the power scheme to high-performance.** Speeds deployment. Note, our sample deployment scripts already set this scheme automatically. See [WinPE: Mount and Customize: High Performance](#).
- **Optimize WinPE:** Recommended for devices with limited RAM and storage (for example, 1GB RAM/16GB storage). After you add drivers or other customizations to Windows PE, see [WinPE: Optimize and shrink the image](#) to help reduce the boot time.

When you add packages to WinPE, performance will be reduced and boot time will increase. Only add packages that you need to complete a successful deployment.

## Mount your WinPE image

To customize a Windows image, you have to mount it before you can work with it. This is true for WinPE images as well. Mounting an image extracts the contents of an image file to a location where it can be viewed and modified. Throughout this lab we'll use DISM to mount and modify images. DISM comes with Windows, but we'll be using the version that is installed by the ADK, which we'll access through the Deployment and imaging tools environment.

Boot.wim is the WinPE image file. You can find it in the files that you copies with copype.cmd.

Mount the image:

- From the **Deployment and imaging tools environment**, mount the image:

```
Dism /mount-image /imagefile:c:\WinPE_amd64\media\sources\boot.wim /index:1 /mountdir:c:\winpe_amd64\mount
```

#### Add packages, optional components, dependencies, and language packs to WinPE (optional)

Use `Dism /Add-Package` to add packages to your mounted WinPE image. The ADK has [WinPE Optional Components](#) you can add for additional WinPE functionality. Some packages have dependencies and require other packages to be installed. For these packages, you'll have to install the dependencies before you add the package. For example, if you want to use Powershell in WinPE, you have to install the NetFx as well as the language-specific OCs. You can find OC CABs in

```
C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\arch>\WinPE_OCs\
```

Here's how to add Powershell support for en-us:

```
Dism /Add-Package /Image:"C:\WinPE_amd64\mount" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-WMI.cab" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-WMI_en-us.cab" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-NetFX.cab" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-NetFX_en-us.cab" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-Scripting.cab" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-Scripting_en-us.cab" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-PowerShell.cab" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-PowerShell_en-us.cab" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-StorageWMI.cab" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-StorageWMI_en-us.cab" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-DismCmdlets.cab" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-DismCmdlets_en-us.cab"
```

#### NOTE

Only add additional packages when necessary. The more packages you add, the greater the impact to boot time and performance.

#### Add drivers to WinPE (If needed)

If you need to add drivers to WinPE, you'll use `Dism /Add-Driver`. You'll only need to do this if WinPE doesn't already include the drivers for your hardware.

Here's how to add drivers to WinPE:

#### NOTE

This method requires .inf-based drivers. Get .inf-based drivers from your hardware vendor.

```
dism /image:C:\winpe_amd64\mount /Add-Driver /driver:"C:\Out-of-Box Drivers\mydriver.inf"
```

Where `C:\Out-of-Box Drivers\mydriver.inf` is the path of the driver you're adding.

To install all of the drivers in a folder and all its subfolders use the `/recurse` option. For example:

```
Dism /Image:C:\Winpe_amd64\mount /Add-Driver /Driver:c:\drivers /recurse
```

Where `C:\drivers` is the drivers folder that you're adding.

## Set the power scheme to high-performance

Setting WinPE to use high-performance mode will speed deployment. The sample scripts set WinPE to high-performance mode when they run, but if you want to make sure that WinPE always runs in high-performance mode, you can modify `startnet.cmd` in the WinPE image.

1. Use notepad to open `C:\Winpe_amd64\mount\windows\system32\startnet.cmd`

```
notepad C:\Winpe_amd64\mount\windows\system32\startnet.cmd
```

2. Add the following line to `startnet.cmd`:

```
powercfg /s 8c5e7fda-e8bf-4a96-9a85-a6e23a8c635c
```

3. Save the file and close Notepad.

## Cleanup your WinPE image

Run `dism /cleanup-image` to reduce the disk and memory footprint of WinPE and increase compatibility with a wide range of devices:

```
DISM /image:c:\winpe_amd64\mount /Cleanup-image /StartComponentCleanup
```

See [WinPE: Optimize and shrink the image](#) for more details.

## Commit your changes and unmount your image

If you've added extra files in your WinPE image, you can delete them to reduce your image size and improve performance. When you're done working with your image, you can commit your changes and unmount your image. Then export your customized WinPE image into your WinPE folder:

```
dism /unmount-image /mountdir:c:\winpe_amd64\mount /commit  
dism /export-image /sourceimagefile:c:\winpe_amd64\media\sources\boot.wim /sourceindex:1  
/DestinationImageFile:c:\winpe_amd64\mount\boot2.wim  
Del c:\winpe_amd64\media\sources\boot.wim  
Copy c:\winpe_amd64\mount\boot2.wim c:\winpe_amd64\media\sources\boot.wim
```

## Create a bootable WinPE drive

Now that you've updated your WinPE image to include everything it needs, you can make a bootable WinPE drive. From the Deployment and Imaging Tools Environment:

1. Connect your USB key to your technician PC.
2. Copy WinPE to the *WinPE* partition:

```
MakeWinPEMedia /UFD C:\winpe_amd64 P:
```

Where P: is the letter of the *WinPE* drive.

When prompted, press **Y** to format the drive and install WinPE.

## Boot your reference PC to WinPE

1. Connect the USB drive to your reference device.
2. Turn off the reference device, and then boot to the USB drive. You usually do this by powering on the

device and quickly pressing a key (for example, the **Esc** key or the **Volume up** key).

#### NOTE

On some devices, you might need to go into the boot menus to choose the USB drive. If you're given a choice between booting in UEFI mode or BIOS mode, choose UEFI mode. To learn more, see [Boot to UEFI Mode or Legacy BIOS mode](#). If the device does not boot from the USB drive, see the troubleshooting tips in [WinPE: Create USB Bootable drive](#).

WinPE starts at a command line, and runs `wpeinit` to set up the system. This can take a few minutes.

Leave this PC booted to Windows PE for now. You can remove the bootable USB drive.

## Customize your Windows image

Now that you have your WinPE image customized for your deployment, we'll get into how to get your Windows image ready for deployment. The process is similar to how we changed our WinPE image, but Windows has many additional customization options.

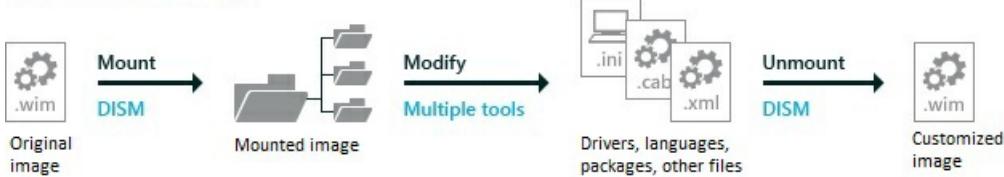
You can make either offline or online or online customizations to a Windows image. Offline customizations are done to the windows image (install.wim) from either the Technician PC or from the destination PC while booted into WinPE. In most scenarios, offline customizations are customizations you perform from the Technician PC. Online customizations are done on the Reference PC after it's been booted into audit mode.

The table below shows which customizations can be made online and offline. In a manufacturing environment, it's recommended to do as many customizations as possible offline.

SCENARIO	OFFLINE	ONLINE
Adding device drivers	X	X
Adding Microsoft Store apps	X	X
Adding Desktop (win32) apps	-	X
Adding language packs	X	X
Remove default language pack	X	-
Adding features-on-demand	X	X
Adding the latest cumulative update	X	X
Image optimization	X	X
Microsoft Store apps duplicate files cleanup	X	-
Microsoft Office	X	X

## Prepare and mount a Windows image

## Customize an image:



In this section we'll cover how to mount Windows images on your technician PC. Mounting a Windows image is the same process that we used to mount the WinPE image earlier. When we mount our Windows image (install.wim), we'll be able to access a second image, WinRE.wim, which is the image that supports recovery scenarios. Updating install.wim and WinRE.wim at the same time helps you keep the two images in sync, which ensures that recovery goes as expected.

Before we continue, make sure that you've created your *USB-B* drive. We showed you how to set it up in the [Get the tools you need](#) section.

### Backup your Windows image file

Before working on your Windows image, you want to make sure you have a backup copy in case something goes wrong. Make a copy of the original image:

First copy the install.wim from your Windows installation media to *USB-B*. Install.wim includes both Home and Professional images. We'll export the Home image from install.wim, and then work with that image during this lab.

1. Insert *USB-B* into your technician computer.
2. Mount the Windows 10 Home .img from the Win Home 10 32-BIT/X64 English OPK.
3. From the mounted image, copy D:\sources\install.wim to C:\temp\lab\images. (Where D: is the drive letter of the mounted image.)
4. From the Start menu, open Windows Kits, open the **Deployment and Imaging Tools Environment** as administrator.
5. Make a copy of your image in case something goes wrong.

```
copy "C:\temp\lab\Images\install.wim" C:\temp\lab\Images\install-backup.wim
```

6. Export the Home edition (index 2) from install.wim as basicimage.wim and delete the original C:\temp\lab\images\install.wim:

```
Dism /export-image /sourceimagefile:C:\temp\lab\images\install.wim /sourceindex:2  
/destinationimagefile:C:\temp\lab\images\basicimage.wim  
Del C:\temp\lab\images\install.wim
```

Now that you have your image exported, you can mount it.

### Mount your Windows image

Create a mount directory and mount basicimage.wim:

```
Md C:\mount\windows  
Dism /Mount-Wim /WimFile:C:\temp\lab\images\basicimage.wim /index:1 /MountDir:C:\mount\windows
```

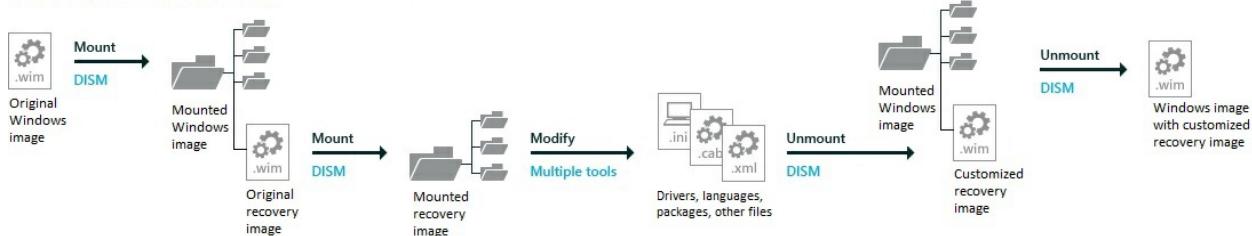
(where E:\ is the drive letter of *USB-B*)

### Mount your WinRE image

If a system can't successfully boot into Windows, it will fail over to the Windows Recovery Environment (WinRE). WinRE can repair common causes of unbootable operating systems. WinRE is based on WinPE, and to make it work for your customers, you can add drivers, languages, Windows PE Optional Components, and other troubleshooting and diagnostic tools.

The WinRE image is included inside the Windows 10 image, and is eventually copied to the Windows RE tools partition on the destination PC or device. To modify the WinRE image, you'll mount the Windows image, then mount the WinRE image inside it. Make your changes, unmount the WinRE image, then unmount the Windows image.

#### Customize the recovery image



You should update your recovery image to ensure a consistent recovery experience whenever you:

- Add boot-critical .inf-style drivers, such as the graphics and storage drivers.
- Add major updates to Windows, like LCUs.
- Add new languages, though this isn't always possible, as not all languages have Windows RE equivalents.)

#### NOTE

This lab assumes you'd rather keep winre.wim inside of install.wim to keep your languages and drivers in sync. If you'd like to save a bit of time on the factory floor, and if you're okay managing these images separately, you may prefer to remove winre.wim from the image and apply it separately.

- Mount the Windows RE Image file from your mounted image.

```
Md c:\mount\winre  
Dism /Mount-Wim /WimFile:C:\mount\windows\Windows\System32\Recovery\winre.wim /index:1  
/MountDir:C:\mount\winre
```

#### TIP

If winre.wim cannot be seen under the specified directory, use the following command to set the file visible:

```
attrib -h -a -s C:\mount\windows\Windows\System32\Recovery\winre.wim
```

**Troubleshoot:** If the mounting operation fails, make sure you're using DISM from the Deployment and Imaging Tools Environment. Do not mount images to protected folders, such as the User\Documents folder. If DISM processes are interrupted, consider temporarily disconnecting from the network and disabling virus protection.

#### Offline customizations

With your images mounted, you can start customizing. We'll show you how to make offline customizations to your Windows image. Offline customizations are changes that you can make to a mounted image without having to boot into the Windows installation. First, we'll walk through adding (and removing) languages from your Windows image.

## About Languages

In this section, we'll show you how to add languages to a Windows installation. If you want to add a language, you'll need a language pack from the Language Pack ISO, as well as an internet connection or access to the Feature on Demand ISO.

## Notes

- **Add languages before major updates.** Major updates include hotfixes, general distribution releases, or service packs. If you add a language later, you'll need to reinstall the updates.
- **Add major updates before apps.** These apps include universal Windows apps and desktop applications. If you add an update later, you'll need to reinstall the apps. We'll show you how to add these later in Lab 6: Add universal Windows apps
- **Add your languages to your recovery image, too:** Many common languages can be added to your recovery image. We'll show you how to add these later in Lab 12: Update the recovery image.

Always use language packs and Features-On-Demand (FOD) packages that match the language and platform of the Windows image.

Features on demand (FODs) are Windows feature packages that can be added at any time. When a user needs a new feature, they can request the feature package from Windows Update. OEMs can preinstall these features to enable them on their devices out of the box.

Common features include language resources like handwriting recognition. Some of these features are required to enable full Cortana functionality.

The following table shows the types of language packages and components available for Windows 10:

COMPONENT	SAMPLE FILE NAME	DEPENDENCIES	DESCRIPTION
Language pack	Microsoft-Windows-Client-Language-Pack_x64_de-de.cab	None	UI text, including basic Cortana capabilities.
Language interface pack	LanguageExperiencePack.am-et.neutral.appx	Requires a specific fully-localized or partially-localized language pack. Example: ca-ES requires es-ES.	UI text, including basic Cortana capabilities. To learn more, see Available Language Packs for Windows.
Basic	Microsoft-Windows-LanguageFeatures-Basic-de-Package	None	Spell checking, text prediction, word breaking, and hyphenation if available for the language. You must add this component before adding any of the following components.
Fonts	Microsoft-Windows-LanguageFeatures-Fonts-Thai-Package	None	Fonts required for some regions. Example, th-TH requires the Thai font pack.
Optical character recognition	Microsoft-Windows-LanguageFeatures-OCR-de-Package	Basic	Recognizes and outputs text in an image.

COMPONENT	SAMPLE FILE NAME	DEPENDENCIES	DESCRIPTION
Handwriting recognition	Microsoft-Windows-LanguageFeatures-Handwriting-de-de-Package	Basic	Enables handwriting recognition for devices with pen input.
Text-to-speech	Microsoft-Windows-LanguageFeatures-TextToSpeech-de-de-Package	Basic	Enables text to speech, used by Cortana and Narrator.
Speech recognition	Microsoft-Windows-LanguageFeatures-Speech-de-de-Package	Basic, Text-To-Speech recognition	Recognizes voice input, used by Cortana and Windows Speech
Retail Demo experience	Microsoft-Windows-RetailDemo-OfflineContent-Content-de-de-Package	Basic, plus the language-neutral Retail Demo package: Microsoft-Windows-RetailDemo-OfflineContent-Content-Package	<a href="#">Retail Demo experience.</a>

#### Add or change languages

In this section, we'll add languages and Features On Demand to your Windows image. We'll add the German (de-de) language pack, then we'll add the Japanese (ja-jp) language. Japanese is an example of a language that requires additional font support.

##### IMPORTANT

If you install an update that contains language-dependent resources prior to installing a language pack, the language-specific changes in the update won't be applied when you add the language pack. You need to reinstall the update to apply language-specific changes. To avoid reinstalling updates, install language packs before installing updates.

Language updates have a specific order they need to be installed in. For example, to enable Cortana, install, in order: **Microsoft-Windows-Client-Language-Pack**, then **-Basic**, then **-Fonts**, then **-TextToSpeech**, and then **-Speech**. If you're not sure of the dependencies, it's OK to put them all in the same folder, and then add them all using `DISM /Add-Package`.

Make sure that you are using language packs and features on demand that match the architecture of the image you are working with. Below are examples for building 64-bit systems.

##### Copy language files

1. Mount the Language pack ISO, and copy the language pack .cab files for the languages you're adding to `C:\temp\lab\LanguagePacks`. The examples below will use the German and Japanese languages.
2. Mount the Feature on Demand ISO, and copy the LanguageFeatures .cab files for the languages you're adding to `C:\temp\lab\LanguagePacks`. The examples below will use the German and Japanese languages.
3. Add German language pack and Feature on Demand language packages.

Use the language packs and Features on Demand from the 64-bit ISOs:

```
Dism /Add-Package /Image:C:\mount\windows /PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-Client-Language-Pack_x64_de-de.cab /PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-LanguageFeatures-Basic-de-de-Package~31bf3856ad364e35~amd64~~.cab  
/PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-LanguageFeatures-OCR-de-de-Package~31bf3856ad364e35~amd64~~.cab /PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-LanguageFeatures-Handwriting-de-de-Package~31bf3856ad364e35~amd64~~.cab  
/PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-LanguageFeatures-TextToSpeech-de-de-Package~31bf3856ad364e35~amd64~~.cab /PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-LanguageFeatures-Speech-de-de-Package~31bf3856ad364e35~amd64~~.cab  
/packagepath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-RetailDemo-OfflineContent-Content-de-de-Package~31bf3856ad364e35~amd64~~.cab
```

#### 4. (Optional) Add Japanese language packs and features on demand.

In Windows 10, some language-specific fonts were separated out into different language .cab files. In this section, we'll add the ja-JP language along with support for Japanese fonts.

Use the language packs and Features on Demand from the 64-bit ISOs:

```
Dism /Add-Package /Image:C:\mount\windows /PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-Client-Language-Pack_x64_ja-jp.cab /PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-LanguageFeatures-Basic-ja-jp-Package~31bf3856ad364e35~amd64~~.cab  
/PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-LanguageFeatures-OCR-ja-jp-Package~31bf3856ad364e35~amd64~~.cab /PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-LanguageFeatures-Handwriting-ja-jp-Package~31bf3856ad364e35~amd64~~.cab  
/PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-LanguageFeatures-TextToSpeech-ja-jp-Package~31bf3856ad364e35~amd64~~.cab /PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-LanguageFeatures-Speech-ja-jp-Package~31bf3856ad364e35~amd64~~.cab  
/PackagePath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-LanguageFeatures-Fonts-Jpan-Package~31bf3856ad364e35~amd64~~.cab /packagepath:C:\Temp\Lab\LanguagePacks\Microsoft-Windows-RetailDemo-OfflineContent-Content-ja-jp-Package~31bf3856ad364e35~amd64~~.cab
```

#### 5. Verify that the language packs are now part of the mounted images:

```
dism /get-packages /image:"C:\mount\windows"
```

Make sure that the added languages are on the list.

```
Package Identity : Microsoft-Windows-Client-LanguagePack ... de-DE~10.0.17134.1  
State : Installed
```

#### 6. Verify that the Features on Demand are in your image:

```
dism /get-capabilities /image:"C:\mount\windows"
```

Make sure that the language FODs are in the list:

```
Capability Identity : Language.Basic~~~de-de~0.0.1.0  
State : Installed  
...  
Capability Identity : Language.Handwriting~~~de-de~0.0.1.0  
State : Installed
```

### Configure language settings

This section covers how to change the default language and timezone of your mounted Windows image.

#### 1. Use DISM to set the default language of the image. We'll set the default language to German, since we

added it into our image in the previous steps.:

```
Dism /Image:C:\mount\windows /Set-AllIntl:de-DE
```

## 2. Verify your changes

```
Dism /Image:C:\mount\windows /Get-Intl
```

### Set the default timezone

You can use DISM to set the default timezone for a PC. Here we'll set the time zone. See [Default time zones](#) for a list of available time zones.

```
Dism /Set-TimeZone:"W. Europe Standard Time" /Image:"C:\mount\windows"
```

### Remove the base language from the image

This section covers removing a language from the Windows image. This is an optional step.

Now that our image has been set to use German as the default language, we can remove the English language features from it and make it a non-English image. To remove en-US completely from the image, you'll have to remove several components.

#### WARNING

Don't remove the English base language if you're shipping a PC in English.

For removing the language components from a 64-bit image:

```
dism /image:"c:\mount\windows" /remove-package /packagename:Microsoft-Windows-Client-LanguagePack-Package~31bf3856ad364e35~amd64~en-US~10.0.17134.1 /packagename:Microsoft-Windows-LanguageFeatures-Basic-en-us-Package~31bf3856ad364e35~amd64~~10.0.17134.1 /packagename:Microsoft-Windows-LanguageFeatures-Handwriting-en-us-Package~31bf3856ad364e35~amd64~~10.0.17134.1 /packagename:Microsoft-Windows-LanguageFeatures-OCR-en-us-Package~31bf3856ad364e35~amd64~~10.0.17134.1 /packagename:Microsoft-Windows-LanguageFeatures-Speech-en-us-Package~31bf3856ad364e35~amd64~~10.0.17134.1 /packagename:Microsoft-Windows-LanguageFeatures-TextToSpeech-en-us-Package~31bf3856ad364e35~amd64~~10.0.17134.1 /packagename:Microsoft-Windows-RetailDemo-OfflineContent-Content-en-us-Package~31bf3856ad364e35~amd64~~10.0.17134.1
```

#### TIP

Troubleshooting: If an error occurs when running these commands, try the command again on the package that failed.

Example:

```
Error: 0x800f0825  
Package Microsoft-Windows-LanguageFeatures-Basic-en-us-Package may have failed due to pending updates to servicing components in the image.
```

If the command completes with errors, check the DISM log file at C:\windows\Logs\DISM\dism.log.

### Add languages to Windows RE

Here we'll show you how to add languages to WinRE. Adding languages to WinRE ensures that the language that a customer expects is available in recovery scenarios. Follow these steps if you added languages to your Windows image.

WinRE uses the same language packs as WinPE. You can find these language packs on the language pack ISO,

and you can find language-specific WinPE OCs in the ADK installation folder at

```
C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\<lang>
```

To complete this section, copy the language packs from the Language pack ISO and the WinPE OCs from

```
C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs
```

to `C:\Temp\Lab\LanguagePacks\RE\<language>`.

## 1. Add German language packages

Use the 64-bit versions of language packs and WinPE optional components:

```
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\de-de\lp.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\Temp\Lab\LanguagePacks\RE\de-deWinPE-Rejuv_de-de.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\de-de\WinPE-EnhancedStorage_de-de.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\de-de\WinPE-Scripting_de-de.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\de-de\WinPE-SecureStartup_de-de.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\de-de\WinPE-SRT_de-de.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\de-de\WinPE-WDS-Tools_de-de.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\de-de\WinPE-WMI_de-de.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\de-de\WinPE-StorageWMI_de-de.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\de-de\WinPE-HTA_de-de.cab"
```

## 2. (Optional) Add Japanese language packs and font support to WinRE. Note that for Japanese, we will add an additional cab that is for font support.

Use the language packs and WinPE optional components from the 64-bit ISOs:

```
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\ja-jp\lp.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\ja-jp\WinPE-Rejuv_ja-jp.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\ja-jp\WinPE-EnhancedStorage_ja-jp.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\ja-jp\WinPE-Scripting_ja-jp.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\ja-jp\WinPE-SecureStartup_ja-jp.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\ja-jp\WinPE-SRT_ja-jp.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\ja-jp\WinPE-WDS-Tools_ja-jp.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\ja-jp\WinPE-WMI_ja-jp.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\ja-jp\WinPE-StorageWMI_ja-jp.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\ja-jp\WinPE-HTA_ja-jp.cab"
Dism /image:C:\mount\winre /add-package /packagepath:"C:\temp\lab\LanguagePacks\RE\ja-jp\WinPE-FontSupport-JA-JP.cab"
```

## 3. Set the default language for WinRE to match the default Windows language.

```
Dism /Image:C:\mount\winre /Set-AllIntl:de-DE
```

#### Remove the base languages from WinRE (Optional)

Similar to removing the base language in install.wim, we can remove the base language from WinRE as well.

For removing language components from a 64-bit image:

```
Dism /image:"c:\mount\winre" /remove-package /packagename:Microsoft-Windows-WinPE-LanguagePack-
Package~31bf3856ad364e35~amd64~en-US~10.0.17134.1 /packagename:WinPE-EnhancedStorage-
Package~31bf3856ad364e35~amd64~en-US~10.0.17134.1 /packagename:WinPE-HTA-Package~31bf3856ad364e35~amd64~en-
US~10.0.17134.1 /packagename:WinPE-Rejuv-Package~31bf3856ad364e35~amd64~en-US~10.0.17134.1 /packagename:WinPE-
Scripting-Package~31bf3856ad364e35~amd64~en-US~10.0.17134.1 /packagename:WinPE-SecureStartup-
Package~31bf3856ad364e35~amd64~en-US~10.0.17134.1 /packagename:WinPE-SRT-Package~31bf3856ad364e35~amd64~en-
US~10.0.17134.1 /packagename:WinPE-StorageWMI-Package~31bf3856ad364e35~amd64~en-US~10.0.17134.1
/packagename:WinPE-WDS-Tools-Package~31bf3856ad364e35~amd64~en-US~10.0.17134.1 /packagename:WinPE-WMI-
Package~31bf3856ad364e35~amd64~en-US~10.0.17134.1
```

## Drivers

You can add drivers to ensure Windows can successfully boot for the first time. Make sure that you add your driver to the right image:

- **DCHU drivers:** Many drivers include an information file (with an .inf extension) to help install the driver. These can be installed using tools described in this section.
- **Boot-critical drivers:** Graphics and storage drivers may sometimes need to be added to the Windows image (as shown in this lab), Windows PE image, and in the WindowsRE image.

The following shows how to add drivers in various ways. If your hardware doesn't require any additional drivers, you don't have to add any.

#### TIP

If you're creating several devices with identical hardware configurations, you can speed up installation time and first boot-up time by maintaining driver configurations when capturing a Windows image.

#### Add drivers to your Windows image

1. Add a single driver that includes an .inf file. In this example, we're using a driver named media1.inf:

```
Dism /Add-Driver /Image:"C:\mount\windows" /Driver:"C:\Drivers\PnP.Media.V1\media1.inf"
```

Where "C:\Drivers\PnP.Media.V1\media1.inf" is the base .inf file in your driver package.

2. If you want to add an entire folder of drivers, you can use the /Recurse option. This adds all .inf drivers in the folder and all its subfolders.

#### WARNING

While /Recurse can be handy, it's easy to bloat your image with it. Some driver packages include multiple .inf driver packages, which often share payload files from the same folder. During installation, each .inf driver package is expanded into a separate folder, each with a copy of the payload files. We've seen cases where a popular driver in a 900MB folder added 10GB to images when added with the /Recurse option.

```
Dism /Add-Driver /Image:"C:\mount\windows" /Driver:c:\drivers /Recurse
```

3. Verify that the drivers are part of the image:

```
Dism /Get-Drivers /Image:"C:\mount\windows"
```

Check the list of packages and verify that the list contains the drivers you added.

#### Add drivers to your WinRE image

If you added drivers to your Windows image, you should also add them to your WinRE image. Adding drivers to your recovery image ensures they are available during recovery scenarios. Adding drivers to a WinRE image is the same process as adding drivers to a regular Windows image.

```
Dism /Add-Driver /Image:"C:\mount\winre" /Driver:"C:\Drivers\PnP.Media.V1\media1.inf"
/LogPath=C:\mount\dism.log
```

#### NOTE

You can use the /recurse option to add an entire folder of drivers

## Updates

While your image is mounted, you can add Windows updates. The process is similar to the one we used to add drivers earlier.

Reminder:

- **Add languages before major updates.** Major updates include hotfixes, general distribution releases, or service packs. If you add a language later, you'll need to re-add the updates.
- **Add major updates before apps.** These apps include universal Windows apps and desktop applications. If you add an update later, you'll need to re-add the apps.
- **For major updates, update the recovery image too:** These may include hotfixes, general distribution releases, service packs, or other pre-release updates. We'll show you how to update these later in Lab 12: Update the recovery image.
- If a **Servicing Stack Update (SSU) is required**, you'll have to apply it before applying the most recent General Distribution Release or any future GDRs.

#### Add Windows updates to your image

Use DISM to apply the latest servicing stack update (SSU) and general distribution release (GDR) as well as any prerequisite KB updates. You can find KB updates in the following locations:

GDR: <https://aka.ms/win10releaseinfo>

SSU: <https://msdn.microsoft.com/en-us/windows/hardware/commercialize/manufacture/whats-new-in-windows-manufacturing>

KB Files: <https://catalog.update.microsoft.com>

#### IMPORTANT

If you install an update (hotfix, general distribution release [GDR], or service pack [SP]) that contains language-dependent resources prior to installing a language pack, the language-specific changes in the update won't be applied when you add the language pack. You need to reinstall the update to apply language-specific changes. To avoid reinstalling updates, install language packs before installing updates.

1. Get a Windows update package. For example, grab the [latest cumulative update listed in Windows 10](#)

update history from the Microsoft Update catalog. Extract the .msu file update to a folder, for example, E:\updates\windows10.0-kb4016240-x64\_0e60aebeb151d4b3598e4cfa9b4ccb1fc80e6e4d.msu. Make sure that your update matches the architecture of the image you are working with.

To learn more, see <https://myoem.microsoft.com/oem/myoem/en/product/winemb/pages/comm-ms-updt-ctlg-trnsta.aspx>.

2. Add the msu to your mounted image using `dism /add-package`.

```
Dism /Add-Package /Image:C:\mount\windows /PackagePath:"E:\updates\windows10.0-kb4000001-x64.msu"
```

You can also add multiple updates in the same command:

```
Dism /Add-Package /Image:C:\mount\windows /PackagePath:"E:\updates\windows10.0-kb4000001-x64.msu" /PackagePath:"E:\updates\windows10.0-kb0000002-x64.msu"
```

#### NOTE

Each package is typically a new KB that increases the build revision number of Windows. You can find the revision number of windows in the following registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\UBR
```

3. Lock in the update to ensure it get restored during recovery.

```
DISM /Cleanup-Image /Image=C:\mount\windows /StartComponentCleanup /ScratchDir:C:\Temp
```

4. Verify that the updates are in the image.

```
Dism /Image:C:\mount\windows /Get-Packages
```

Review the resulting list of packages and verify that the list contains the package. For example:

```
Package Identity : Package_for_RollupFix~31bf3856ad364e35~amd64~~15063.250.1.1
State : Installed
Release Type : Security Update
Install Time : 04/29/2017 6:26 PM
The operation completed successfully.
```

#### Add Update packages to WinRE

In this section, we cover how to add updates to the WinRE image.

#### IMPORTANT

You have to apply cumulative updates to your WinRE image in addition to your Windows image. Because updates are cumulative, when a new update is installed, old updates can be removed. The WinRE optimization that we cover later in the lab will remove unnecessary updates which will keep the WinRE image from growing in size.

To apply the update that you downloaded in the previous section to your WinRE image, you have to run `dism /add-package` to apply the update to the mounted WinRE image.

```
Dism /Add-Package /Image:C:\mount\winre /PackagePath:"E:\updates\windows10.0-kb4000001-x64.msu"
```

## Features and Apps

### Features on Demand

Add a Feature on Demand (FOD) to your Windows image. [Features on Demand](#) are features that you can choose to preinstall. You can see a list of available FODs, and recommendations for preinstallation [here](#).

Here we'll show you how to preinstall the .Net Framework Feature on Demand.

Note: While it's possible to add FODs using the /add-package command, we recommend using DISM with the /Add-Capability option.

1. Mount the Features on Demand ISO
2. On your technician PC, use DISM to get a list of available FODs in an image:

```
dism /image:C:\mount\windows /get-capabilities
```

This will show a list of available capabilities.

3. Add the .NET framework.

```
dism /image:C:\mount\windows /add-capability /capabilityname:NetFX3~~~~ /Source:E:
```

Where E: is the mounted FOD ISO.

.NET framework is now added to your image.

## Apps

This section covers working with Apps, including reinstalling inbox apps after updates, how to add Microsoft Store apps, and how to add Microsoft Office.

This section continues to use your mounted Windows image. If your image isn't still mounted, mount it.

**Reminder:** Install apps only after you have installed languages and updates, in that order.

### Reinstall inbox apps

Now that you've added languages and updates to your image, you have to reinstall the apps that come with Windows. This makes sure that the apps will work and include the languages you have added to your image. To reinstall these apps, you'll need the App update OPK or the inbox Apps ISO.

1. Extract the inbox apps ISO to c:\temp\lab\apps\inbox\amd64
2. Run the `E:\apps\ReinstallInboxApps-x64.bat` script.

Your apps are now ready to work with your image.

### Add a Microsoft Store app

To complete this section, you'll need to have the App update OPK or the inbox apps ISO. Whichever you are using, we'll refer to it as the App update OPK in the following steps.

1. Use DISM to add the HEVC codec .appx from the files you extracted in Step 1:
2. Install the HEVC .appx:

```
DISM /Add-ProvisionedAppxPackage /Image:c:\mount\windows  
/PackagePath:"C:\temp\lab\apps\amd64\Microsoft.HEVCVideoExtension_8wekyb3d8bbwe.x64.appx"  
/licensepath:"C:\temp\lab\apps\inbox\amd64\Microsoft.HEVCVideoExtension_8wekyb3d8bbwe.x64.xml"  
/DependencyPackagePath:"C:\temp\lab\apps\inbox\amd64\Microsoft.VCLibs.x64.14.00.appx"  
/DependencyPackagePath:"C:\temp\lab\apps\inbox\amd64\Microsoft.VCLibs.x86.14.00.appx"
```

#### NOTE

Include both the x86 and x64 versions of the dependency packages.

3. Use `DISM /Add-ProvisionedAppxPackage` to add any additional apps to your image.

4. Verify that the apps are installed:

```
Dism /Image:"C:\mount\windows" /Get-ProvisionedAppxPackages
```

#### Install a Microsoft Store app that won't be pinned to the Start Menu

For this section, you'll need to have at least two apps to add to your image. If you don't have any signed apps you can skip to the next section.

**New in Windows 10, version 1803:** A Microsoft Store app can now be installed without being pinned to the Start Menu. To install an app without also pinning it, use `DISM /Add-ProvisionedAppxPackage` with the `/region` switch when installing the app. When you create a custom Start Menu later in the lab, you'll be able to exclude an installed app from the Start Menu.

1. Gather your apps for installation

2. Install your apps, specifying a region with the `/region` option for each app. You can specify multiple regions by separating the regions with a `;`. We'll show you how you can use `LayoutModification.xml` with `/region` later in the lab:

```
Dism /Add-ProvisionedAppxPackage /PackagePath:app1.appxbundle /region="all"  
Dism /Add-ProvisionedAppxPackage /PackagePath:app2.appxbundle /region="US"
```

**Note:** If your apps have dependencies, include them in the command using the `/DependencyPackagePath`.

Like the examples in the previous section, it's common for apps to have dependencies on .NET.CoreRuntime and .VCLibs, but if your apps do not share those dependencies, then do not include them in your command.

#### Optimize your installed apps

**New in Windows 10, version 1803:** You can reduce app disk usage by running

`DISM /Optimize-ProvisionedAppxPackages` after you install apps. This command will only work when run against an offline image:

```
DISM.exe /Image:"C:\mount\windows" /Optimize-ProvisionedAppxPackages
```

#### Preinstall Microsoft Office

##### Related sets

The Office apps are delivered as a set of apps that are installed and serviced together. The main package for Office is a set of shared code and each Office app (for example, Word, Excel, and PowerPoint) is installed as an optional package. These packages are delivered as appxbundles that support all Store languages.

APP	PACKAGE ID	FILES
Shared code & required app (not visible to user)	Microsoft.Office.Desktop_8wekyb3d8bbwe	shared.appxbundle shared_license1.xml
Access	Microsoft.Office.Desktop.Access_8wekyb3d8bbwe	access.appxbundle access_license1.xml
Excel	Microsoft.Office.Desktop.Excel_8wekyb3d8bbwe	excel.appxbundle excel_license1.xml
Outlook	Microsoft.Office.Desktop.Outlook_8wekyb3d8bbwe	outlook.appxbundle outlook_license1.xml
PowerPoint	Microsoft.Office.Desktop.PowerPoint_8wekyb3d8bbwe	powerpoint.appxbundle powerpoint_license1.xml
Publisher	Microsoft.Office.Desktop.Publisher_8wekyb3d8bbwe	publisher.appxbundle publisher_license1.xml
Word	Microsoft.Office.Desktop.Word_8wekyb3d8bbwe	word.appxbundle word_license1.xml

#### Add Office apps to your image

To add the Office apps to an image, use DISM with the `/Add-ProvisionedAppxPackage` option. This option also requires the following information for each app you add:

- `/PackagePath` : This is only used to specify the path to the .appxbundle file for the shared code package.
- `/OptionalPackagePath` : This is used to specify the path to the .appxbundle file for an individual app, such as Word or Excel.
- `/LicensePath` : This is used to specify the path to the \_License1.xml file for an individual app. This is needed for both the shared package and each of the optional app packages.

1. Extract the Office OPK to C:\temp\lab\apps.
2. Use DISM to add all the Office apps to an offline image. The following example assumes the appxbundle and license xml files are in subdirectories on *USB-B* (D:). The example also excludes the /region switch because we want Office to appear in both the All Apps list, and as a Start Menu tile.

```
DISM /Image:C:\mount\windows /Add-ProvisionedAppxPackage /PackagePath="C:\temp\lab\apps\Office
Apps\shared.PreinstallKit\shared.appxbundle" /OptionalPackagePath="C:\temp\lab\apps\Office
Apps\excel.PreinstallKit\excel.appxbundle" /OptionalPackagePath="C:\temp\lab\apps\Office
Apps\powerpoint.PreinstallKit\powerpoint.appxbundle" /OptionalPackagePath="C:\temp\lab\apps\Office
Apps\word.PreinstallKit\word.appxbundle" /OptionalPackagePath="C:\temp\lab\apps\Office
Apps\outlook.PreinstallKit\outlook.appxbundle" /OptionalPackagePath="C:\temp\lab\apps\Office
Apps\publisher.PreinstallKit\publisher.appxbundle" /OptionalPackagePath="C:\temp\lab\apps\Office
Apps\access.PreinstallKit\access.appxbundle" /LicensePath="C:\temp\lab\apps\Office
Apps\shared.PreinstallKit\shared_license1.xml" /LicensePath="C:\temp\lab\apps\Office
Apps\excel.PreinstallKit\excel_license1.xml" /LicensePath="C:\temp\lab\apps\Office
Apps\powerpoint.PreinstallKit\powerpoint_license1.xml" /LicensePath="C:\temp\lab\apps\Office
Apps\word.PreinstallKit\word_license1.xml" /LicensePath="C:\temp\lab\apps\Office
Apps\outlook.PreinstallKit\outlook_license1.xml" /LicensePath="C:\temp\lab\apps\Office
Apps\publisher.PreinstallKit\publisher_license1.xml" /LicensePath="C:\temp\lab\apps\Office
Apps\access.PreinstallKit\access_license1.xml"
```

**TIP**

You need to specify both an appxbundle and a license package for the shared package, as well as for each individual app that you want to install.

**NOTE**

The command for adding the Office apps in audit mode (online instead of offline) would be the same, but replace /Image:C:\mount\windows with /online.

## Notes on Language Support

When install Office with DISM, Office language files are automatically added that match the languages that are in the Windows image.

- By default, Office will use the UI language of Windows. To configure Windows UI language, see DISM Languages and International Servicing Command-Line Options.
- There isn't a way to add additional Office languages that aren't part of the Windows image.
- Both display and proofing resources for Office will be installed for each Windows language.
- Some languages supported by Windows are not supported by Office – for these cases, Office will use the closest available language (for example, es-mx will attempt to use es-es instead).
- You can find installed Windows languages in Windows Settings > Time & language > Language.

3. Verify Office was installed:

```
Dism /Image:"C:\mount\windows" /Get-ProvisionedAppxPackages
```

where C is the drive letter of the drive that contains the image.

Review the resulting list of packages and verify that the list contains the Office apps, such as:

```
...
Displayname : Microsoft.Office.Desktop.Access
Version : 16000.8528.2136.0
Architecture : neutral
ResourceID : ~
PackageName : Microsoft.Office.Desktop.Access_16000.8528.2136.0_neutral_~_8wekyb3d8bbwe
Regions : None

Displayname : Microsoft.Office.Desktop.Excel
Version : 16000.8528.2136.0
Architecture : neutral
ResourceID : ~
PackageName : Microsoft.Office.Desktop.Excel_16000.8528.2136.0_neutral_~_8wekyb3d8bbwe
Regions : None

Displayname : Microsoft.Office.Desktop.Outlook
Version : 16000.8528.2136.0
Architecture : neutral
ResourceID : ~
PackageName : Microsoft.Office.Desktop.Outlook_16000.8528.2136.0_neutral_~_8wekyb3d8bbwe
Regions : None
...
```

To have the apps appear on the Start screen, follow the steps in the next section: Configuring Start tiles and taskbar pins.

To complete the Office install, you'll need to unmount the image and commit your changes, which we'll do this after we've completed all customizations at the end of this lab.

## Modify the Start layout

### New in Windows 10, version 1803

Apps can now be pinned to the All Apps list without also having to be pinned as a Start tile. This is done through the new "region" switch (as described in the previous section). In the previous section, we added three apps to our image: App1, App2, and Office. In this lab, we will omit App1 from LayoutModification.xml to ensure that App1 only appears in the All Apps list, and not also as a Start menu tile. We will also include App2 in LayoutModification.xml to demonstrate that even if the "region" switch was used when adding the app to the image, the app will still appear in the Start Menu if included in the LayoutModification.xml file. In other words, the LayoutModification.xml file takes precedence.

#### Start menu

If you don't create a LayoutModification.xml file and you use the Start Unattend settings, Windows will take the first 12 `SquareTiles` or `DesktoporSquareTiles` settings specified in the Unattend file. The system then places these tiles automatically within the newly-created groups at the end of Start. The first six tiles are placed in the first OEM group and the second set of six tiles are placed in the second OEM group. If `OEMName` is specified in the Unattend file, the value for this element is used to name the OEM groups that get created.

#### NOTE

The Start layout and taskbar pins can be lost if the user resets their PC with the built-in recovery tools. To make sure these settings stay on the device, update the recovery image along with the Windows image.

The Start tile layout in Windows 10 provides OEMs the ability to append tiles to the default Start layout to include Web links, secondary tiles, classic Windows applications, and universal Windows apps. OEMs can use this layout to make it applicable to multiple regions or markets without duplicating a lot of the work. In addition, OEMs can add up to three default apps to the frequently used apps section in the system area, which delivers system-driven lists, including important or frequently accessed system locations and recently installed apps.

To take advantage of the new features, and to have the most robust and complete Start customization experience for Windows 10, consider creating a LayoutModification.xml file. This file specifies how the OEM tiles should be laid out in Start. For more information about how to customize the new Start layout, see [Customize the Windows 10 Start Layout](#).

**Get started:** Use the sample layoutmodification.xml we've included in the `USB-B` files. Start with this file for this section of this lab. You can find it in `USB-B\StartLayout\layoutmodification.xml`.

To learn more about layoutmodification.xml, see [LayoutModification XML](#).

#### About the Start Menu layout

1. Use the optional `Region` attribute in the `RequiredStartGroups` element to use different layouts for different regions. The `Region` value must be equal to two-letter country/region codes, and separated by a pipe "|" delimiter if you're specifying multiple regions. The regions listed in the group are related to the regions you specified when you added apps to your image using the `/region` option. If the country/region setting for the Windows device matches a `RequiredStartGroups`, then the tiles laid out within the `RequiredStartGroups` is applied to Start. If you specify a region-agnostic `RequiredStartGroups` (or one without the optional `Region` attribute) then the region-agnostic `RequiredStartGroups` is applied to Start.

In your layoutmodification.xml, add regions to `RequiredStartGroups`:

```
<RequiredStartGroups Region="DE|ES|FR|GB|IT|US">
```

- Specify the tiles you want to add within an AppendGroup. OEMs can add a maximum of two AppendGroups. The following example shows two groups called "Fabrikam Group 1" and "Fabrikam Group 2", which contains tiles that will be applied if the device country/region matches what's specified in Region (in this case, the regions are Germany, Spain, France, United Kingdom, Italy, and United States). Each group contains three tiles and the various elements you need to use depending on the tile that you want to pin to Start.

```

<LayoutModificationTemplate
    xmlns="http://schemas.microsoft.com/Start/2014/LayoutModification"
    xmlns:defaultlayout="http://schemas.microsoft.com/Start/2014/FullDefaultLayout"
    xmlns:start="http://schemas.microsoft.com/Start/2014/StartLayout"
    Version="1">
    <RequiredStartGroupsCollection>
        <RequiredStartGroups
            Region="DE|ES|FR|GB|IT|US">
            <AppendGroup Name="Fabrikam Group 1">
                <start:DesktopApplicationTile
                    DesktopApplicationID="Microsoft.Windows.Explorer"
                    Size="2x2"
                    Row="0"
                    Column="4"/>
            </AppendGroup>
            <AppendGroup
                Name="Fabrikam Group 2">
                <start:Tile AppUserModelID="Microsoft.MicrosoftEdge_8wekyb3d8bbwe!MicrosoftEdge"
                    Size="2x2"
                    Row="0"
                    Column="0"/>
            </AppendGroup>
        </RequiredStartGroups>
    </RequiredStartGroupsCollection>

```

- In the Add Microsoft Store Apps section above, we asked you to install two apps using the /region switch: App1 and App2. Since we included a `/region` switch for both, both will be installed and appear in the All Apps list. However, to get just App2 to also appear as a Start menu tile, add the following line inside your second in the LayoutModification.xml file, such as:

Since we set the region to "US" for App2 when provisioning it, we should set the region to "US" here in our LayoutModification.xml to ensure App2 only appears as a Start menu tile in US images. So, ensure your region parameter looks like this:

```

<RequiredStartGroups Region="US">
    <AppendGroup Name="MyGroup">
        <start:Tile AppUserModelID="App2!App" Size="2x2" Row="2" Column="0"/>
    </AppendGroup>

```

Or at least includes the US, such as this:

```

<RequiredStartGroups Region="DE|US|JA">
    <AppendGroup Name="MyGroup">
        <start:Tile AppUserModelID="App2!App" Size="2x2" Row="2" Column="0"/>
    </AppendGroup>

```

- Add Microsoft Office Start Menu tiles. This is a requirement for OEMs that participate in the Jumpstart program.

You can have tiles appear on the Windows 10 Start menu for Word, PowerPoint and Excel. The tiles appear in a designated area for Microsoft apps (upper left in the following diagram). All additional apps will be accessible in the Apps list on the left.

Add the following to the LayoutModification.xml file to add the tiles:

```
<AppendOfficeSuite/>
<AppendOfficeSuiteChoice Choice="DesktopBridgeSubscription"/>
```

#### NOTE

To Add a desktop app, use the start:DesktopApplicationTile tag. If you know the application user model ID for the app, use that to identify it. Otherwise, if you pinned tiles that require .url or .lnk files, add the files to the following legacy Start Menu directories:

- %APPDATA%\Microsoft\Windows\Start Menu\Programs\
- %ALLUSERSPROFILE%\Microsoft\Windows\Start Menu\Programs\

Example:

```
Copy E:\StartLayout\Bing.url "C:\mount\Windows\ProgramData\Microsoft\Windows\Start Menu\Programs"
Copy E:\StartLayout\Paint.lnk "C:\mount\Windows\ProgramData\Microsoft\Windows\Start Menu\Programs"
Copy E:\StartLayout\Bing.url "C:\mount\Windows\Users\All Users\Microsoft\Windows\Start Menu\Programs"
Copy E:\StartLayout\Paint.lnk "C:\mount\Windows\Users\All Users\Microsoft\Windows\Start Menu\Programs"
```

6. Save the Start Layout file as layoutmodification.xml.

7. Copy the saved file to your mounted image, to the

C:\Mount\Windows\Users\Default\AppData\Local\Microsoft\Windows\Shell folder. If a layoutmodification.xml file already exists in the folder, replace the existing file with the new one.

#### Add a license agreement and info file

##### Add an OEM-specific license

In this section, we'll cover how an OEM can add their own license terms during OOBE.

#### NOTE

If the license terms are included, the OEM must include a version of the license terms in each language that is preinstalled onto the PC. A license term text must be an .rtf file, and have an .html file with a matching name in the same folder. See [OEM license terms](#) for more information on license files.

To begin adding license terms, you'll have to create folders for your license files, and then configure OOBE to show the license on first boot.

1. Create folders for your system languages under the following directory:

C:\mount\windows\Windows\System32\oobe\info\default\

2. Name each folder under the C:\mount\windows\Windows\System32\oobe\info\default\ directory as the Language Decimal Identifier that corresponds to the language. Do this step for each language pack that's in the Windows image.

**Note:** Please see [this link to see complete list of language decimal identifiers of corresponding languages](#).

For example, if en-us and de-de language packs are added to the Windows image, add a folder named "1033" (representing en-us language) in C:\mount\windows\Windows\System32\oobe\info\default. Then add a folder named "1031" (de-de language) under the same C:\mount\windows\Windows\System32\oobe\info\default\ directory.

```
MD c:\mount\windows\windows\system32\oobe\info\default\1031
MD c:\mount\windows\windows\system32\oobe\info\default\1033
```

3. Create a license terms .rtf file for each language you have in your image, and copy them to the language-specific oobe folder.

For example: Move the English agreement.rtf file to C:\mount\windows\Windows\System32\oobe\info\default\1033\ directory and move the German agreement.rtf to C:\mount\windows\Windows\System32\oobe\info\default\1031.

```
copy E:\resources\english-agreement.rtf
c:\mount\windows\windows\system32\oobe\info\default\1033\agreement.rtf
copy E:\resources\german-agreement.rtf
c:\mount\windows\windows\system32\oobe\info\default\1031\agreement.rtf
```

4. Open a text editor and create .html versions of your license terms. Save the terms to the same folders as the .rtf versions. You can use the [EULA example](#) from [OEM license terms](#) to create sample files. The names of the EULA files should be identical, except for the extension.

```
C:\mount\windows\windows\system32\oobe\info\default\1033\agreement.html (English version)
C:\mount\windows\windows\system32\oobe\info\default\1031\agreement.html (German version)
```

5. Create an oobe.xml file to specify the license agreement.rtf file path. Windows will automatically find the accompanying .html file. Below is a sample oobe.xml which is located at *USB-B\ConfigSet\oobe.xml*

```
<FirstExperience>
<oobe>
  <oem>
    <eulafilename>agreement.rtf</eulafilename>
  </oem>
</oobe>
</FirstExperience>
```

6. Copy oobe.xml file to `C:\mount\windows\windows\system32\oobe\info\`
7. Copy oobe.xml to the language-specific folders you created earlier. For example: Copy oobe.xml to C:\mount\windows\Windows\System32\oobe\info\default\1033, which has a file called agreement.rtf in English. To add the German agreement, copy oobe.xml to C:\mount\windows\Windows\System32\oobe\info\default\1031\ directory, which has the German agreement.rtf file.

```
copy E:\configset\oobe.xml c:\mount\windows\windows\system32\oobe\info\default\1033
copy E:\configset\oobe.xml c:\mount\windows\windows\system32\oobe\info\default\1031
```

8. Now each language folder has an oobe.xml, agreement.rtf, and agreement.thml file in that corresponding language.

When the image first boots into OOBE, it will display the license agreement.

### Create an image info file and add it to your image

- Create an csup.txt file to specify when the Windows image was created. This file must include the date that the image was created, in the form of 'MM-DD-YYYY', with no other characters, on a single line at the top of the file. This command will create the file:

```
echo 4-24-2018 >"C:\mount\windows\Windows\csup.txt"
```

## Customize Windows with an answer file

Answer files (or Unattend files) can be used to modify Windows settings in your images during Setup. You can also create settings that trigger scripts in your images that run after the first user creates his/her account and picks the default language. Answer files allow you specify various setup options, including how to partition disks, the location of the Windows image to install, and the product key to apply. Values that apply to the Windows installation, such as the names of user accounts, display settings, and Internet Explorer Favorites can also be specified. The answer file for Setup is typically called Unattend.xml.

Unattend files include several sections, each of which are processed at different times throughout the Windows installation process. These phases are called configuration passes. Here are the most often-used passes:

You can specify which configuration pass to add new settings to:

- **1 - windowsPE:** These settings are used by the Windows Setup installation program. If you're modifying existing images, you can usually ignore these settings.
- **2 - offlineServicing:** Settings in offlineServicing are processed when DISM is used to apply an unattend file to an offline image.
- **4 - specialize:** Most settings should be added here. These settings are triggered both at the beginning of audit mode and at the beginning of OOBE. If you need to make multiple updates or test settings, generalize the device again and add another batch of settings in the Specialize Configuration pass.
- **6 - auditUser:** Runs as soon as you start audit mode. AuditUser is a good place to run a system test script. We'll add Microsoft-Windows-Deployment\RunAsynchronousCommand as our example. To learn more, see Add a Custom Script to Windows Setup.
- **7 - oobeSystem:** Use sparingly. Most of these settings run after the user completes OOBE. The exception is the Microsoft-Windows-Deployment\Reseal\Mode = Audit setting, which we'll use to bypass OOBE and boot the PC into audit mode. If your script relies on knowing which language the user selects during OOBE, you'd add it to the oobeSystem pass.

While you can set many Windows settings in audit mode, some settings can only be set by using an answer file or Windows Configuration Designer, such as adding manufacturer's support information. A full list of answer file settings (also known as Unattend settings) is in the [Unattended Windows Setup reference](#).

### Make an answer file

Use Windows System Image Manager (SIM) to create and modify unattend files. SIM is installed as part of the ADK. We have included some answer files in *USB-B* that will get you started. Make sure that your answer settings include the required settings as outlined in the OEM Policy Document:

- For OA 3.0 systems:

```
md c:\mount\windows\windows\panther  
copy /y E:\AnswerFiles\OA3.0\Unattend.xml C:\Mount\Windows\Windows\Panther
```

(where E:\ is *USB-B*)

- For non-OA 3.0 systems:

```
md c:\mount\windows\Windows\panther  
copy /y E:\AnswerFiles\Non_OA3.0\Unattend.xml C:\Mount\Windows\Windows\Panther
```

(where E:\ is *USB-B*)

### Create a catalog file in Windows SIM

Catalog files (.clg) are files with information about settings that are applicable to a particular Windows image. When working with a Windows image in SIM, you have to first create a catalog file for the Windows WIM you're working with.

1. Start Windows System Image Manager (WSIM).
2. Click File > Select Windows Image.
3. In Select a Windows Image, browse to and select the image file (D:\install.wim).
4. Select the Home edition of Windows 10 and click OK.
5. Click Yes to create the catalog file. Windows SIM creates the file based on the image file, and saves it to the same folder as the image file. This process can take several minutes.

The catalog file appears in the Windows Image pane. Windows SIM lists the configurable components and packages in that image.

### Create an answer file

If you're not working with the existing unattend file, you can create a new one in Windows SIM:

- Click file > New Answer File

The new answer file will appear in the right-pane.

### Add answer file settings

In this section, we'll show you how to create an answer file that can configure settings when you deploy your Windows image.

Before you start, create a folder called Panther in your mounted Windows image. Windows will automatically look in this folder for an answer file.

```
md c:\mount\windows\Windows\panther
```

#### Add OEM info (optional)

1. In the Windows Image pane, expand Components, right-click amd64\_Microsoft-Windows-Shell-Setup\_(version), and then select Add Setting to Pass 4 specialize.
2. In the Answer File pane, select Components\4 specialize\amd64\_Microsoft-Windows-Shell-Setup\_neutral\OEMInformation.
3. In the OEMInformation Properties pane, in the Settings section, set:

```
Manufacturer=Fabrikam  
Model=Notebook Model 1  
Logo=C:\Fabrikam\Fabrikam.bmp
```

4. Save the answer file in the Panther folder as `USB-B\AnswerFiles\unattend.xml`.

Next you'll have to make sure that the logo you specified above is in your Windows image.

1. Create a 32-bit color image with a maximum size of 120x120 pixels. Save it as D:\AnswerFiles\Fabrikam.bmp file on your Technician PC. We've included a sample on USB-B, which you can use for this lab: D:\Logos\Fabrikam.bmp.
2. Create a folder in your mounted Windows image called `Fabrikam`.

```
mkdir c:\mount\windows\Fabrikam
```

3. Copy Fabrikam.bmp to the Fabrikam folder you just created:

```
copy fabrikam.bmp c:\mount\windows\fabrikam
```

Set the device to automatically boot to audit mode

Using the same unattend.xml file, set the PC to automatically [boot into Audit mode](#).

1. In the Windows Image pane, expand Components, right-click amd64\_Microsoft-Windows-Deployment\_(version), and then select Add Setting to Pass 7 oobeSystem.
2. In the Answer File pane, select Components\7\_oobeSystem\amd64\_Microsoft-Windows-Deployment\_neutral\Reseal.
3. In the Reseal Properties pane, in the Settings section, select [Mode=Audit](#).
4. Save the answer file in the Panther folder as unattend.xml.

Enable S-Mode

#### NOTE

If you don't want to enable S-Mode, you can [skip to the next section](#).

**New in Windows 10, version 1803:** Windows 10 S is no longer a separate SKU from other versions of Windows. S-mode is now a mode that can be activated on Home and Pro SKUs.

**Set S Mode** In this section, we'll show you how to enable S mode in a Windows image. We'll use an Unattend file that has a setting in **Pass 2 - offlineServicing**, and use DISM to apply it to our mounted Windows image.

1. Use Windows SIM to modify unattend.xml.
2. Add the amd64\_Microsoft\_Windows\_CodeIntegrity component to Pass 2 offline Servicing.
3. Set amd64\_Microsoft\_Windows\_CodeIntegrity\SkuPolicyRequired to [1](#).
4. Save the answer file in the Panther folder as unattend.xml.
5. Use DISM to apply the unattend file and enable S Mode:

```
dism /image:C:\mount\windows /apply-unattend:C:\mount\windows\windows\panther\unattend.xml
```

**Note:** Only Pass 2 - offline Servicing is processed when an unattend file is applied with DISM.

S mode is now applied to the Windows image. When the PC boots, the same Code Integrity policy that is enforced in Windows 10 S will be enforced on your Windows installation.

#### Enable Manufacturing mode

If you plan to make additional changes to your image in audit mode, such as editing the registry, running a script, or running a command from the command-prompt, you have to temporarily enable Manufacturing Mode, which allows unsigned code that is normally blocked in S Mode to run in Audit mode. This allows you to run scripts, installers and diagnostic tools (i.e., unsigned code) during the manufacturing process. Manufacturing Mode is enabled by adding a registry key to the offline image, and it's disabled by removing the key when booted into audit mode.

1. On your mounted image, load the SYSTEM registry hive from your mounted image into regedit on your Technician PC. We'll use a temporary hive called HKLM\Windows10S.

```
reg load HKLM\Windows10S C:\Mount\Windows\Windows\System32\Config\System
```

2. Add the manufacturing registry key.

```
reg add HKLM\Windows10S\ControlSet001\Control\CI\Policy /v ManufacturingMode /t REG_DWORD /d 1
```

### 3. Unload the registry hive from your Technician PC.

```
reg unload HKLM\Windows10S
```

After you unmount your image and commit your changes (below), your Windows 10 in S Mode image will have the manufacturing key that will allow you to run unsigned code in audit mode.

#### **IMPORTANT**

Make sure to remove the manufacturing registry key before shipping your Windows 10 in S mode device. We show you how to do that later in the lab, or you can learn how at [Manufacturing mode](#)

## **Set the Windows edition**

In this section, we will upgrade the Windows OS edition from Home to Pro.

**Notes:** - You cannot set a Windows image to a lower edition. - Note: You should not use this procedure on an image that has already been changed to a higher edition. - Since S-mode is not enabled until this image is booted on the Reference device, it doesn't matter if you run the /Set-Edition command before or after applying the unattend settings (which includes the S-mode `<SkuPolicyRequired>` element). - You wouldn't normally switch editions offline like we'll show you here. Instead of upgrading editions, you would've mounted a Windows Professional wim to begin with. This step is included here for instructional purposes; just to show you the use of the /Set-Edition command.

1. Determine available editions by running the following command to determine what images you can upgrade the image to:

```
Dism /Get-TargetEditions /Image:C:\mount\windows
```

Note the edition IDs available.

2. Upgrade the edition to the Professional edition.

```
Dism /Set-Edition:Professional /Image:C:\mount\windows
```

## **Keep Windows settings through a recovery**

In recovery scenarios, Windows doesn't automatically save settings created through with answer files, Windows Start Menu customizations created with LayoutModification.xml, or first-login info from oobe.xml.

To make sure Windows saves your customizations:

- Save copies of unattend.xml, LayoutModification.xml, plus your C:\mount\windows\Windows\System32\Oobe folder, in C:\Recovery\OEM.
- Add scripts ResetConfig.xml and EnableCustomizations.cmd to C:\Recovery\OEM. Get these from [Sample scripts: Keeping Windows settings through a recovery](#).

## **Optimize WinRE (part 1)**

1. Increase the scratchspace size of the WinRE image.

```
Dism /image:c:\mount\winre /set-scratchspace:512
```

## 2. Cleanup unused files and reduce the size of winre.wim

```
dism /image:"c:\mount\winre" /Cleanup-Image /StartComponentCleanup
```

### Unmount your images

1. Close all applications that might be accessing files from the image, including File Explorer.

2. Commit the changes and unmount the Windows RE image:

```
Dism /Unmount-Image /MountDir:"C:\mount\winre" /Commit
```

where C is the drive letter of the drive that contains the image.

This process can take a few minutes.

3. Make a backup copy of the updated Windows RE image and replace the old WinRE image with the newly exported one:

```
dism /export-image /sourceimagefile:c:\mount\windows\windows\system32\recovery\winre.wim /sourceindex:1  
/DestinationImageFile:c:\temp\lab\winre_bak.wim  
Del c:\mount\windows\windows\system32\recovery\winre.wim  
Copy c:\temp\lab\winre_bak.wim c:\mount\windows\windows\system32\recovery\winre.wim
```

If prompted, specify **F** for file

Troubleshoot: If you cannot see winre.wim under the specified directory, use the following command to set the file visible:

```
attrib -h -a -s C:\mount\windows\Windows\System32\Recovery\winre.wim
```

4. Check the new size of the Windows RE image:

```
Dir "C:\mount\windows\Windows\System32\Recovery\winre.wim"
```

5. Based on the size of the new winre.wim, adjust the partition sizes in the deployment scripts so they include enough room for winre.wim, as well as some additional free space.

Follow the below partition layout size chart to determine the size of your recovery partition in the createpartitions-<firmware>-<imageformat>.txt files. The amount of free space left is after you copy winre.wim to the hidden partition. See the below disk partition rules for more information.

PARTITION SIZE	FREE SPACE
Less than 500 MB	Minimum 50 MB free
450 MB - 680 MB	Minimum 320 MB free
More than 680 MB	1024 MB free

Example of the recovery partition size from the createpartitions diskpart scripts.:

```
rem == 3. Windows RE tools partition =====
create partition primary size=465
```

- Optimize the Windows image the same way you did with the WinRE image:

```
Dism /Image:c:\mount\windows /Cleanup-Image /StartComponentCleanup
```

- Commit the changes and unmount the Windows image:

```
Dism /Unmount-Image /MountDir:"C:\mount\windows" /Commit
```

Where C is the drive letter of the drive that contains the image. This process may take several minutes.

- Finish optimizing your image by exporting the image. During the export process, DISM removes files that were superseded and the image will have a reduced file size. Export the Windows image into a new image file:

```
Dism /Export-Image /SourceImageFile:"C:\temp\lab\Images\basicimage.wim" /SourceIndex:1
/DestinationImageFile:"C:\temp\lab\Images\install.wim"
```

You now have a customized Windows image that you can deploy to other PCs. In the next section, we'll show you how to deploy the image to your reference PC, make online changes, and then finalize the Windows image to get it ready for final deployment.

## Deploy your images to a new PC

In this section we'll deploy an image to a PC so that it can be customized in Audit mode. Before you start this section:

- Copy your customized images to *USB-B\Images*

```
copy c:\temp\lab\images\install.wim e:\images
copy c:\temp\lab\images\winre_bak.wim e:\images
```

- Make sure the deployment scripts in *USB-B\Deployment*. These should have been copied when you extracted the USB-B download to *USB-B*.

### Boot to WinPE

If you're not already booted into WinPE on the device you're deploying your image to, boot into WinPE:

- Connect the USB key with the *WinPE* partition boot the reference computer.
- If you're using two separate USB drives, after WinPE has been booted connect *USB-B*.
- At the X:\Windows\system32> command line, type `diskpart` and press Enter.
- At the \DISKPART> command line type `list volume`.
- In the "Label" column, note the letter of the volume under the "Ltr" column. This is the drive letter of your USB key. (example E)
- Type exit to quit Diskpart

### Use a deployment script to apply your image

Run a script to create partitions and apply your image. We'll use `applyimage.bat` in *USB-B\deployment* to do this for us.

ApplyImage.bat uses diskpart scripts to create the partitions and define the partition layout. This script and the subscripts it calls must be placed in the same subfolder. You can update these scripts to change the partition sizes as you desire.

#### NOTE

If you're going to be capturing and deploying your final image as an FFU, choose the options to not configure recovery. This allows you to expand the Windows partition, if needed, after you apply your FFU. You can configure recovery after you expand the Windows partition.

1. Run applyimage.bat and specify the image to apply:

```
D:  
cd Deployment  
ApplyImage.bat D:\Images\install.wim
```

Where D: is the drive letter of your script and image on your USB Storage drive.

When prompted by the script:

- a. Choose whether or not to configure the recovery partition
  - Y : Configures the Windows recovery partition
  - N : Does not configure the recovery partition. The recovery partition can be configured later.  
Choose this option if you're going to be capturing and deploying your image as an FFU.
- b. Press  Y to format the drive
- c. Choose  N to not to deploy as Compact OS.
- d. Press  N to indicate the image does not include extended attributes (EA)

#### NOTE

Only use compact OS on Flash drive based devices as compact OS performance is heavily dependent on the storage device capabilities. Compact OS is NOT recommended on rotational devices. Please reference Compact OS for more information.

2. Remove the USB drive and reboot the reference PC.

```
exit
```

The PC should boot into Windows Audit mode based on the settings of the unattend file we created earlier. If you're not using an unattend.xml file that sets the PC to boot into Audit Mode, you can press  Ctrl+Shift+F3 during OOB to boot into Audit mode.

## Make online customizations (audit mode)

You can use audit mode to customize Windows using the familiar Windows environment. In audit mode, you can add Windows desktop applications, change system settings, add data, and run scripts.

To make sure your audit mode changes are included in the recovery image, you'll need to capture these changes into a provisioning package using ScanState. This package gets used by the system recovery tools to restore your changes if things go wrong. You can optionally save drive space by running the applications directly from the compressed recovery files; this is known as single-instancing.

If you want to capture the changes in an image and apply it to other devices, you'll need to use Sysprep to generalize the image.

### Verify Customizations in Audit Mode

We don't recommend connecting your PC to the internet during manufacturing, and also don't recommend installing updates from Windows Update in audit mode because it will likely generate an error during sysprep.

1. After setup has finished, the computer logs into Windows in Audit mode automatically as an Administrator.
2. Verify the changes from the answer file (see manufacturer name, support phone number and other customizations) are present.

### Verify the edition

Earlier in the lab, we upgraded the edition from Windows Home to Windows Professional. Verify this change:

1. Open Command Prompt as administrator
2. Run:

```
dism /online /get-current-edition
```

3. Make sure it's the right edition. It should look like this:

```
Current edition is:  
Current Edition : Professional  
The operation completed successfully.
```

**Note:** If you are working with a device that has S Mode enabled, the edition will still just say "Professional." This is because S Mode is a mode, not an edition.

### Verify S Mode

If you enabled S Mode earlier in the lab, verify that S Mode is enabled.

1. From Start, open Settings.
2. In Settings, open Update & Security
3. Choose Activation from the left pane

If your device has S Mode enabled, it will show here.

### Apps and Store opportunities

Through Windows 10 and the Microsoft Store, you have tremendous opportunities for brand and device differentiation, revenue creation, and customer access.

Microsoft Store apps are at the center of the Windows 10 experience. They are Windows universal apps, so you can build apps for desktops, tablets, or phones that run Windows 10. As an OEM, you can provide an engaging customer experience and increase brand loyalty by providing a great set of value-added software and services along with the high-quality hardware that you build.

**Important:** The key below must be set in Audit mode.

You have to change a registry setting to add your OEM ID. If you're an OEM Microsoft Store Program participant, contact [PartnerOps@microsoft.com](mailto:PartnerOps@microsoft.com) to get your OEM ID.

ITEM	LOCATION IN REGISTRY
OEMID	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Store, (REG_SZ) OEMID
SCM ID	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Store, (REG_SZ) StoreContentModifier

## OEMID

1. Run regedit.exe from Command Prompt
2. Navigate to HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Store
3. Right click under (Default) -> click new
4. Click String Value
5. Type OEMID
6. Double click OEMID and enter OEM name in Value data: text field

## SCMID

1. Run regedit.exe from Command Prompt
2. Navigate to HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Store
3. Right click under (Default) -> click new
4. Click String Value
5. Type StoreContentModifier
6. Double click StoreContentModifier and enter OEM name in Value data: text field

### IMPORTANT

The OEMID registry key is not restored automatically during PBR in Windows 10. Please refer to the scanstate section of this guide on how to restore the OEMID registry key during PBR operation.

## Prepare your image for Push Button Reset

This section provides guidance for setting up the recovery environment for Push Button Reset (PBR) scenarios.

Please reference Push-button reset and Windows Recovery Environment (Windows RE) and Hard Drives and Partitions for more information.

Push-button reset, is a built-in recovery tool which allows users to recover the OS while preserving their data and important customizations, without having to back-up their data in advance. It reduces the need for custom recovery applications by providing users with more recovery options and the ability to fix their own PCs with confidence.

|

### Prepare ScanState

To start working with Push Button Reset, you'll need to copy ScanState to *USB-B*.

Use scanstate to capture Classic Windows applications and settings on your image.

**Note:** You'll use your technician PC to prepare ScanState.

1. On Technician PC Insert *USB-B*
2. Open Deployment and Imaging tools command prompt as administrator

3. Run copydandi.cmd to copy necessary files to *USB-B\scanstate*:

```
Copydandi.cmd amd64 e:\scanstate
```

Where E: is the letter of USB-B drive.

### Create a Scanstate migration file

In this section, you'll create a configuration file that will restore files and registry keys during Push-button reset.

Create a migration XML file used to restore registry values manually entered during manufacturing process. The sample below restores the OEMID registry value set earlier in this document.

**Note:** *USB-B\recovery\recoveryimage\regrecover.xml* already contains the registry values. You can use this file instead of creating a new file.

1. Open notepad
2. Copy and paste the following xml into Notepad. This tells ScanState to migrate the OEMID registry key:

```
<migration urlid="https://www.microsoft.com/migration/1.0/migxmltest/test">
    <component type="System" context="UserAndSystem">
        <displayName>OEMID</displayName>
        <role role="Settings">
            <rules>
                <include>
                    <objectSet>
                        <pattern type="Registry">HKLM\Software\Microsoft\Windows\CurrentVersion\Store
[OEMID]</pattern>
                    </objectSet>
                </include>
            </rules>
        </role>
    </component>
</migration>
```

3. If you enabled S mode, make sure that the manufacturing registry key doesn't get migrated by telling ScanState to exclude it. Add the following rule to exclude the registry key:

```
<unconditionalExclude>
    <objectSet>
        <pattern type="Registry">HKLM\SYSTEM\CurrentControlSet\Control\CI\Policy [ManufacturingMode]
    </pattern>
    </objectSet>
</unconditionalExclude>
```

4. Save the file as regerecover.xml.

### Create recovery package using Scanstate

#### On your reference PC:

Use ScanState to capture installed customizations into a provisioning package, and then save it to *c:\Recovery\customizations*. We'll use samples from *USB-B\Recovery\RecoveryImage* to create the provisioning package.

**Important:** For Push Button Reset to work properly, packages have to be .ppkg files that are stored in *C:\Recovery\Customizations*.

1. Create the recovery OEM folder and copy contents of *USB-B\Recovery\RecoveryImage*

**Important:** To retain the customized start layout menu during recovery the layoutmodification.xml needs to be copied again during recovery process. We'll copy it here and then use EnableCustomizations.cmd to copy it during recovery.

```
Copy E:\Recovery\RecoveryImage c:\recovery\OEM  
Copy E:\StartLayout\layoutmodification.xml c:\recovery\OEM
```

## 2. Run ScanState to gather app and customizations

For x64 Windows 10 PCs:

```
mkdir c:\recovery\customizations  
E:\ScanState\scanstate.exe /apps /ppkg C:\Recovery\Customizations\apps.ppkg  
/i:c:\recovery\oem\regrecover.xml /config:E:\scanstate\Config_AppsAndSettings.xml /o /c /v:13  
/l:C:\ScanState.log
```

Where E: is the drive letter of *USB-B*

## 3. When ScanState completes successfully, delete scanstate.log and miglog.xml files:

```
del c:\scanstate.log  
del c:\miglog.xml
```

### Create Extensibility scripts to restore additional settings

You can customize the Push-button reset experience by configuring extensibility points. This enables you to run custom scripts, install additional applications, or preserve additional user, application, or registry data.

During recovery, PBR calls EnableCustomizations.cmd which we'll configure to do 2 things:

1. Copy the unattend.xml file used for initial deployment to the \windows\panther.
2. Copy the layoutmodification.xml to the system.

This will restore the additional layout settings from these 2 answer files during PBR.

```
!Important Recovery scripts and unattend.xml must be copied to c:\Recovery\OEM for PBR to pickup and  
restore settings defined in the unattend.xml.
```

### Copy unattend.xml files for restoring settings

For OA 3.0 systems:

```
Copy e:\AnswerFiles\oa3.0\unattendsysprep.xml c:\Recovery\OEM\unattend.xml
```

For non-OA 3.0 systems:

```
Copy e:\AnswerFiles\non_oa3.0\unattendsysprep.xml c:\Recovery\OEM\unattend.xml
```

### Copy winre.wim backup

During the deployment winre.wim file is moved. Before capturing a final image, the backup winre.wim we created must be copied back, or the recovery environment will not work in the final image deployment.

```
Copy e:\images\winre_bak.wim c:\windows\system32\recovery\winre.wim
```

## **Reseal the image**

In this section, we'll use sysprep.exe to reseal our image and get it ready for factory deployment.

1. Delete installation folders and files that have been created of the preloaded applications. These folders can increase the size of a captured .wim file.
2. If the SysPrep Tool is open, close it and open Command Prompt in Administrator mode.
3. Generalize the image by using answer file with additional settings:

```
C:\Windows\System32\Sysprep\sysprep /oobe /generalize /unattend:c:\recovery\oem\Unattend.xml /shutdown
```

## **Remove the Windows 10 in S mode manufacturing key**

If you have manufacturing mode enabled, remove the manufacturing registry key:

1. Open Command Prompt.
2. Remove the registry key:

```
reg delete HKLM\system\ControlSet001\Control\CI\Policy /v ManufacturingMode
```

### **IMPORTANT**

Don't ship a Windows 10 in S mode device with the registry key in place.

## **Finalize and Capture your image**

We'll show you how to finalize and capture a factory image for mass deployment. To start this section, make sure your reference machine is shutdown after running sysprep in the previous section.

1. Boot the Reference computer into WinPE.
2. After WinPE has booted connect *USB-B*.

### **Troubleshooting:**

- If the reference PC boots from its internal HDD, Windows will enter the specialize and OOBE passes. You won't be able to capture a stable and generalized image if any of the configuration passes have completed. If either of those passes have completed, you'll need To generalize the image again. You can do with in Audit Mode (<Ctrl> + <Shift> + <F3> during OOBE). In Audit mode, run the Sysprep command from above. Make sure the PC boots to WinPE on the next restart.
  - If the system still boots to the internal HDD, check the PC's boot priority. Make sure that the USB has a higher boot priority than the internal hard drive.
3. Identify Windows Partition Drive letter using diskpart.
    - a. At the X:\windows\system32> prompt, type diskpart and press the key to start Diskpart.
    - b. At the \DISKPART> prompt type `list volume`.
    - c. Under the "Label" column, locate the volume that is labeled "Windows".
    - d. Note what letter it is has been assigned under the "Ltr" column (Example: C). This is the USB key's drive letter.

```

Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Windows\System32>diskpart

Microsoft DiskPart version 10.0.10240

Copyright (C) 1999-2013 Microsoft Corporation.

On computer: COMPNAME

DISKPART> list vol

  Volume ###  Ltr  Label        Fs  Type        Size     Status      Info
  -----  ---  -----  -----  -----  -----  -----
  Volume 0      System    NTFS  Partition   100 MB  Healthy  System
  Volume 1      C        Windows   NTFS  Partition   465 GB  Healthy  Boot
  Volume 2      Recovery  NTFS  Partition   500 MB  Healthy  Hidden
  Volume 3      D        Removable NTFS  Removable    0 B    No Media

```

- e. Type exit to quit Diskpart.

### (CompactOS Only) Convert installed customizations

This section shows how to reduce the size of ScanState packages.

#### **IMPORTANT**

Only do this step if you are deploying to a device with limited storage. Single instancing impacts the launch performance of some desktop applications.

See [Compact OS](#) for more information.

To reduce the size of your ScanState recovery packages, run the following command from WinPE on your reference device:

```
DISM /Apply-CustomDataImage /CustomDataImage:C:\Recovery\Customizations\apps.ppkg /ImagePath:C:\ /SingleInstance
```

## Capture your image

In this section, we'll tell you how to capture your sysprepped image. You can capture either a [WIM](#) or an [FFU](#).

### **Capture a WIM**

#### **On your reference PC:**

1. Identify Windows Partition Drive letter.
  - a. At the X:\windows\system32> prompt, type diskpart and press the key to start Diskpart.
  - b. At the \DISKPART> prompt type list volume
  - c. Under the "Label" column, locate the volume that is labeled "Windows"
  - d. Note what letter it is has been assigned under the "Ltr" column (Example: C). This is the drive letter that needs to be used
  - e. Type exit to quit Diskpart
2. Capture the image of the windows partition to USB-B. This process takes several minutes.

Note: We recommend using a cache directory when running DISM. In this step we'll create scratchdir on the USB-B key for temporary files, but you can choose any hard drive with available space for your scratch directory.

```
MD e:\scratchdir
Dism /Capture-Image /CaptureDir:C:\ /ImageFile:E:\Images\CustomImage.wim /Name:"CustomImage"
/scratchdir:e:\scratchdir
```

This captures an image called CustomImage.wim to E:\Images. When the image capture is complete, you can shut down your reference PC.

With your image captured, you can skip to [Verify your final image](#).

## Capture an FFU

### On your reference PC:

1. Identify Windows Partition Drive letter.
  - a. At the X:\windows\system32> prompt, type diskpart and press the key to start Diskpart.
  - b. At the \DISKPART> prompt type list disk
  - c. Under the "Disk ####" column, identify the the disk that has the Windows installation, and note the assigned disk number. This will look something like *Disk 0*.
  - d. Type exit to quit Diskpart
2. Capture an image of the windows disk to USB-B. This process takes several minutes.

```
DISM.exe /capture-ffu /imagefile=E:\Images\CustomImage.wim /Name:"CustomImage"
/capturedrive=\\.\PhysicalDrive0 /description:"Windows 10 FFU"
```

This captures an image called CustomImage.wim to E:\Images. When the image capture is complete, you can shut down your reference PC.

## Verify your final image

In this section, we'll cover how to deploy your captured image for testing and verification.

### Deploy your image to the reference device

1. Boot the PC you want to test your image on into WinPE.
2. Run applyimage.bat to deploy the image.

If you captured a WIM called customimage.wim:

```
E:\Deployment\applyimage.bat E:\Images\customimage.wim
```

or

If you captured an FFU called CustomImage.FFU:

```
E:\Deployment\applyimage.bat E:\Images\CustomImage.FFU
```

3. Follow the script's prompts to apply the image.

4. Type `exit` to close WinPE and restart the PC.

### Validate the configuration

Your PC will restart and boot into Windows for the first time.

1. In OOOE, create a dummy user which will be deleted later.
2. Verify that any applications and offline customizations are still in your image and working properly.

Some things to check are:

- Taskbar
- Pinned Apps
- Desktop Wallpaper is set to display the right image
- OEM Information displays correctly
- OEM App ID registry key is set
- Default Theme is the one you chose
- Store apps start properly
- Desktop applications start ok
- Desktop applications applied via SPP start ok
- If you enabled S mode, make sure the manufacturing registry key is removed

## Verify Recovery

1. Verify that your customizations are restored after recovery, and that they continue to function by running the Keep my files and Remove everything features from the following entry points:
  - Settings a. From the Start Menu, click Settings, b. In the Settings app, click Update & security, and then click Recovery. c. Click the Get Started button under Reset this PC and follow the on-screen instructions.
  - Windows RE a. From the Choose an option screen in Windows RE, click Troubleshoot b. Click Reset this PC and then follow the on-screen instructions
2. Verify that recovery media can be created, and verify its functionality by running the bare metal recovery feature: a. Launch Create a recovery drive from Control Panel b. Follow the on-screen instructions to create the USB recovery drive c. Boot the PC from the USB recovery drive d. From the Choose an option screen, click Troubleshoot e. Click Recover from a drive and then follow the on-screen instructions

**Note:** The Push-button reset UI has been redesigned in Windows 10. The Keep my files option in the UI now corresponds to the Keep my files feature. Remove everything corresponds to the Remove everything feature.

## Optimize final image

At this point, you have a Windows image that is almost ready for deployment. In this section, we'll show you how to put the finishing touches on your image so you're ready for deployment.

- Remove unused packages from your image, by exporting a copy of it.

```
Dism /export-image /sourceimagefile:E:\images\customimage.wim /sourceindex:1  
/destinationimagefile:e:\images\MasterImage_Pro.wim
```

## Final shipment

You have to boot a PC at least once to allow the specialize configuration pass of Windows Setup to complete before shipping a PC.

The specialize configuration pass adds hardware-specific information to the PC and is complete when Windows OOOE appears.

Reference the OEM Policy Documentation for more details.

## Reducing Disk Footprints

Throughout this guide, we have shown a few places where you can reduce the disk footprint:

- Using Dism /export-image
- Using Compact OS
- Using Compact OS with Single Instancing

This section shows a few more ways you can gain additional free space.

### Reducing and turning off Hiberfile

Reducing and turning off hiberfile can return between 400MB to 1.5GB OS space on the deployed OS.

#### Reducing Hiberfile by 30%

When you run sysprep.exe with unattend.xml, you can add a FirstLogonCommand that will reduce hiberfile:

```
<unattend xmlns="urn:schemas-microsoft-com:unattend">
  <settings pass="oobeSystem">
    <component name="Microsoft-Windows-Shell-Setup" processorArchitecture=
      <FirstLogonCommands>
        <SynchronousCommand wcm:action="add">
          <CommandLine>powercfg /h /type reduced</CommandLine>
          <Description>Reduce hiberfile size</Description>
          <Order>1</Order>
          <RequiresUserInput>false</RequiresUserInput>
        </SynchronousCommand>
      </FirstLogonCommands>
    </component>
  </settings>
  <cpi:offlineImage cpi:source="wim:c:/images/" xmlns:cpi="urn:schemas-micro
</unattend>
```

#### Turn Off Hiberfile

When you run sysprep.exe with unattend.xml, you can add a FirstLogonCommand that will turn off hiberfile:

```
<unattend xmlns="urn:schemas-microsoft-com:unattend">
  <settings pass="oobeSystem">
    <component name="Microsoft-Windows-Shell-Setup" processorArchitecture=
      <FirstLogonCommands>
        <SynchronousCommand wcm:action="add">
          <CommandLine>powercfg /h Off</CommandLine>
          <Description>Reduce hiberfile size</Description>
          <Order>1</Order>
          <RequiresUserInput>false</RequiresUserInput>
        </SynchronousCommand>
      </FirstLogonCommands>
    </component>
  </settings>
  <cpi:offlineImage cpi:source="wim:c:/images/" xmlns:cpi="urn:schemas-micro
</unattend>
```

Capture your image with the unattend.xml file that contains these settings.

#### Disk footprint with optimizations

The table below shows additional space saved by using compact OS, Single instancing, and reducing or turning Off Hiberfile on 2GB (x86) and 4GB (x64).

FOOTPRINT TYPE	WINDOWS 10 HOME X86 2GB MEMORY	WINDOWS 10 HOME X64 4GB MEMORY
Base Footprint	11.68GB (Additional Space)	15.06GB (Additional Space)
Compact, no single instancing	8.85GB (>2.75GB)	11.3GB (>3.7GB)
Compact, single instanced	7.66GB (>4GB)	10.09GB (>4.75GB)

FOOTPRINT TYPE	WINDOWS 10 HOME X86 2GB MEMORY	WINDOWS 10 HOME X64 4GB MEMORY
Hiberfile off, no compact	10.87GB (>825MB)	13.48GB (>1.5GB)
Hiberfile reduced, no compact	11.27GB (>400MB)	14.15GB (>930MB)

# Sample scripts

7/11/2019 • 19 minutes to read

[Download the sample scripts used in this lab](#)

Copy these scripts to the root of your storage USB drive. Refer to this page to understand what's in the scripts.

The sample scripts ZIP download includes all the scripts below:

## Image deployment scripts

The following scripts set up Windows devices by using either a WIM or an FFU image file, and then give the option to configure push-button reset features.

The following files make up the deployment scripts:

- ApplyImage.bat
- ApplyRecovery.bat
- CreatePartitions-BIOS.txt
- CreatePartitions-BIOS-FFU.txt
- CreatePartitions-UEFI.txt
- CreatePartitions-UEFI-FFU.txt
- HideRecoveryPartitions-BIOS.txt
- HideRecoveryPartitions-UEFI.txt
- CreateRecoveryPartitions-BIOS.txt
- CreateRecoveryPartitions-UEFI.txt

### ApplyImage.bat

Use this script applies a Windows image to a new device.

**Note:** If you copy and paste the contents below to create a .bat file, you may get an error when detecting firmware. For firmware detection to succeed, ensure that the lines that begin `for /f "tokens=2* delims= " %%A` has a tab followed by a space in between `delims=` and `" %%A`.

```
@echo Apply-Image.bat
@echo      Run from the reference device in the WinPE environment.
@echo.
@echo      This script erases the primary hard drive and applies a new image.
@echo.
@echo      Make sure that this script is run from the folder that contains the
@echo      supporting scripts
@echo.
@echo UPDATE (November 2017)
@echo * Added support for FFU deployments.
@echo.
@echo UPDATE (JULY 2016):
@echo * This script stops just after applying the image.
@echo      This gives you an opportunity to add siloed provisioning packages (SPPs)
@echo      so that you can include them in your recovery tools.
@echo.
@echo      After the script is complete, use apply-recovery.bat to finish
@echo      setting up the recovery tools.
@echo.
@echo * This script creates a now includes support for the /EA variables for quicker
@echo      image capture and recovery.
```

```

@echo.
@echo * This script now includes support for the /EA variables for quicker
@echo   image capture and recovery.
@echo.
@echo * This script now checks to see if you're booted into Windows PE.
@echo.
@if not exist X:\Windows\System32 echo ERROR: This script is built to run in Windows PE.
@if not exist X:\Windows\System32 goto END
@if %1==. echo ERROR: To run this script, add a path to a Windows image file.
@if %1==. echo Example: ApplyImage D:\WindowsWithFrench.wim
@if %1==. goto END
@echo ****
@echo == Setting high-performance power scheme to speed deployment ==
@call powercfg /s 8c5e7fda-e8bf-4a96-9a85-a6e23a8c635c
@echo ****
@echo Checking to see the type of image being deployed
@if "%~x1" == ".wim" (GOTO WIM)
@if "%~x1" == ".ffu" (GOTO FFU)
@echo ****
@if not "%~x1" == ".ffu". if not "%~x1" == ".wim" echo Please use this script with a WIM or FFU image.
@if not "%~x1" == ".ffu". if not "%~x1" == ".wim" GOTO END
:WIM
@echo Starting WIM Deployment
@echo ****
@echo Checking to see if the PC is booted in BIOS or UEFI mode.
wpeutil UpdateBootInfo
for /f "tokens=2* delims=      %%A in ('reg query HKLM\System\CurrentControlSet\Control /v PEFirmwareType') DO
SET Firmware=%B
@echo           Note: delims is a TAB followed by a space.
@if x%Firmware%==x echo ERROR: Can't figure out which firmware we're on.
@if x%Firmware%==x echo           Common fix: In the command above:
@if x%Firmware%==x echo           for /f "tokens=2* delims=      "
@if x%Firmware%==x echo           ...replace the spaces with a TAB character followed by a space.
@if x%Firmware%==x goto END
@if %Firmware%==0x1 echo The PC is booted in BIOS mode.
@if %Firmware%==0x2 echo The PC is booted in UEFI mode.
@echo ****
@echo Do you want to create a Recovery partition?
@echo   (If you're going to be working with FFUs, and need
@echo     to expand the Windows partition after applying the FFU, type N).
@SET /P RECOVERY=(Y or N):
@if %RECOVERY%==y. set RECOVERY=Y
@echo Formatting the primary disk...
@if %Firmware%==0x1 echo     ...using BIOS (MBR) format and partitions.
@if %Firmware%==0x2 echo     ...using UEFI (GPT) format and partitions.
@echo CAUTION: All the data on the disk will be DELETED.
@SET /P READY=Erase all data and continue? (Y or N):
@if %READY%==y. set READY=Y
@if not %READY%==Y. goto END
@if %Firmware%==0x1. if %RECOVERY%==Y. diskpart /s CreatePartitions-BIOS.txt
@if %Firmware%==0x1. if not %RECOVERY%==Y. diskpart /s CreatePartitions-BIOS-FFU.txt
@if %Firmware%==0x2. if %RECOVERY%==Y. diskpart /s CreatePartitions-UEFI.txt
@if %Firmware%==0x2. if not %RECOVERY%==Y. diskpart /s CreatePartitions-UEFI-FFU.txt
@echo ****
@echo == Apply the image to the Windows partition ==
@SET /P COMPACTOS=Deploy as Compact OS? (Y or N):
@if %COMPACTOS%==y. set COMPACTOS=Y
@echo Does this image include Extended Attributes?
@echo   (If you're not sure, type N).
@SET /P EA=(Y or N):
@if %EA%==y. set EA=Y
@if %COMPACTOS%==Y. if %EA%==Y.     dism /Apply-Image /ImageFile:%1 /Index:1 /ApplyDir:W:\ /Compact /EA
@if not %COMPACTOS%==Y. if %EA%==Y.     dism /Apply-Image /ImageFile:%1 /Index:1 /ApplyDir:W:\ /EA
@if %COMPACTOS%==Y.     if not %EA%==Y. dism /Apply-Image /ImageFile:%1 /Index:1 /ApplyDir:W:\ /Compact
@if not %COMPACTOS%==Y. if not %EA%==Y. dism /Apply-Image /ImageFile:%1 /Index:1 /ApplyDir:W:\ /
@echo ****
@echo == Copy boot files to the System partition ==
W:\Windows\System32\bcdboot W:\Windows /s S:
@echo ****

```

```

@echo Next steps:
@echo * Add Windows Classic apps (optional):
@echo     DISM /Apply-SiloedPackage /ImagePath:W:\ 
@echo         /PackagePath:"D:\App1.spp" /PackagePath:"D:\App2.spp" ...
@echo.
@echo * Configure the recovery partition with ApplyRecovery.bat
@echo.
@echo * Reboot:
@echo     exit
@GOTO END
:FFU
@echo Starting FFU Deployment
@echo list disk > x:\listdisks.txt
@echo exit >> x:\listdisks.txt
@diskpart /s x:\listdisks.txt
@del x:\listdisks.txt
@echo Enter the disk number of the drive where you're going to deploy your FFU (usually 0).
@SET /P DISKNUMBER=(Enter the Disk Number from above):
@echo This will remove all data from disk %DISKNUMBER%. Continue?
@SET /P ERASEALL=(Y or N):
@if %ERASEALL%==y. set ERASEALL=Y
@if %ERASEALL%==Y DISM /apply-ffu /ImageFile=%1 /ApplyDrive:\\.\PhysicalDrive%DISKNUMBER%
@if not %ERASEALL%==Y GOTO END
@echo FFU applied. Would you like to configure the recovery partition?
@SET /P CONFIGRECOVERY=(Y or N):
@if %CONFIGRECOVERY%==y. SET CONFIGRECOVERY=Y
@if %CONFIGRECOVERY%==Y ApplyRecovery.bat
@if not %CONFIGRECOVERY%==Y GOTO END
:END

```

ApplyImage.bat relies on the following DiskPart scripts, which must be placed in the same folder:

### CreatePartitions scripts

Use these scripts together with DiskPart to format and set up the hard disk partitions for Windows, including recovery tools. Adjust the partition sizes to fill the drive as necessary.

#### CreatePartitions-UEFI.txt

Creates the System, MSR, Windows, and recovery tools partitions for UEFI-based PCs.

This script temporarily assigns these drive letters: System=S, Windows=W, and Recovery=R. The MSR partition doesn't get a letter. The letter W is used to avoid potential drive letter conflicts. After the device reboots, the Windows partition is assigned the letter C, and the other partitions don't receive drive letters.

The Recovery partition must be the partition after the Windows partition to ensure winre.wim can be kept up-to-date during life of the device.

The following diagram shows the resulting partition configuration:

### Disk 0 default partition layout (UEFI-based PCs)



```
rem == CreatePartitions-UEFI.txt ==
rem == These commands are used with DiskPart to
rem     create four partitions
rem     for a UEFI/GPT-based PC.
rem     Adjust the partition sizes to fill the drive
rem     as necessary. ==
select disk 0
clean
convert gpt
rem == 1. System partition =====
create partition efi size=100
rem     ** NOTE: For Advanced Format 4Kn drives,
rem             change this value to size = 260 **
format quick fs=fat32 label="System"
assign letter="S"
rem == 2. Microsoft Reserved (MSR) partition =====
create partition msr size=16
rem == 3. Windows partition =====
rem ==     a. Create the Windows partition =====
create partition primary
rem ==     b. Create space for the recovery tools ===
rem         ** Update this size to match the size of
rem             the recovery tools (winre.wim)
rem             plus some free space.
shrink minimum=500
rem ==     c. Prepare the Windows partition =====
format quick fs=ntfs label="Windows"
assign letter="W"
rem == 4. Recovery partition =====
create partition primary
format quick fs=ntfs label="Recovery"
assign letter="R"
set id="de94bba4-06d1-4d40-a16a-bfd50179d6ac"
gpt attributes=0x8000000000000001
list volume
exit
```

#### CreatePartitions-UEFI-FFU.txt

This script is based off of CreatePartitions-UEFI.txt, but it does not create a recovery partition. This is so that the Windows partition is the last partition on the drive and can be expanded. If this script is used, the recovery partition can be configured later with ApplyRecovery.bat.

```

rem == CreatePartitions-UEFI-FFU.txt ==
rem == These commands are used with DiskPart to
rem     create four partitions
rem     for a UEFI/GPT-based PC.
rem     Adjust the partition sizes to fill the drive
rem     as necessary. ==
select disk 0
clean
convert gpt
rem == 1. System partition =====
create partition efi size=100
rem     ** NOTE: For Advanced Format 4Kn drives,
rem             change this value to size = 260 **
format quick fs=fat32 label="System"
assign letter="S"
rem == 2. Microsoft Reserved (MSR) partition =====
create partition msr size=16
rem == 3. Windows partition =====
rem ==     a. Create the Windows partition =====
create partition primary
rem ==     c. Prepare the Windows partition =====
format quick fs=ntfs label="Windows"
assign letter="W"
list volume
exit

```

#### **CreatePartitions-BIOS.txt**

Creates the System, Windows, and recovery tools partitions for BIOS-based PCs.

This script temporarily assigns these drive letters: System=S, Windows=W, and Recovery=R. The letter W is used to avoid potential drive letter conflicts. After the device reboots, the Windows partition is assigned the letter C, and the other partitions don't receive drive letters.

The Recovery partition must be the partition after the Windows partition to ensure winre.wim can be kept up-to-date during life of the device.

The following diagram shows the resulting partition configuration:

**Disk 0 default partition layout (BIOS-based PCs)**



```

rem == CreatePartitions-BIOS.txt ==
rem == These commands are used with DiskPart to
rem     create three partitions
rem     for a BIOS/MBR-based computer.
rem     Adjust the partition sizes to fill the drive
rem     as necessary. ==
select disk 0
clean
rem == 1. System partition =====
create partition primary size=100
format quick fs=ntfs label="System"
assign letter="S"
active
rem == 2. Windows partition =====
rem ==     a. Create the Windows partition =====
create partition primary
rem ==         b. Create space for the recovery tools
rem             ** Update this size to match the size of
rem                 the recovery tools (winre.wim)
rem                 plus some free space.
shrink minimum=500
rem ==     c. Prepare the Windows partition =====
format quick fs=ntfs label="Windows"
assign letter="W"
rem == 3. Recovery partition =====
create partition primary
format quick fs=ntfs label="Recovery image"
assign letter="R"
set id=27
list volume
exit

```

#### **CreatePartitions-BIOS-FFU.txt**

This script is based off of CreatePartitions-BIOS.txt, but it doesn't create a recovery partition. This is so that the Windows partition is the last partition on the drive and can be expanded. If this script is used, the recovery partition can be configured later with ApplyRecovery.bat.

```

rem == CreatePartitions-BIOS-FFU.txt ==
rem == These commands are used with DiskPart to
rem     create three partitions
rem     for a BIOS/MBR-based computer.
rem     Adjust the partition sizes to fill the drive
rem     as necessary. ==
select disk 0
clean
rem == 1. System partition =====
create partition primary size=100
format quick fs=ntfs label="System"
assign letter="S"
active
rem == 2. Windows partition =====
rem ==     a. Create the Windows partition =====
create partition primary
rem ==         c. Prepare the Windows partition =====
format quick fs=ntfs label="Windows"
assign letter="W"
list volume
exit

```

#### **ApplyRecovery.bat**

Use this script to prepare the Windows recovery partition. This script is called by ApplyImage.bat, but can also be run on its own.

**Note:** If you copy and paste the contents below to create a .bat file, you may get an error when detecting firmware. For firmware detection to succeed, ensure that the lines that begin `for /f "tokens=2* delims= " %%A` has a tab followed by a space in between `delims=` and `" %%A`.

```
@echo == ApplyRecovery.bat ==
@rem ****
@echo Checking to see if the PC is booted in BIOS or UEFI mode.
wpeutil UpdateBootInfo
for /f "tokens=2* delims= " %%A in ('reg query HKLM\System\CurrentControlSet\Control /v PEFFirmwareType') DO
SET Firmware=%B
@echo           Note: delims is a TAB followed by a space.
@if %Firmware%==x echo ERROR: Can't figure out which firmware we're on.
@if %Firmware%==x echo      Common fix: In the command above:
@if %Firmware%==x echo          for /f "tokens=2* delims= "
@if %Firmware%==x echo          ...replace the spaces with a TAB character followed by a space.
@if %Firmware%==x goto END
@if %Firmware%==0x1 echo The PC is booted in BIOS mode.
@if %Firmware%==0x2 echo The PC is booted in UEFI mode.
@echo ****
@echo Do you already have a recovery partition on this disk? (Y or N):
@SET /P RECOVERYEXIST=(Y or N):
@if %RECOVERYEXIST%.==y. set RECOVERYEXIST=Y
@if %RECOVERYEXIST%.==Y. GOTO COPYTOTOOLSPARTITION
@if not %RECOVERYEXIST%.==Y. GOTO CREATEFFURECOVERY
@echo ****
:COPYTOTOOLSPARTITION
@echo == Copy the Windows RE image to the Windows RE Tools partition ==
md R:\Recovery\WindowsRE
xcopy /h W:\Windows\System32\Recovery\Winre.wim R:\Recovery\WindowsRE\
@echo ****
@echo == Register the location of the recovery tools ==
W:\Windows\System32\Reagentsc /Setreimage /Path R:\Recovery\WindowsRE /Target W:\Windows
@echo ****
@IF EXIST W:\Recovery\Customizations\USMT.ppkg (GOTO CUSTOMDATAIMAGEWIM) else goto HIDEWIMRECOVERYTOOLS
:CUSTOMDATAIMAGEWIM
@echo == If Compact OS, single-instance the recovery provisioning package ==
@echo.
@echo      *Note: this step only works if you created a ScanState package called
@echo      USMT.ppkg as directed in the OEM Deployment lab. If you aren't
@echo      following the steps in the lab, choose N.
@echo.
@echo      Options: N: No
@echo      Y: Yes
@echo      D: Yes, but defer cleanup steps to first boot.
@echo          Use this if the cleanup steps take more than 30 minutes.
@echo          defer the cleanup steps to the first boot.
@SET /P COMPACTOS=Deploy as Compact OS? (Y, N, or D):
@if %COMPACTOS%.==y. set COMPACTOS=Y
@if %COMPACTOS%.==d. set COMPACTOS=D
@if %COMPACTOS%.==Y. dism /Apply-CustomDataImage /CustomDataImage:W:\Recovery\Customizations\USMT.ppkg
/ImagePath:W:\ /SingleInstance
@if %COMPACTOS%.==D. dism /Apply-CustomDataImage /CustomDataImage:W:\Recovery\Customizations\USMT.ppkg
/ImagePath:W:\ /SingleInstance /Defer
@echo ****
:HIDEWIMRECOVERYTOOLS
@echo == Hiding the recovery tools partition
if %Firmware%==0x1 diskpart /s %~dp0HideRecoveryPartitions-BIOS.txt
if %Firmware%==0x2 diskpart /s %~dp0HideRecoveryPartitions-UEFI.txt
@echo ****
@echo == Verify the configuration status of the images. ==
W:\Windows\System32\Reagentsc /Info /Target W:\Windows
@echo      (Note: Windows RE status may appear as Disabled, this is OK.)
@echo ****
@echo      All done!
@echo      Disconnect the USB drive from the reference device.
@echo      Type exit to reboot.
@echo.
```

```

GOTO END
:CREATEFFURECOVERY
@echo ****
@echo == Creating the recovery tools partition
@if %Firmware%==0x1 diskpart /s CreateRecoveryPartitions-BIOS.txt
@if %Firmware%==0x2 diskpart /s CreateRecoveryPartitions-UEFI.txt
@echo finding the Windows Drive
@echo ****
@if EXIST C:\Windows SET windowsdrive=C:\
@if EXIST D:\Windows SET windowsdrive=D:\
@if EXIST E:\Windows SET windowsdrive=E:\
@if EXIST W:\Windows SET windowsdrive=W:\
@echo The Windows drive is %windowsdrive%
md R:\Recovery\WindowsRE
@echo ****
@echo Finding Winre.wim
@if EXIST %windowsdrive%\Recovery\WindowsRE\winre.wim SET recoveryfolder=%windowsdrive%\Recovery\WindowsRE\
@if EXIST %windowsdrive%\Windows\System32\Recovery\winre.wim SET
recoveryfolder=%windowsdrive%\Windows\System32\Recovery\
@echo ****
@echo copying Winre.wim
xcopy /h %recoveryfolder%\Winre.wim R:\Recovery\WindowsRE\
@echo ****
@echo == Register the location of the recovery tools ==
%windowsdrive%\Windows\System32\Reagents /Setreimage /Path R:\Recovery\WindowsRE /Target %windowsdrive%\Windows
@echo ****
@if EXIST W:\Recovery\Customizations\USMT.ppkg (GOTO CUSTOMDATAIMAGEFFU) else goto HIDERECOVERYTOOLSFFU
:CUSTOMDATAIMAGEFFU
@echo == If Compact OS, single-instance the recovery provisioning package ==
@echo.
@echo      *Note: this step only works if you created a ScanState package called
@echo      USMT.ppkg as directed in the OEM Deployment lab. If you aren't
@echo      following the steps in the lab, choose N.
@echo.
@echo      Options: N: No
@echo              Y: Yes
@echo              D: Yes, but defer cleanup steps to first boot.
@echo                  Use this if the cleanup steps take more than 30 minutes.
@echo                  defer the cleanup steps to the first boot.
@SET /P COMPACTOS=Deploy as Compact OS? (Y, N, or D):
@if %COMPACTOS%==y set COMPACTOS=Y
@if %COMPACTOS%==d set COMPACTOS=D
@if %COMPACTOS%==Y. dism /Apply-CustomDataImage
/CUSTOMDATAIMAGE:%windowsdrive%\Recovery\Customizations\USMT.ppkg /ImagePath:%windowsdrive% /SingleInstance
@if %COMPACTOS%==D. dism /Apply-CustomDataImage
/CUSTOMDATAIMAGE:%windowsdrive%\Recovery\Customizations\USMT.ppkg /ImagePath:%windowsdrive% /SingleInstance
/Defer
:HIDERECOVERYTOOLSFFU
@rem ****
@echo == Hiding the recovery tools partition
@if %Firmware%==0x1 diskpart /s HideRecoveryPartitions-BIOS.txt
@if %Firmware%==0x2 diskpart /s HideRecoveryPartitions-UEFI.txt
@echo ****
@echo == Verify the configuration status of the images. ==
%windowsdrive%\Windows\System32\Reagents /Info /Target %windowsdrive%\Windows
@echo      (Note: Windows RE status may appear as Disabled, this is OK.)
@echo ****
@echo      All done!
@echo      Disconnect the USB drive from the reference device.
@echo      Type exit to reboot.
@GOTO END
:END

```

ApplyRecovery.bat relies on the following DiskPart scripts, which must be placed in the same folder:

**CreateRecoveryPartitions-UEFI.txt**

```
rem == CreateRecoveryPartitions-UEFI.txt ==
rem == Find and label the Windows partition
rem ** This script assumes the partition layout from
rem     CreatePartitions-UEFI or CreatePartitions-UEFI-FFU
select disk 0
select partition 3
assign letter="W"
rem == Extend the Windows partition
extend
rem == Create space for the recovery tools
rem ** Update this size to match the size of
rem     the recovery tools (winre.wim)
rem     plus some free space.
shrink minimum=500
rem == Create recovery partition
create partition primary
format quick fs=ntfs label="Recovery"
assign letter="R"
set id="de94bba4-06d1-4d40-a16a-bfd50179d6ac"
gpt attributes=0x8000000000000001
list volume
exit
```

#### CreateRecoveryPartitions-BIOS.txt

```
rem == CreateRecoveryPartitions-BIOS.txt ==
rem == Find and label the Windows partition
rem ** This script assumes the partition layout from
rem     CreatePartitions-BIOS or CreatePartitions-BIOS-FFU
select disk 0
select partition 2
assign letter="W"
rem == Extend the Windows partition ==
shrink minimum=500
extend
rem == Create space for the recovery tools
rem ** Update this size to match the size of the
rem     recovery tools (winre.wim), plus some free space.
shrink minimum=500
rem == Prepare the recovery partition =====
select disk 0
create partition primary
format quick fs=ntfs label="Recovery image"
assign letter="R"
set id=27
list volume
exit
```

#### HideRecoveryPartitions-UEFI.txt

```
rem === HideRecoveryPartitions-UEFI.txt ===
rem ** This script assumes the partition layout from
rem     CreatePartitions-UEFI or CreatePartitions-UEFI-FFU
select disk 0
select partition 4
set id=de94bba4-06d1-4d40-a16a-bfd50179d6ac
gpt attributes=0x8000000000000001
remove
list volume
exit
```

#### HideRecoveryPartitions-BIOS.txt

```
rem === HideRecoveryPartitions-BIOS.txt ===
rem ** This script assumes the partition layout from
rem     CreatePartitions-BIOS or CreatePartitions-BIOS-FFU
select disk 0
select partition 3
set id=27
remove
list volume
exit
```

## Start layout (LayoutModification.xml)

The Start tile layout in Windows 10 provides OEMs the ability to append tiles to the default Start layout to include Web links, secondary tiles, Windows apps, and Windows desktop applications. OEMs can use this layout to make it applicable to multiple regions or markets without duplicating a lot of the work. In addition, OEMs can add up to three default apps to the frequently used apps section in the system area, which delivers system-driven lists o the user including important or frequently accessed system locations and recently installed apps.

To take advantage of all these new features and have the most robust and complete Start customization experience for Windows 10, consider creating a LayoutModification.xml file. This file specifies how the OEM tiles should be laid out in Start. For more information about how to customize the new Start layout, see the topic [Customize the Windows 10 Start screen](#) in the Windows 10 Partner Documentation.

Sample **LayoutModification.xml**:

```

<LayoutModificationTemplate
    xmlns="http://schemas.microsoft.com/Start/2014/LayoutModification"
    xmlns:defaultlayout="http://schemas.microsoft.com/Start/2014/FullDefaultLayout"
    xmlns:start="http://schemas.microsoft.com/Start/2014/StartLayout"
    Version="1">
    <RequiredStartGroupsCollection>
        <RequiredStartGroups
            Region="DE|ES|FR|GB|IT|US">
            <AppendGroup Name="Fabrikam Group 1">
                <start:DesktopApplicationTile
                    DesktopApplicationID="Microsoft.Windows.Explorer"
                    Size="2x2"
                    Row="0"
                    Column="4"/>
            </AppendGroup>
            <AppendGroup
                Name="Fabrikam Group 2">
                <start:Tile AppUserModelID="Microsoft.MicrosoftEdge_8wekyb3d8bbwe!MicrosoftEdge"
                    Size="2x2"
                    Row="0"
                    Column="0"/>
                <!-- <start:Tile AppUserModelID="App2!App" Size="2x2" Row="2" Column="0"/> Update the APUMID to
reflect the app you installed with the region specified -->
            </AppendGroup>
        </RequiredStartGroups>
        <RequiredStartGroups>
            <AppendGroup Name="Fabrikam Group 1">
                <start:SecondaryTile
                    AppUserModelID="Microsoft.MicrosoftEdge_8wekyb3d8bbwe!MicrosoftEdge"
                    TileID="MyWebLinkTile"
                    Arguments="http://www.fabrikam.com"
                    DisplayName="Fabrikam"
                    Square150x150LogoUri="ms-appx:///Assets/MicrosoftEdgeSquare150x150.png"
                    Wide310x150LogoUri="ms-appx:///Assets/MicrosoftEdgeWide310x150.png"
                    ShowNameOnSquare150x150Logo="true"
                    ShowNameOnWide310x150Logo="false"
                    BackgroundColor="#FF112233"
                    Size="2x2"
                    Row="0"
                    Column="2"/>
                <!-- <start:Tile AppUserModelID="App2!App" Size="2x2" Row="2" Column="0"/> Update the APUMID to
reflect the app you installed with no specific region -->
            </AppendGroup>
        </RequiredStartGroups>
    </RequiredStartGroupsCollection>
    <AppendOfficeSuite/>
    <AppendOfficeSuiteChoice Choice="DesktopBridgeSubscription"/>
</LayoutModificationTemplate>

```

## TaskbarLayoutModification.xml

You can pin apps to the taskbar. To learn more, see [OEM Taskbar tiles](#).

```

<?xml version="1.0" encoding="utf-8"?>
<LayoutModificationTemplate
    xmlns="http://schemas.microsoft.com/Start/2014/LayoutModification"
    xmlns:defaultlayout="http://schemas.microsoft.com/Start/2014/FullDefaultLayout"
    xmlns:start="http://schemas.microsoft.com/Start/2014/StartLayout"
    xmlns:taskbar="http://schemas.microsoft.com/Start/2014/TaskbarLayout"
    Version="1">

    <CustomTaskbarLayoutCollection PinListPlacement="Replace">
        <defaultlayout:TaskbarLayout>
            <taskbar:TaskbarPinList>
                <taskbar:UWA AppUserModelID="Microsoft.Windows.Photos_8wekyb3d8bbwe!App" />
                <taskbar:DesktopApp DesktopApplicationLinkPath="%ALLUSERSPROFILE%\Microsoft\Windows\Start
Menu\Programs\Accessories\Paint.lnk"/>
            </taskbar:TaskbarPinList>
        </defaultlayout:TaskbarLayout>
        <defaultlayout:TaskbarLayout Region="US|GB">
            <taskbar:TaskbarPinList>
                <taskbar:DesktopApp DesktopApplicationLinkPath="%APPDATA%\Microsoft\Windows\Start
Menu\Programs\Accessories\Notepad.lnk" />
                <taskbar:UWA AppUserModelID="Microsoft.WindowsCalculator_8wekyb3d8bbwe!App" />
            </taskbar:TaskbarPinList>
        </defaultlayout:TaskbarLayout>
        <defaultlayout:TaskbarLayout Region="CN|TW">
            <taskbar:TaskbarPinList>
                <taskbar:DesktopApp DesktopApplicationLinkPath="%APPDATA%\Microsoft\Windows\Start
Menu\Programs\Accessories\Notepad.lnk" />
                <taskbar:UWA AppUserModelID="Microsoft.Windows.Photos_8wekyb3d8bbwe!App" />
                <taskbar:DesktopApp DesktopApplicationLinkPath="%ALLUSERSPROFILE%\Microsoft\Windows\Start
Menu\Programs\Accessories\Paint.lnk"/>
            </taskbar:TaskbarPinList>
        </defaultlayout:TaskbarLayout>
    </CustomTaskbarLayoutCollection>
</LayoutModificationTemplate>

```

## BootToAudit

Add an answer file to the Windows image in C:\mount\windows\Windows\Panther\unattend.xml to instruct it to boot into audit mode. You can create this answer file in Windows System Image Manager.

### BootToAudit-x64

```

<?xml version="1.0" encoding="utf-8"?>
<unattend xmlns="urn:schemas-microsoft-com:unattend">
    <!-- BootToAudit-x64.xml -->
    <settings pass="oobeSystem">
        <component name="Microsoft-Windows-Deployment" processorArchitecture="amd64"
publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS"
xmlns:wcm="http://schemas.microsoft.com/WMICConfig/2002/State" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
            <Reseal>
                <Mode>Audit</Mode>
            </Reseal>
        </component>
    </settings>
</unattend>

```

## Keeping Windows settings through a recovery

Windows doesn't automatically save settings created through unattend.xml setup files, nor Windows Start Menu customizations created with LayoutModification.xml during a full-system reset, nor first-login info from oobe.xml.

To make sure your customizations are saved, that includes steps to put the unattend.xml, LayoutModification.xml, and oobe.xml files back into place. Here's some sample scripts that show how to retain these settings and put them back into the right spots. Save copies of unattend.xml, LayoutModification.xml, oobe.xml, plus these two text files: ResetConfig.xml and EnableCustomizations.cmd, in C:\Recovery\OEM\:

### ResetConfig.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- ResetConfig.xml -->
<Reset>
  <Run Phase="BasicReset_AfterImageApply">
    <Path>EnableCustomizations.cmd</Path>
    <Duration>2</Duration>
  </Run>
  <Run Phase="FactoryReset_AfterImageApply">
    <Path>EnableCustomizations.cmd</Path>
    <Duration>2</Duration>
  </Run>
</Reset>
```

### EnableCustomizations.cmd

```
rem EnableCustomizations.cmd

rem Set the variable %TARGETOS%      (Typically this is C:\Windows)
for /F "tokens=1,2,3 delims= " %%A in ('reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RecoveryEnvironment" /v TargetOS') DO SET TARGETOS=%C

rem Set the variable %TARGETOSDRIVE% (Typically this is C:)
for /F "tokens=1 delims=\" %%A in ('Echo %TARGETOS%') DO SET TARGETOSDRIVE=%A

rem Add back Windows settings, Start menu, and OOBE.xml customizations
copy "%TARGETOSDRIVE%\Recovery\OEM\Unattend.xml" "%TARGETOS%\Panther\Unattend.xml" /y
copy "%TARGETOSDRIVE%\Recovery\OEM\LayoutModification.xml"
"%TARGETOSDRIVE%\Users\Default\AppData\Local\Microsoft\Windows\Shell\LayoutModification.xml" /y
xcopy "%TARGETOSDRIVE%\Recovery\OEM\OOBE\Info" "%TARGETOS%\System32\Info\" /s

rem Recommended: Create a pagefile for devices with 1GB or less of RAM.
wpeutil CreatePageFile /path=%TARGETOSDRIVE%\PageFile.sys /size=256
```

To learn more about using extensibility points for push-button reset, see [Add extensibility scripts to push-button reset](#).

## Reinstall Windows inbox apps

Reinstall Windows apps after adding a new language. You can reinstall the apps without removing them first.

### ReinstallInboxApps-x64.cmd

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.Getstarted_8wekyb3d8bbwe.appxbundle
/licensepath:.\Appx\Microsoft.Getstarted_8wekyb3d8bbwe.xml
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x64.1.6.appx
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x86.1.6.appx
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x64.1.6.appx
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x86.1.6.appx

DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.GetHelp_8wekyb3d8bbwe.appxbundle
/licensepath:.\Appx\Microsoft.GetHelp_8wekyb3d8bbwe.xml
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx
```





```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.WindowsAlarms_8wekyb3d8bbwe.appxbundle  
/licensepath:.Appx\Microsoft.WindowsAlarms_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.WindowsCalculator_8wekyb3d8bbwe.appxbundle  
/licensepath:.Appx\Microsoft.WindowsCalculator_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.WindowsCamera_8wekyb3d8bbwe.appxbundle  
/licensepath:.Appx\Microsoft.WindowsCamera_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x64.1.7.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x86.1.7.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x64.1.7.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x86.1.7.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.WindowsCommunicationsApps_8wekyb3d8bbwe.appxbundle  
/licensepath:.\\Appx\\Microsoft.WindowsCommunicationsApps_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\\Temp\\Lab\\Apps\\Inbox\\amd64\\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\\Temp\\Lab\\Apps\\Inbox\\amd64\\Microsoft.VCLibs.x86.14.00.appx  
/dependencypackagepath:C:\\Temp\\Lab\\Apps\\Inbox\\amd64\\Microsoft.Advertising.Xaml.x64.10.0.appx  
/dependencypackagepath:C:\\Temp\\Lab\\Apps\\Inbox\\amd64\\Microsoft.Advertising.Xaml.x86.10.0.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.WindowsFeedbackHub_8wekyb3d8bbwe.appxbundle  
/licensepath:.Appx\Microsoft.WindowsFeedbackHub_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x64.1.7.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x86.1.7.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x64.1.7.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x86.1.7.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.WindowsMaps_8wekyb3d8bbwe.appxbundle  
/licensepath:.\Appx\Microsoft.WindowsMaps_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x64.1.7.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x86.1.7.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x64.1.7.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x86.1.7.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.WindowsSoundRecorder_8wekyb3d8bbwe.appxbundle  
/licensepath:.Appx\Microsoft.WindowsSoundRecorder_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.WindowsStore_8wekyb3d8bbwe.appxbundle  
/licensepath:.\Appx\Microsoft.WindowsStore_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x64.1.6.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x86.1.6.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x64.1.6.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x86.1.6.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\x64\Microsoft.XboxApp_8wekyb3d8bbwe.appxbundle
```

```
/licensepath:.\\Appx\\Microsoft.XboxApp_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\\Temp\\Lab\\Apps\\Inbox\\amd64\\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\\Temp\\Lab\\Apps\\Inbox\\amd64\\Microsoft.VCLibs.x86.14.00.appx  
/dependencypackagepath:C:\\Temp\\Lab\\Apps\\Inbox\\amd64\\Microsoft.NET.Native.Framework.x64.1.6.appx  
/dependencypackagepath:C:\\Temp\\Lab\\Apps\\Inbox\\amd64\\Microsoft.NET.Native.Framework.x86.1.6.appx  
/dependencypackagepath:C:\\Temp\\Lab\\Apps\\Inbox\\amd64\\Microsoft.NET.Native.Runtime.x64.1.6.appx  
/dependencypackagepath:C:\\Temp\\Lab\\Apps\\Inbox\\amd64\\Microsoft.NET.Native.Runtime.x86.1.6.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.XboxGameOverlay_8wekyb3d8bbwe.appxbundle  
/licensepath:.Appx\Microsoft.XboxGameOverlay_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.XboxIdentityProvider_8wekyb3d8bbwe.appxbundle  
/licensepath:.Appx\Microsoft.XboxIdentityProvider_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x64.1.6.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x86.1.6.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x64.1.6.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x86.1.6.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.XboxSpeechToTextOverlay_8wekyb3d8bbwe.appxbundle  
/licensepath:.\\Appx\Microsoft.XboxSpeechToTextOverlay_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.ZuneMusic_8wekyb3d8bbwe.appxbundle  
/licensepath:.\\Appx\\Microsoft.ZuneMusic_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.ZuneVideo_8wekyb3d8bbwe.appxbundle  
/licensepath:.Appx\Microsoft.ZuneVideo_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\microsoft.print3d_8wekyb3d8bbwe.appxbundle  
/licensepath:.Appx\microsoft.print3d_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencyvpackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.Xbox.TCUI_8wekyb3d8bbwe.appxbundle  
/licensepath:.\\Appx\Microsoft.Xbox.TCUI_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x64.1.6.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Framework.x86.1.6.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x64.1.6.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.NET.Native.Runtime.x86.1.6.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.XboxGamingOverlay_8wekyb3d8bbwe.appxbundle  
/licensepath:.\\Appx\Microsoft.XboxGamingOverlay_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx
```

```
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage  
/packagepath:C:\Temp\Lab\Apps\Inbox\x64\Microsoft.MixedReality.Portal_8wekyb3d8bbwe.appxbundle  
/licensepath:.\\Appx\Microsoft.MixedReality.Portal_8wekyb3d8bbwe.xml  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\x64\Microsoft.VCLibs.x64.14.00.appx  
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\x64\Microsoft.VCLibs.x86.14.00.appx
```

```

DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.ScreenSketch_8wekyb3d8bbwe.appxbundle
/licensepath:.\\Appx\Microsoft.ScreenSketch_8wekyb3d8bbwe.xml
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx

DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.YourPhone_8wekyb3d8bbwe.appxbundle
/licensepath:.\\Appx\Microsoft.YourPhone_8wekyb3d8bbwe.xml
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx

DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.HEIFImageExtension_8wekyb3d8bbwe.x64.appx
/licensepath:.\\Appx\Microsoft.HEIFImageExtension_8wekyb3d8bbwe.x64.xml
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx

DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VP9VideoExtensions_8wekyb3d8bbwe.x64.appx
/licensepath:.\\Appx\Microsoft.VP9VideoExtensions_8wekyb3d8bbwe.x64.xml
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx

DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.WebpImageExtension_8wekyb3d8bbwe.x64.appx
/licensepath:.\\Appx\Microsoft.WebpImageExtension_8wekyb3d8bbwe.x64.xml
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx

@echo Optional codecs - the following packages are optional depending on your codec choices and needs
DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.HEVCVideoExtension_8wekyb3d8bbwe.x64.appx
/licensepath:.\\Appx\Microsoft.HEVCVideoExtension_8wekyb3d8bbwe.x64.xml
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx

DISM /image:C:\Mount\Windows /add-ProvisionedAppxPackage
/packagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.MPEG2VideoExtension_8wekyb3d8bbwe.x64.appx
/licensepath:.\\Appx\Microsoft.MPEG2VideoExtension_8wekyb3d8bbwe.x64.xml
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x64.14.00.appx
/dependencypackagepath:C:\Temp\Lab\Apps\Inbox\amd64\Microsoft.VCLibs.x86.14.00.appx

```

## Find drive letters with a script

Use this script in Windows PE to identify a drive that has a folder called "Images."

```

@echo Find a drive that has a folder titled Images.
@for %%a in (C D E F G H I J K L M N O P Q R S T U V W X Y Z) do @if exist %%a:\Images\ set IMAGESDRIVE=%%a
@echo The Images folder is on drive: %IMAGESDRIVE%
@dir %IMAGESDRIVE%:\Images /w

```

# System builder deployment of Windows 10 for desktop editions

1/25/2019 • 26 minutes to read

This guide demonstrates how to create customized Windows 10 images that system builders can use to deploy to a line of devices, complete with customizations like apps, drivers, languages, and settings. We show how to make customizations both online and offline. We cover 64-bit and 32-bit Windows 10 for desktop editions (Home, Pro, Enterprise, and Education).

## Prepare your lab environment

For your work PC (technician PC): If you plan to deploy only x64 devices, you can use either a Windows 10 x86 or x64 PC. However, if you plan to deploy x86 devices, you'll need an x86 PC for some steps.

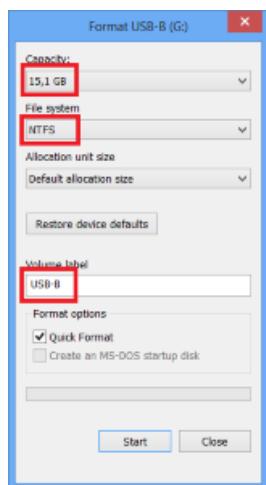
Before starting the deployment procedure, you need to download the kits that will be used throughout the guide. Go to the [Device Partner Center](#) > **Downloads and Installation** > **Understanding ADKs and OPKs**. For a list of resources and kits that will be used and where to obtain them, see [What you will need and where to get it](#).

For this guide, we use two USB drives. USB-A will be used to boot the system in Windows Preinstallation Environment (WinPE). USB-B will be used to move files between computers, store deployment and recovery scripts, and store and apply created images. ([You can also format a single USB drive to store both WinPE and your images](#)).

USB HARD DRIVE NAME	FORMAT	MINIMUM SIZE
USB-A	FAT32	~4GB
USB-B	NTFS	~16GB x86 ~32GB amd64

### Creating my USB-B

1. Format your USB drive and name it as follows:



2. Then download [USB-B.zip](#) from the Microsoft Download Center. Save the .zip file to USB-B and extract the contents there. The contents of the configuration files included in USB-B are examples that you may change according to your branding and manufacturing choices. However, file names and hierarchy of the folders

and files must be the same as demonstrated below in order to align your deployment procedure with this guide.

## Customizations throughout the document

PASS	SETTING	ACTION
<b>WinPE</b>	Setup UI Language	EN-US
	User Data	Preinstallation Product Key for ODR - Defined
<b>Specialize</b>	Internet Explorer Home Page	in the answer file
	OEM Name	Defined in the answer file
	OEM Logo	Defined in the answer file
	Model	Defined in the answer file
	Support Info	Defined in the answer file
<b>OOBE System</b>	Reseal	Audit/OOBE
	StartTiles	Square Tiles / SquareOrDesktopTiles set to pin only desktop apps
	TaskbarLinks (up to 6 pinned .lnk files)	Paint and Control Panel shortcuts have been set
	Themes	Custom Theme with the OEM logo as the desktop background has been set
	Visual Effects	SystemDefaultBackground set

## Additional customizations

### Product deployment

- Office Single Image v16.5 OPK preloaded

### Image customization

- Adding language interface packs to Windows
- Adding drivers and update packages
- Adding OEM-specific logo and background files to Windows
- Image size optimization
- Pinning desktop applications to start screen

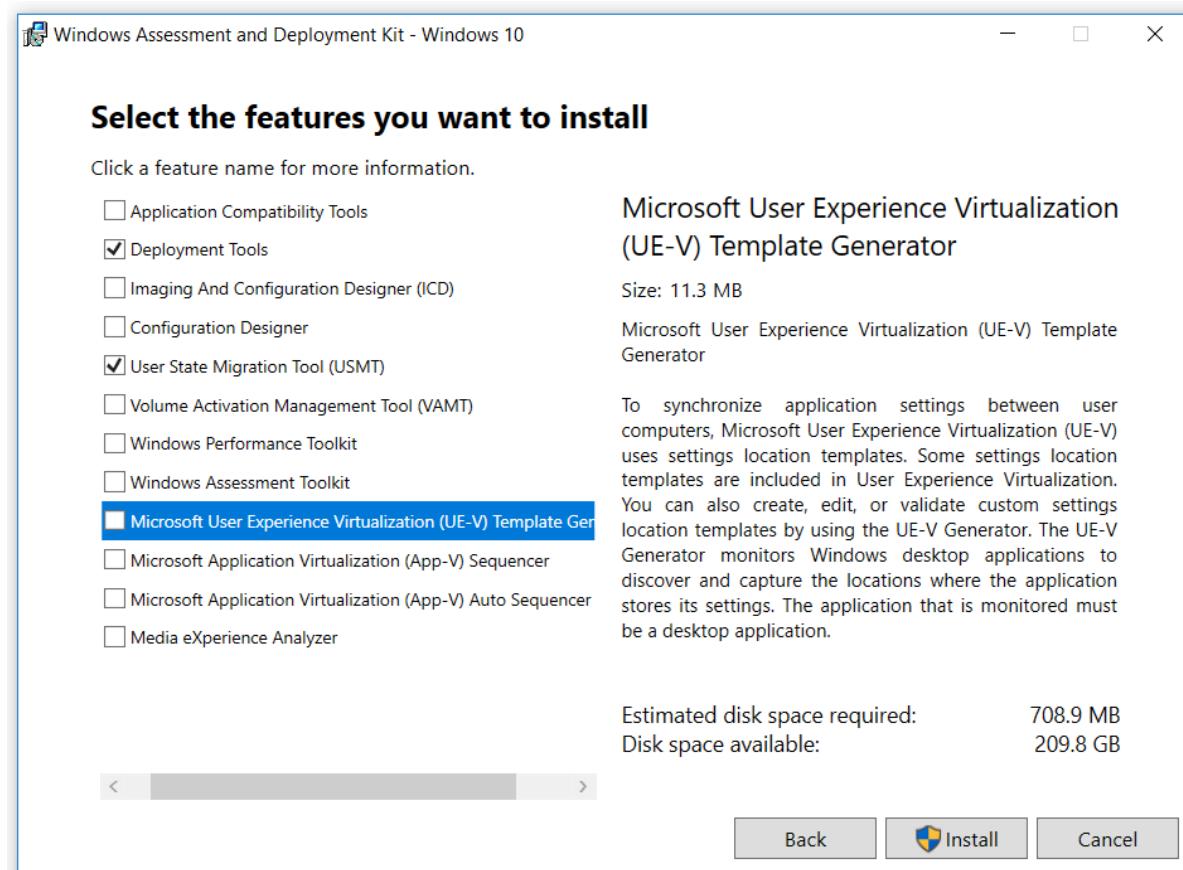
## Create a USB drive that can boot to WinPE

You must use the matching version of Windows ADK for the images being customized. For example, if you're building an image for Windows 10, version 1809, use the Windows ADK for Windows 10, version 1809. For more

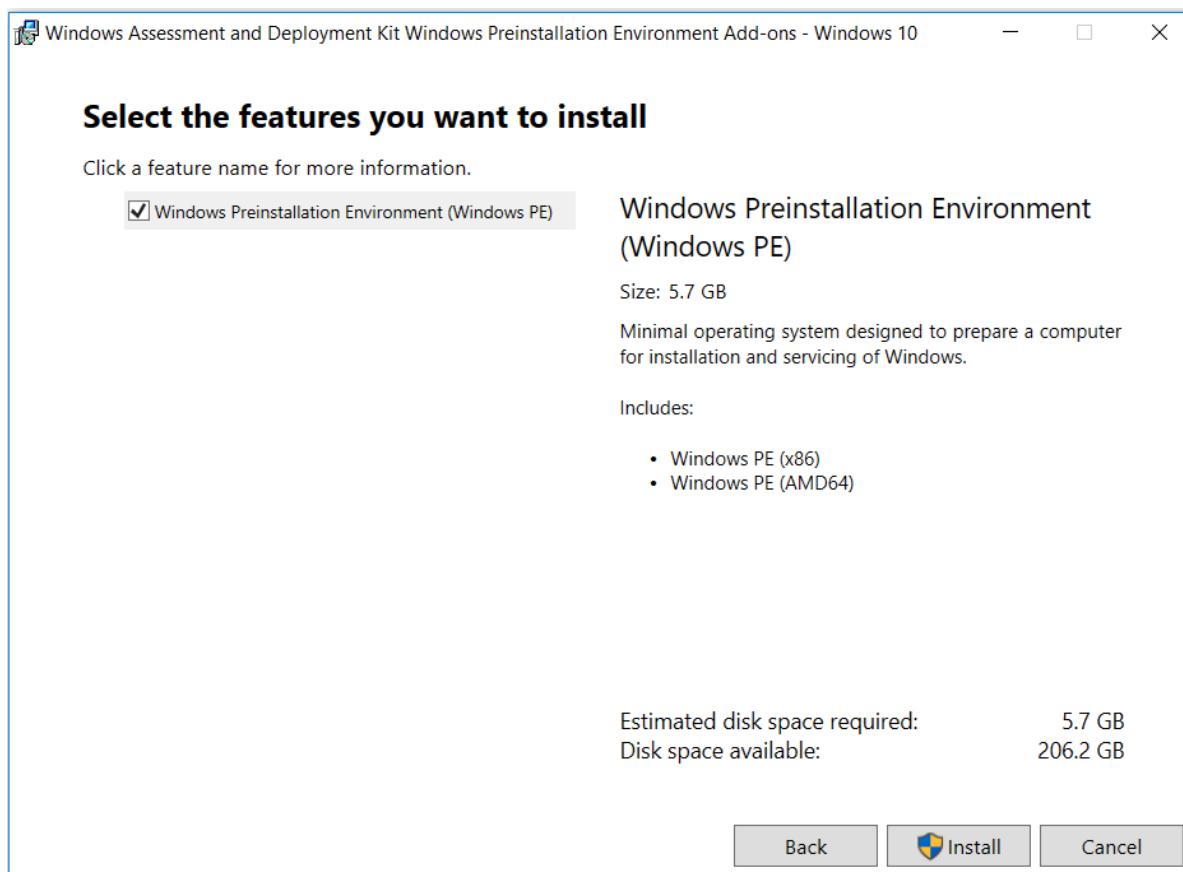
details about the Windows ADK, see the [Windows 10 ADK Documentation Homepage](#).

Visit [Download the Windows ADK and the Windows PE Add-On](#) to download the ADK.

1. Install the Windows ADK, including the **Deployment Tools** and **User State Migration Tool (USMT)** features.



2. From the same page, download the **Windows PE Add-on for the ADK**.



3. Press the Windows key to display the **Start** menu. Type:

```
Deployment and Imaging Tools Environment
```

Right-click the name of the tool, and then click **Run as administrator**.

4. Windows ADK allows you to create **Windows Preinstallation Environment**. Copy base WinPE to new folder.

If you use an **x64** Windows 10 image, copy the x64 WinPE folder structure:

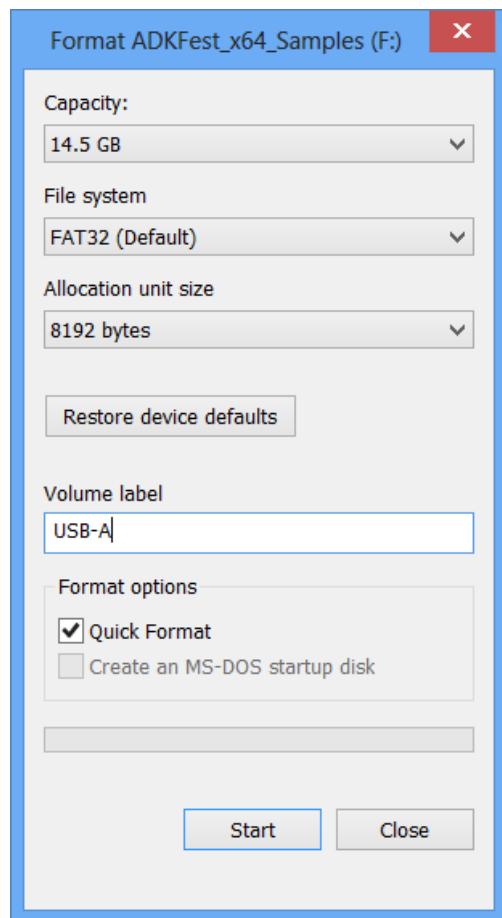
```
Copytree amd64 C:\winpe_amd64
```

If you use an **x86** Windows 10 image, copy the x86 WinPE folder structure:

```
Copytree x86 C:\winpe_x86
```

5. You may [add packages and/or drivers to WinPE](#) here, if you need them. Typically, the built-in WinPE drivers are enough.

6. Connect a USB drive that is at least 4 GB. Format it as shown in this diagram:



7. Make the inserted USB a new WinPE bootable USB.

If you use an **x64** Windows 10 image, make an x64 WinPE USB:

```
MakeWinPEMedia /UFD C:\winpe_amd64 F:
```

If you use an **x86** Windows 10 image, make an x86 WinPE USB:

```
MakeWinPEMedia /UFD C:\winpe_x86 F:
```

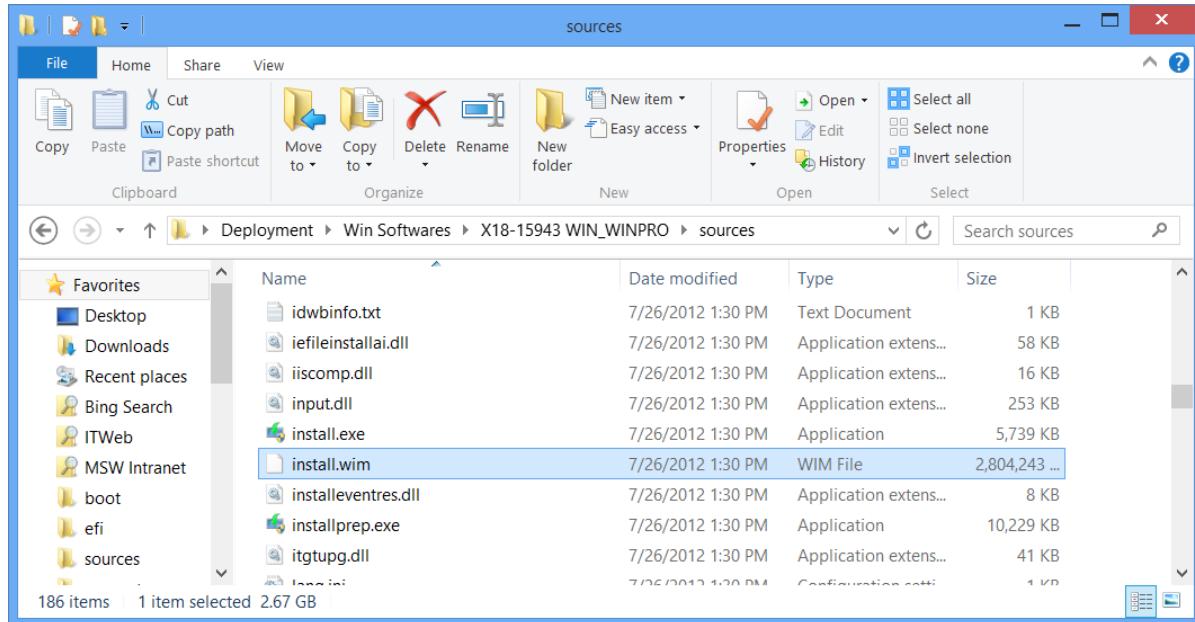
(Where F: is the drive letter of USB)

## Install Windows with basic customizations

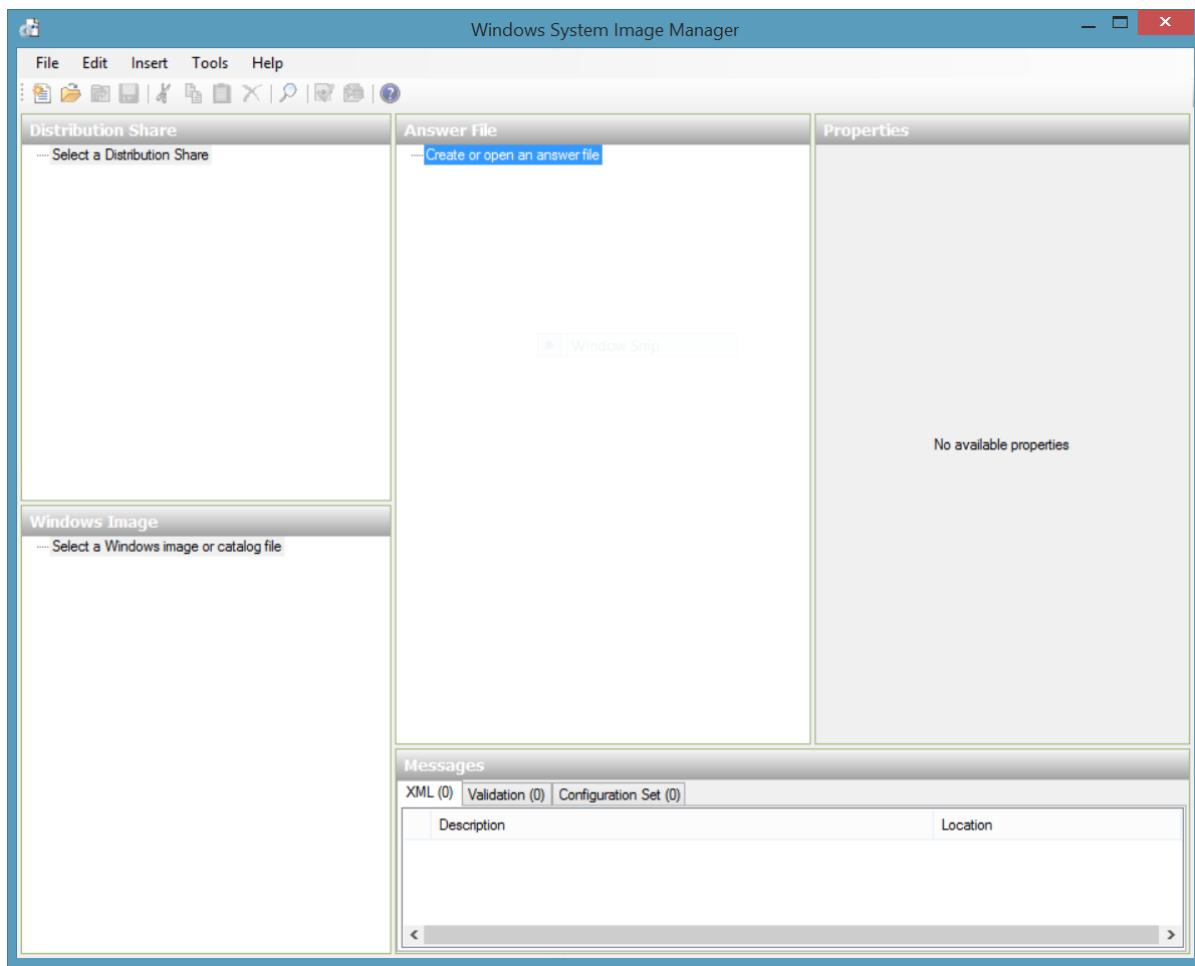
Use Windows 10 x86/x64 DVD media from a Microsoft Authorized Distributor.

See the [Windows Guidelines for System Builders](#) and [Windows Policy for System Builders](#) for information on how to tailor the customizations in your unattend.xml file.

1. Copy the `sources\Install.wim` file from the directory in the Windows 10 media that you will be deploying to your local Desktop (~3gb).



2. Run **Windows System Image Manager** to start creating an answer file from scratch. This tool allows you to create or manage your answer files in an easy and organized manner.



3. Navigate to **File > Select Windows Image**. Browse to your local desktop and select **Install.wim**. A catalog file (.clg) will be created for the specified wim.

Troubleshoot: Catalog creation may fail due to several reasons. Please make sure install.wim has read/write permissions. If you continue getting error, make sure correct architecture (x86 or x64) Windows 10 is installed on technician PC. If you are creating catalog for x64 Windows 10 image, you are required to use x64 Windows 10 installed on x64 Windows 10 computer. Install.wim image and Windows 10 ADK versions must be the same.

4. Open a sample answer file or create a new one. `USB-B\AnswerFiles\Unattend.xml` is the sample answer file included on USB-B.
5. Click **OK** to associate the answer file with the Windows Image.
6. To add a driver to Windows PE, click **Insert** select **Driver Path** and select pass **1 windowsPE** and then browse to the driver. Note: This step is optional and only required if a third-party driver is needed for use in the Windows Preinstallation Environment.
7. To add a package, click **Insert**, select **Package**, and then browse to the package you want to add. This step is optional.

#### Customize the answer file

Troubleshoot: A blank character in **specialize | Microsoft-Windows-Shell-Setup | Computer Name** will result in Windows installation failure.

1. See `USB-B\AnswerFiles\Unattend.xml` for an example of an answer file that has basic customizations. -

You may use the sample answer file and modify relevant parts or start from scratch by specifying some basic customizations.

Please see and use the Windows 10 default product key from [Device Partner Center](#) listed under **Default**

**product keys** tab.

2. Add a product key that matches the Windows edition. This key isn't used to activate Windows, so you can reuse the same key for multiple installations:

- In the **Answer File** pane, select **Components\1\_windowsPE\amd64\_Microsoft-Windows-Setup\_neutral\UserData\ProductKey**. In the **ProductKey Properties** pane, under **Settings**, enter the value next to Key.

Important: These product keys *cannot* be used for activation. You will need to type a software product key during the installation process for activation. These keys will be removed when sysprep generalize is run. The end user will be required to type the unique product key from the Certificate of Authenticity (COA) label when first booting Windows 10.

3. Add your support information:

In the **Answer File** pane, select **Components\4\_specialize\amd64\_Microsoft-Windows-Shell-Setup\_neutral\OEMInformation**.

In the **OEMInformation Properties** pane, in the **Settings** section, update the following values: company name (Manufacturer), hours (SupportHours), phone number (SupportPhone), and website (SupportURL).

4. Prepare your computer to boot to audit mode after the Windows installation is complete:

In the **Windows Image** pane, expand **Components**, right-click **amd64\_Microsoft-Windows-Deployment**, and then select **Add Setting to Pass 7\_oobeSystem**.

In the **Answer File** pane, select **Components\7\_oobeSystem\amd64\_Microsoft-Windows-Deployment\_neutral\Reseal**.

In the **Reseal Properties** pane, in the **Settings** section, add the following value: Mode =Audit.

5. Set the Internet Explorer home page:

In the **Windows Image** pane, right-click **amd64\_Microsoft-Windows-IE-InternetExplorer**, and then select **Add Setting to Pass 4\_specialize**.

In the **Answer File** pane, select **Components\4\_specialize\amd64\_Microsoft-Windows-Microsoft-Windows-IE-InternetExplorer\_neutral**.

In the **IE-InternetExplorer Properties** pane, in the **Settings** section, select Home\_page, and add the URL of your website.

6. OEMs can specify **Disk Configuration** which is used to create/modify disk partitions and set image installation partition. This step is optional and configuration is included in the sample answer file USB-B\AnswerFiles\Unattend.xml.

Save the answer file to USB-B\AnswerFiles\Unattend.xml and close Windows SIM.

## Update images for each model: offline servicing

Before mounting and editing the image, make a copy. Use a filename that describes the changes you want to make for this model-specific image you're making, for example:

```
Dism /export-image /sourceimagefile:e:\images\install.wim /sourceindex:2  
/destinationimagefile:e:\images\modelspecificimage.wim
```

### Mount images

1. Mount Windows image (ModelSpecificImage.wim). This process extracts the contents of the image file to a

location where you can view and modify the mounted image.

```
Md C:\mount\windows  
Dism /Mount-Image /ImageFile:E:\Images\ModelSpecificImage.wim /Index:1 /MountDir:C:\mount\windows
```

Where E:\ is the drive letter of USB-B.

## 2. Mount Windows RE Image file.

```
Md c:\mount\winre  
Dism /Mount-Image /ImageFile:C:\mount\windows\Windows\System32\Recovery\winre.wim /index:1  
/MountDir:C:\mount\winre
```

Troubleshoot: If mounting operation fails, make sure that you are using the Windows 10 version of DISM that is installed with the Windows ADK and not an older version from your technician computer. Don't mount images to protected folders, such as your User\Documents folder. If DISM processes are interrupted, consider temporarily disconnecting from the network and disabling virus protection.

► Computer ► OSDisk (C:) ► mount

Name	Date modified	Type
windows	9/10/2012 1:05 PM	File folder
winre	8/27/2012 4:47 PM	File folder

► Computer ► OSDisk (C:) ► mount ► windows

Name	Date modified	Type
PerfLogs	7/26/2012 10:33 A...	File folder
Program Files	7/26/2012 1:06 PM	File folder
Program Files (x86)	7/26/2012 11:13 A...	File folder
sources	8/16/2012 11:37 A...	File folder
Users	9/10/2012 2:15 PM	File folder
Windows	9/10/2012 2:19 PM	File folder

## Modify images

### Add drivers

If you use an x64 Windows 10 image, add x64 drivers; if you use an x86 Windows 10 image, add x86 drivers.

1. Adding driver packages one by one. (.inf files) SampleDriver\driver.inf is a **sample** driver package that is specific to the computer model. Type your own specific driver path. If you have multiple driver packages, skip to the next step.

```
Dism /Add-Driver /Image:C:\mount\windows /Driver:"C:\SampleDriver\driver.inf"  
Dism /Add-Driver /Image:C:\mount\winre /Driver:"C:\SampleDriver\driver.inf"
```

2. Multiple drivers can be added on one command line if you specify a folder instead of an .inf file. To install all of the drivers in a folder and all its subfolders, use the **/recurse** option.

```
Dism /Image:C:\mount\windows /Add-Driver /Driver:c:\drivers /Recurse
```

3. Review the contents of the %WINDIR%\Inf\ (C:\mount\windows\Windows\Inf) directory in the mounted Windows image to ensure that the .inf files were installed. Drivers added to the Windows image are named Oem\*.inf. This is to ensure unique naming for new drivers added to the computer. For example, the files MyDriver1.inf and MyDriver2.inf are renamed Oem0.inf and Oem1.inf.

4. Verify your driver has been installed for both images.

```
Dism /Image:C:\mount\windows /Get-Drivers  
Dism /Image:C:\mount\winre /Get-Drivers
```

Important: If the driver contains only the installer package and doesn't have an .inf file, you may choose to install the driver in AUDIT mode by double-clicking the corresponding installer package. Some drivers may be incompatible with Sysprep tool; they will be removed after sysprep generalize even if they have been injected offline.

In this case, you need to add an extra parameter to USB-B\AnswerFiles\UnattendSysprep.xml in order to persist the drivers in the image when the image will be generalized.

```
<PersistAllDeviceInstalls>true</PersistAllDeviceInstalls>
```

This property must be added to USB-B\AnswerFiles\UnattendSysprep.xml during generalize pass in order to persist the drivers in the image. For more information about the details of this property and how to add it to an answer file, see [PersistAllDeviceInstalls](#).

#### Add languages

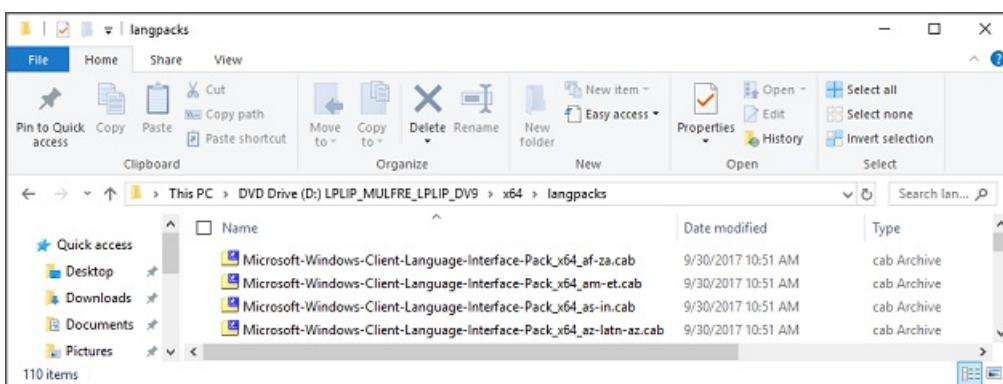
Get the Windows 10 Languages from the [Microsoft OEM site](#) or [Device Partner Center](#). The LP ISO includes both .cabs and .appx Language Interface Packs.

To add languages, see [Add languages to Windows](#).

**Important: language and LIP Versions must match other Windows component versions, for both the image and the ADK.**

If you use an x64 Windows 10 image, install x64 LIPs; if you use an x86 Windows 10 image, install x86 LIPs.

1. Copy the LIP to the USB-B\LanguagePack\x64 or USB-B\LanguagePack\x86 folder:



2. Apply the LIP to mounted image.

#### Amd64 architecture

```
DISM /Image:c:\mount\windows /Add-ProvisionedAppxPackage /PackagePath:  
E:\LIP_x64\LocalExperiencePack\eu-es\LanguageExperiencePack.eu-ES.Neutral.appx /LicensePath:  
E:\LIP_x64\LocalExperiencePack\License.xml
```

#### X86 architecture

```
DISM /Image:c:\mount\windows /Add-ProvisionedAppxPackage /PackagePath:  
E:\LIP_X86\LocalExperiencePack\eu-es\LanguageExperiencePack.eu-ES.Neutral.appx /LicensePath:  
E:\LIP_x86LocalExperiencePack\License.xml
```

## IMPORTANT

If you install an update (hotfix, general distribution release [GDR], or service pack [SP]) that contains language-dependent resources prior to installing a language pack, the language-specific changes in the update won't be applied when you add the language pack. You need to reinstall the update to apply language-specific changes. To avoid reinstalling updates, install language packs before installing updates.

### Add update packages

If you use an x64 Windows 10 image, add x64 update packages; if you use an x86 Windows 10 image, add x86 update packages.

To get update packages, download them from [Microsoft Update Catalog](#).

1. To see what packages you'll need to get, go to the [Windows 10 Release information](#) page to see which packages you should obtain from Microsoft Update Catalog.
2. Type every single update package one by one into the search box and click **Search**.

The screenshot shows the Microsoft Update Catalog website. At the top, there's a search bar with the text "KB4016871" and a "Search" button. Below the search bar, there's a "Welcome" message: "Welcome to the Microsoft Update Catalog site. We want your feedback! Visit our newsgroup or send us an email to provide us with your thoughts and suggestions. To get started using the site, enter in your search terms in the Search box above or visit our FAQ for search tips." At the bottom left, there are links for "FAQ" and "help".

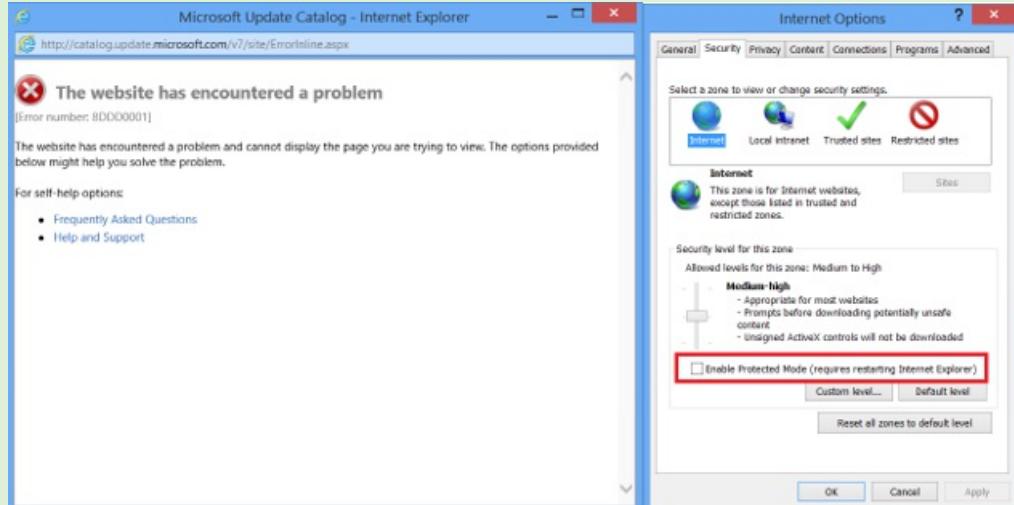
3. After each search completes, click **Download** next to the version and architecture of the package you wish to download.

The screenshot shows the Microsoft Update Catalog search results for "4048955". The search bar at the top has "4048955" in it. Below the search bar, there's a message: "Search results for "4048955"". It says "Updates: 1 - 8 of 8 (page 1 of 1)". There are "Previous" and "Next" navigation buttons. A table below lists eight update packages. The columns are: Title, Products, Classification, Last Updated, Version, Size, and Download link. All packages listed are for Windows 10 and are Security Updates.

Title	Products	Classification	Last Updated	Version	Size	Download
2017-11 Delta Update for Windows 10 Version 1709 for ARM64-based Systems (KB4048955)	Windows 10	Security Updates	11/15/2017	n/a	306.8 MB	<a href="#">Download</a>
2017-11 Delta Update for Windows 10 Version 1709 for x86-based Systems (KB4048955)	Windows 10	Security Updates	11/10/2017	n/a	167.5 MB	<a href="#">Download</a>
2017-11 Delta Update for Windows Server 2016 (1709) for x64-based Systems (KB4048955)	Windows Server 2016	Security Updates	11/10/2017	n/a	299.5 MB	<a href="#">Download</a>
2017-11 Delta Update for Windows 10 Version 1709 for x64-based Systems (KB4048955)	Windows 10	Security Updates	11/10/2017	n/a	299.5 MB	<a href="#">Download</a>
2017-11 Delta Update for Windows 10 Version 1709 for x64-based Systems (KB4048955)	Windows 10	Security Updates	11/10/2017	n/a	299.5 MB	<a href="#">Download</a>
2017-11 Delta Update for Windows 10 Version 1709 for x64-based Systems (KB4048955)	Windows 10	Security Updates	11/10/2017	n/a	299.5 MB	<a href="#">Download</a>
2017-11 Delta Update for Windows 10 Version 1709 for x64-based Systems (KB4048955)	Windows 10	Security Updates	11/10/2017	n/a	299.5 MB	<a href="#">Download</a>

## TIP

If you encounter an error that says "The website has encountered a problem" when trying to download your updates, try turning off the pop-up blocker in IE or temporarily disabling Protected Mode in IE.



- After downloading your update packages, add them to the image one by one by using the following command, substituting the filename in the command with the name of the files that you downloaded:

*Amd64 architecture*

```
Dism /Add-Package /Image:C:\mount\windows /PackagePath:"C:\windows10.0-kb4016871-x64_27dfce9dbd92670711822de2f5f5ce0151551b7d.msu"
```

*X86 architecture*

```
Dism /Add-Package /Image:C:\mount\windows /PackagePath:"C:\windows10.0-kb4016871-x86_5901409e58d1c6c9440e420d99c42b08f227356e.msu"
```

- Add updates to winre.wim (where they apply; not all updates apply to winre.wim)

*Amd64 architecture*

```
Dism /Add-Package /Image:C:\mount\winre /PackagePath:"C:\windows10.0-kb4016871-x64_27dfce9dbd92670711822de2f5f5ce0151551b7d.msu"
```

*X86 architecture*

```
Dism /Add-Package /Image:C:\mount\winre /PackagePath:"C:\windows10.0-kb4016871-x86_5901409e58d1c6c9440e420d99c42b08f227356e.msu"
```

## Add OEM specific visual customizations

- Create OEM folder under C:\mount\windows\Windows\system32\ directory.
- Create an OEM logo in .bmp format, with the size of 120px x 120 px. For more details, see the Windows Guidelines for System Builders.
- Copy the OEM logo to the folder, for example: C:\mount\windows\Windows\system32\OEM\FabrikamLogo.bmp . You'll reference this file location later in the unattend file in **OEM Information | Logo** property.

```

<OEMInformation>
  <Manufacturer>Fabrikam</Manufacturer>
  <SupportHours>08:30 - 21:30</SupportHours>
  <SupportPhone>12345678</SupportPhone>
  <SupportURL>http://www.fabrikam.com</SupportURL>
  <Model>FabrikamFabrikam</Model>
  <Logo>%windir%\system32\OEM\FabrikamLogo.bmp</Logo>
</OEMInformation>

```

- To display an OEM specific desktop background picture, the image file must be placed in %windir%\system32\OEM\Fabrikam.bmp\* directory. Verify that the path is same in answer file corresponding to oobeSystem > Microsoft-Windows-Shell-Setup > Themes > DesktopBackground property. See the below image to add desktop background in an answer file.

```

<settings pass="oobeSystem">
  <component language="neutral" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State" versionScope="nonSxS">
    publicKeyToken="31bf3856ad364e35" processorArchitecture="amd64" name="Microsoft-Windows-Shell-Setup">
      <Themes>
        <DesktopBackground>%windir%\system32\OEM\Fabrikam.bmp</DesktopBackground>
        <ThemeName>OEM Theme</ThemeName>
        <DefaultThemesOff>false</DefaultThemesOff>
      </Themes>

```

#### Modify Start layout

The Start tile layout in Windows 10 provides OEMs the ability to append tiles to the default Start layout to include Web links, secondary tiles, Windows desktop applications, and universal Windows apps. OEMs can use this layout to make it applicable to multiple regions or markets without duplicating a lot of the work. In addition, OEMs can add up to three default apps to the frequently used apps section in the system area, which delivers system-driven lists to the user including important or frequently accessed system locations and recently installed apps.

- Create LayoutModification.xml.

Note: It is recommended to start with the sample on **USB-B\StartLayout\LayoutModification.xml** as it conforms to the samples in this guide (Example Only).

The Sample LayoutModification.xml shows two groups called "Fabrikam Group 1" and "Fabrikam Group 2", which contain tiles that will be applied if the device country/region matches what's specified in Region (in this case, the regions are Germany and United States). Each group contains three tiles and the various elements you need to use depending on the tile that you want to pin to Start.

Keep the following in mind when creating your LayoutModification.xml file:

- If you are pinning a Windows desktop application using the **start:DesktopApplicationTile** tag and you don't know the application's application user model ID, you need to create a .lnk file in a legacy Start Menu directory before first boot.
- If you use the **start:DesktopApplicationTile** tag to pin a legacy .url shortcut to Start, you must create a .url file and add this file to a legacy Start Menu directory before first boot.

For the above scenarios, you can use the following directories to put the .url or .lnk files:

- %APPDATA%\Microsoft\Windows\Start Menu\Programs\
- %ALLUSERSPROFILE%\Microsoft\Windows\Start Menu\Programs\

- Save the LayoutModification.xml file.
- Add your LayoutModification.xml file to the Windows image. You'll need to put the file in the following specific location before first boot. If the file exists, you should replace the LayoutModification.XML that is already included in the image.

```

Copy E:\StartLayout\LayoutModification.xml
c:\mount\windows\users\default\AppData\Local\Microsoft\Windows\Shell\

```

Where E: is the drive letter of USB-B.

4. If you pinned tiles that require .url or .lnk files, add the files to the following legacy Start Menu directories:

- %APPDATA%\Microsoft\Windows\Start Menu\Programs\
- %ALLUSERSPROFILE%\Microsoft\Windows\Start Menu\Programs\

```
Copy E:\StartLayout\Bing.url "C:\mount\windows\ProgramData\Microsoft\Windows\Start Menu\Programs\"  
Copy E:\StartLayout\Paint.lnk "C:\mount\windows\ProgramData\Microsoft\Windows\Start Menu\Programs\"  
Copy E:\StartLayout\Bing.url "C:\mount\windows\users\All Users\Microsoft\Windows\Start Menu\Programs\"  
Copy E:\StartLayout\Paint.lnk "C:\Mount\Windows\Users\All Users\Microsoft\Windows\Start Menu\Programs\"
```

Note: If you don't create a LayoutModification.xml file and you continue to use the Start Unattend settings, the OS will use the Unattend answer file and take the first 12 SquareTiles or DesktopOrSquareTiles settings specified in the Unattend file. The system then places these tiles automatically within the newly-created groups at the end of Start. The first six tiles are placed in the first OEM group, and the second set of six tiles are placed in the second OEM group. If OEMName is specified in the Unattend file, the value for this element is used to name the OEM groups that will be created.

#### **Copy the answer file**

You may want to make additional customizations through an unattend file. The sample unattend file on USB-B contains additional common customizations.

```
Copy /y E:\AnswerFiles\Unattend.xml C:\Mount\Windows\Panther
```

Where E:\ is USB-B.

#### **Optimize WinRE**

1. Increase scratchspace size.

```
Dism /image:c:\mount\winre /set-scratchspace:512
```

2. Cleanup unused files and reduce size of winre.wim

```
Dism /image:"c:\mount\winre" /Cleanup-Image /StartComponentCleanup /Resetbase
```

#### **Unmount images**

1. Close all applications that might access files from the image

2. Commit the changes and unmount the Windows RE image:

```
Dism /Unmount-Image /MountDir:"C:\mount\winre" /Commit
```

where C is the drive letter of the drive that contains the image.

This process can take a few minutes.

3. Make a backup copy of the updated Windows RE image.

Troubleshoot: If you cannot see winre.wim under the specified directory, use the following command to set the file visible:

```
attrib -h -a -s C:\mount\windows\Windows\System32\Recovery\winre.wim
Dism /export-image /sourceimagefile:c:\mount\windows\Windows\System32\Recovery\winre.wim /sourceindex:1
/DestinationImageFile:e:\images\winre_bak.wim
Del c:\mount\windows\Windows\System32\Recovery\winre.wim
Copy e:\images\winre_bak.wim c:\mount\windows\Windows\System32\Recovery\winre.wim
```

When prompted, specify **F** for file

#### 4. Check the new size of the Windows RE image.

```
Dir "C:\mount\windows\Windows\System32\Recovery\winre.wim"
```

Use the following partition layout size guidance to determine the size of your recovery partition in `CreatePartitions-<firmware>.txt` files. The amount of free space left is after you copy `winre.wim` to the hidden partition.

Please reference [Disk Partition rules](#) for more information.

- If the partition is less than 500 MB, it must have at least 50 MB of free space.
- If the partition is 500 MB or larger, it must have at least 320 MB of free space.
- If the partition is larger than 1 GB, we recommend that it should have at least 1 GB free.

```
rem == Windows RE tools partition =====
create partition primary size=500
```

Optional: This section assumes you'd rather keep `winre.wim` inside of `install.wim` to keep your languages and drivers in sync. If you'd like to save a bit of time on the factory floor, and if you're OK managing these images separately, you may prefer to pull `winre.wim` from the image and apply it separately.

#### 5. Commit the changes and unmount the Windows image:

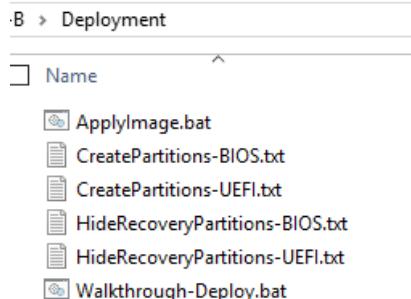
```
Dism /Unmount-Image /MountDir:"C:\mount\windows" /Commit
```

Where C is the drive letter of the drive that contains the image.

This process may take several minutes.

## Deploy the image to new computers (Windows installation)

1. On the technician computer, locate the following files in USB-B/Deployment. Please see [Creating My USB-B](#) to create and place the files in correct paths.



2. Boot the reference computer and connect USB-A.

3. After WinPE starts, connect USB-B.
4. Type `diskpart` to start Diskpart. Then type `list volume` to identify volume label of Windows Installation volume labelled "Windows" (For example: E:). Finally, type `exit` to quit Diskpart.
5. Apply the model-specific image:

```
E:\Deployment\ApplyImage.bat E:\Images\ModelSpecificImage.wim
```

Note: There are several pauses in the script. You will be prompted Y/N for the Apply operation if this is a Compact OS deployment.

**NOTE**

Only use Compact OS on flash-drive-based devices (solid-state drives), because Compact OS performance depends on the storage device capabilities. Compact OS is NOT recommended on rotational devices. For more information, see [Compact OS](#).

6. Remove USB-A and USB-B, and then type:

```
Exit
```

## Update images manually by using AUDIT MODE (online servicing)

Important: Connecting the computer to internet is not recommended during manufacturing stages. We don't recommend getting updates from Windows Update in audit mode, as it will likely generate errors when you generalize + sysprep the machine from audit mode.

### Add Office apps to your image

To add the Office apps to an image, use DISM with the `/Add-ProvisionedAppxPackage` option. This option also requires the following information for each app you add:

- `/PackagePath` : This is only used to specify the path to the .appxbundle file for the shared code package.
- `/OptionalPackagePath` : This is used to specify the path to the .appxbundle file for an individual app, such as Word or Excel.
- `/LicensePath` : This is used to specify the path to the \_License1.xml file for an individual app. This is needed for both the shared package and each of the optional app packages.

1. Extract the Office 16.5 OPK to C:\temp\lab\apps\Office Apps\Shared.Preinstallkit.
2. Use DISM to add all the Office apps to an offline image. The following example assumes the appxbundle and license xml files are in subdirectories on *USB-B* (D:). The example also excludes the /region switch because we want Office to appear in both the All Apps list, and as a Start Menu tile.

```
DISM /online /Add-ProvisionedAppxPackage /PackagePath="C:\temp\lab\apps\Office  
Apps\shared.PreinstallKit\shared.appxbundle" /OptionalPackagePath="C:\temp\lab\apps\Office  
Apps\excel.PreinstallKit\excel.appxbundle" /OptionalPackagePath="C:\temp\lab\apps\Office  
Apps\powerpoint.PreinstallKit\powerpoint.appxbundle" /OptionalPackagePath="C:\temp\lab\apps\Office  
Apps\word.PreinstallKit\word.appxbundle" /OptionalPackagePath="C:\temp\lab\apps\Office  
Apps\outlook.PreinstallKit\outlook.appxbundle" /OptionalPackagePath="C:\temp\lab\apps\Office  
Apps\publisher.PreinstallKit\publisher.appxbundle" /OptionalPackagePath="C:\temp\lab\apps\Office  
Apps\access.PreinstallKit\access.appxbundle" /LicensePath="C:\temp\lab\apps\Office  
Apps\shared.PreinstallKit\shared_license1.xml" /LicensePath="C:\temp\lab\apps\Office  
Apps\excel.PreinstallKit\excel_license1.xml" /LicensePath="C:\temp\lab\apps\Office  
Apps\powerpoint.PreinstallKit\powerpoint_license1.xml" /LicensePath="C:\temp\lab\apps\Office  
Apps\word.PreinstallKit\word_license1.xml" /LicensePath="C:\temp\lab\apps\Office  
Apps\outlook.PreinstallKit\outlook_license1.xml" /LicensePath="C:\temp\lab\apps\Office  
Apps\publisher.PreinstallKit\publisher_license1.xml" /LicensePath="C:\temp\lab\apps\Office  
Apps\access.PreinstallKit\access_license1.xml"
```

#### TIP

You need to specify both an appxbundle and a license package for the shared package, as well as for each individual app that you want to install.

3. Verify Office was installed:

```
Dism /Image:"C:\mount\windows" /Get-ProvisionedAppxPackages
```

where C is the drive letter of the drive that contains the image.

Review the resulting list of packages and verify that the list contains the Office Desktop Bridge apps, such as:

```
...  
Displayname : Microsoft.Office.Desktop.Access  
Version : 16000.8528.2136.0  
Architechture : neutral  
ResourceID : ~  
PackageName : Microsoft.Office.Desktop.Access_16000.8528.2136.0_neutral_~_8wekyb3d8bbwe  
Regions : None  
  
Displayname : Microsoft.Office.Desktop.Excel  
Version : 16000.8528.2136.0  
Architechture : neutral  
ResourceID : ~  
PackageName : Microsoft.Office.Desktop.Excel_16000.8528.2136.0_neutral_~_8wekyb3d8bbwe  
Regions : None  
...
```

To have the apps appear on the Start screen, follow the steps in the next section: Configuring Start tiles and taskbar pins.

To complete the Office install, you'll need to unmount the image and commit your changes, which we'll do this after we've completed all customizations at the end of this lab.

#### Pin Office tiles to the Start menu

We'll pin the Office tiles to the Start menu so Windows won't remove the Office files during OOBE.

1. Open a command prompt and type:

```
notepad C:\Users\Default\AppData\Local\Microsoft\Windows\Shell\LayoutModification.xml.
```

2. Add the following to layoutmodification to pin the Office apps to your Start Menu:

```
<AppendOfficeSuite/>
<AppendOfficeSuiteChoice Choice="DesktopBridgeSubscription"/>
```

3. Close and save layoutmodification.xml.

Note: for recovery purposes the layoutmodification.xml will need to be copied during recovery.

4. Open a command prompt and type:

```
copy C:\Users\Default\AppData\Local\Microsoft\Windows\Shell\LayoutModification.xml
c:\Recovery\AutoApply
```

Once the machine is booted to desktop after going through OOBE, the Start menu will have the Office tiles added to the Start Menu.

## Prepare recovery tools

[Push-button reset](#) can help users recover the OS while preserving their existing data and customizations without requiring them to back-up their data in advance.

Any languages, Universal Windows apps and Universal Windows drivers that are included in your image are automatically restored during push-button recovery operations. Make sure other customizations, like desktop apps and Start Menu customizations get restored, too.

In Windows 10, version 1809, you can use [auto-apply folders](#) to restore common Windows settings such as the Start Menu, taskbar layout, and OOBE customizations. For previous Windows versions, or to perform other actions after a push-button reset, use [extensibility scripts](#) instead. Sample extensibility scripts are included in the USB-B sample files.

### Copy the ScanState tool to your USB key

The ScanState tool is included in the USB-B sample files you downloaded earlier.

You'll use ScanState tool to capture your classic Windows apps and settings so they can be restored later during a push-button reset recovery.

You can also get a copy using the tools in the Windows ADK:

#### On your technician PC:

1. Start the **Deployment and Imaging Tools Environment** as administrator.
2. Run the CopyDandI.cmd script to copy the files to your USB key:

x64:

```
CopyDandI.cmd amd64 E:\ScanState_amd64
```

Where E: is the letter of USB-B drive.

If you're using an x86 Windows 10 image, make x86 Scanstate directory:

```
CopyDandI.cmd x86 e:\ScanState_x86
```

Where E: is the letter of USB-B drive.

## Create a recovery package

### On your reference PC:

1. In Windows 10, version 1809, create auto-apply folders to restore common Windows settings such as the Start Menu, taskbar layout, and OOBE customizations.

Create a folder in your Windows image called `C:\Recovery\AutoApply`

```
MkDir C:\Recovery\AutoApply
```

2. Copy configuration files and the related asset files

- Unattend.xml:

```
copy Copy the unattend.xml file you want for recovery to `C:\Recovery\AutoApply\` and any asset files to `C:\Recovery\AutoApply\CustomizationFiles`
```

- Start menu:

```
Copy E:\StartLayout\LayoutModification.xml C:\Recovery\AutoApply\CustomizationFiles
```

- Taskbar pins:

```
copy
```

- Copy your TaskbarLayoutModification.xml to `C:\Recovery\AutoApply\` and any asset files to `C:\Recovery\AutoApply\CustomizationFiles`
- Copy `%windir%\System32\OOBE\info` and all its contents to `C:\Recovery\AutoApply\OOBE`

3. Use ScanState to capture installed customizations into a provisioning package, and then save it to `c:\Recovery\customizations`.

**Important:** For push-button reset to recover your apps and customizations, you must store the packages file as a .ppkg file in the `C:\Recovery\Customizations` folder.

Run ScanState to gather app and customizations

x64:

```
mkdir c:\recovery\customizations
E:\ScanState_amd64\scanstate.exe /apps /ppkg C:\Recovery\Customizations\apps.ppkg
/i:c:\recovery\oem\regrecover.xml config:E:\scanstate_amd64\Config_AppsAndSettings.xml /o /c /v:13
/l:C:\ScanState.log
```

Where E: is the drive letter of USB-B

x86:

```
E:\ScanState_x86\scanstate.exe /apps /ppkg C:\Recovery\Customizations\apps.ppkg  
/i:c:\recovery\oem\regrecover.xml /config:e:\scanstate_x86\Config_AppsAndSettings.xml /o /c /v:13  
/l:C:\ScanState.log
```

Where E: is the drive letter of USB-B

- When ScanState completes successfully, delete scanstate.log and miglog.xml files:

```
del c:\scanstate.log  
del c:\miglog.xml
```

### Copy a backup of WinRE

During a PC deployment, winre gets moved. Before you capture a final image, copy the backup of winre.wim back into the Windows image.

```
Copy e:\images\winre_bak.wim c:\windows\system32\recovery\winre.wim
```

## Reseal the image

- Delete the installation folders and files you have created for the preloaded applications. Extra folders may increase the size of the .wim when the Windows image gets captured.
- If Sysprep is open, close it and open an elevated command prompt.
- Copy unattend.xml to the recovery folder to enable recovery of unattend settings during Push Button Reset.

```
copy USB-B\answerfiles\unattendsysprep.xml c:\Recovery\OEM\unattend.xml
```

- Generalize the image by using the answer file which reflects the changes made in the section [Update images manually by using AUDIT MODE \(online servicing\)](#).

These changes include Microsoft Office tile component pinned to the Start screen.

```
Cmd /c C:\Windows\System32\Sysprep\sysprep /unattend:c:\Recovery\OEM\Unattend.xml /generalize /oobe  
/shutdown
```

- Boot reference computer and connect USB-A.
- After WinPE has been booted connect USB-B.
- Type `diskpart` to start Diskpart. Then type `list volume` to identify volume label of Windows Installation volume labelled "Windows" (For example: E:). Finally, type `exit` to quit Diskpart.
- Start cleanup of the image.

Important: By default, non-major updates (such as ZDPs, or LCUs) are not restored. To ensure that updates preinstalled during manufacturing are not discarded after recovery, they should be marked as permanent by using the `/Cleanup-Image` command in DISM with the `/StartComponentCleanup` option.

```
MD e:\scratchdir  
dism /Cleanup-Image /Image:e:\ /StartComponentCleanup /scratchdir:e:\scratchdir
```

- Capture the image of the windows partition. This process takes several minutes.

```
dism /Capture-Image /CaptureDir:E:\ /ImageFile:F:\Images\ModelSpecificImage.wim  
/Name:"myWinImageWithMSIUpdated" /scratchdir:e:\scratchdir
```

Where E: is the volume label of Windows and F is the volume label of USB-B.

This will overwrite the image created in the section [Deploy the image to new computers](#).

## Deploy the image

Use the deployment script to layout the partitions on the device and apply the image. The applyimage.bat in USB-B\deployment folder will partition the device based on device mode.

**Important: The Recovery partition must be the partition after the Windows partition to ensure winre.wim can be kept up-to-date during the life of the device.**

Run the following command to deploy your image to the reference PC:

```
E:\Deployment\applyimage.bat E:\Images\modelspecificimage.wim
```

Note: There are several pauses in the script. You will be prompted Y/N for the Apply operation if this is a Compact OS deployment.

Note: Only use Compact OS on high end storage devices because Compact OS performance depends on the storage device capabilities. Compact OS is NOT recommended on rotational devices or storage greater than 32 GB. For more information, see [Compact OS](#).

Remove USB-A and USB-B and type *exit* to reboot your computer with Windows 10.

## Finalize deployment

- Upon deploying your model specific image to destination computers, boot the computer with master image for the first time in AUDIT mode

Important: In order to minimize the first boot time, (Boot > Specialize > OOBE > Start screen) specialize pass must be completed in the factory. Specialize pass will configure hardware specific information which Windows will run on.

For more information about the first boot time requirements, see [Windows Policy for System Builders](#).

- Please note that at the end of the section [Update images manually by using AUDIT MODE \(online servicing\)](#), the system was sealed with OOBE mode. Please proceed with Audit. If the system boots in OOBE, press Ctrl+Shift+F3 in order to pass OOBE and boot in audit mode.
- If you want to apply additional steps, such as executing OEM diagnostics tests and so on, apply them here.
- Finally, run the Sysprep tool (C:\Windows\System32\Sysprep\sysprep.exe) and seal the system back to **OOBE** and **Shutdown** but *without Generalize*.
- The system is ready to ship.

Important: If you are manufacturing a small amount of devices without using an image managing tool such as disk duplicators or Windows Deployment Service, you can choose to use the following practice:

- You can manufacture those devices by first booting in WinPE - inserting USB-A.
- Then insert USB-B where final manufacturing image is contained.
- Run the applyimage.bat script to apply the image.
- After you applied the image, follow the steps in this Finalize deployment section.

- e. Now the device is ready to be shipped with your final manufacturing image and PBR feature implemented.
- f. Finally, replicate the same procedure with the other devices.

## Appendix

### Differences between 64-bit and 32-bit deployment

It is recommended to consider 64-bit deployment versus 32-bit deployment disk footprint according to the storage of the device you are manufacturing.

The overall deployment flow mentioned in this guide doesn't differ between 64-bit and 32-bit deployment. Only some of the resource versions and the way those resources are created differs. The following table covers the x64/x86 distinctions.

DISTINCTION	DESCRIPTION	RELATED SECTION
Windows installed on technician PC	When Windows ADK gets installed on a technician PC the deployment tools in the ADK would be installed according to the architecture of the Windows on technician PC. In short if ADK is installed on Windows x64, the tools would be installed 64-bit version, or vice-versa.	<a href="#">Prepare your lab environment</a>
Creating WinPE folder structure	WinPE differs between x64 and x86 architecture, so you have to use different commands to create a different WinPE folder for each architecture.	<a href="#">Create WinPE bootable USB</a>
Drivers	Driver versions differ between different architectures. If you are manufacturing a 64-bit Windows image, please use x64 drivers, and vice-versa for 32-bit Windows.	<a href="#">Add drivers</a>
Update Packages for Windows Image	Update package versions differ between different architectures. If you are manufacturing a 64-bit Windows image please use x64 update packages, and vice-versa for 32-bit Windows.	<a href="#">Add update packages</a>
Language Interface Packs	IF you will be using x64 Windows 10 image, install x64 LIPs or if you will be using x86 Windows 10 image install x86 LIPs.	<a href="#">Prepare the system for recovery with Push-button reset</a>

### What you will need and where to get it

Before starting the deployment procedure OEM requires to download certain kits which will be used throughout the guide, such as Microsoft Office, update packages, language interface packs. Below is the complete list of resources/kits an OEM requires to download and where they download them.

RESOURCE/KIT	AVAILABLE AT	RELATED SECTION
Windows 10 ADK	<a href="#">Download the Windows ADK</a>	<a href="#">Create WinPE bootable USB</a>

RESOURCE/KIT	AVAILABLE AT	RELATED SECTION
Windows 10 x64/x86 DVD Media (desired language)	Obtain Windows 10 media which you will be customizing from Microsoft Authorized Distributor	<a href="#">Install Windows with basic customizations</a>
Windows 10 Default Product Keys	Default Product Keys are located at <a href="#">Device Partner Center</a> listed under <b>Default product keys</b> tab	<a href="#">Customize the answer file</a>
Language packs	Language packs are located at <a href="#">Device Partner Center</a> listed under <b>LIPs</b> tab	<a href="#">Prepare the system for recovery with Push Button Reset</a>
Update Packages	Obtain update packages by downloading from <a href="#">Microsoft Update Catalog</a> . The detailed procedure downloading update packages is mentioned in the related section.	<a href="#">Add language interface packs</a>
Microsoft Office v16.5	Obtain Microsoft Office v16.5 by downloading from Device Partner Center	<a href="#">Microsoft Office v16.5 OPK</a>

## References

[Windows Guidelines for System Builders](#)

[Windows Policy for System Builders](#)

# Manufacturing Windows Engineering Guide (WEG)

1/18/2019 • 24 minutes to read

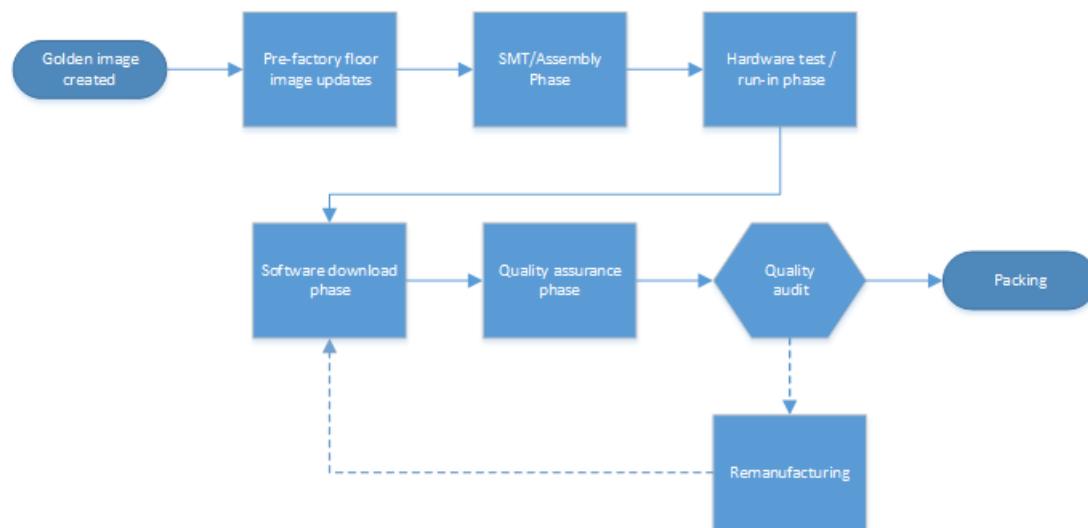
The Manufacturing WEG provides original equipment manufacturer (OEM) and ODM partners with a roadmap of the ideal manufacturing process for Windows 10 devices, with guidance for potential pitfalls and opportunities to streamline the process.

## Manufacturing overview

Many decisions that affect manufacturability are made early in the engineering effort of a new device, so careful consideration should be made to ensure the lowest overhead production process is selected. Every extra minute spent on the manufacturing floor equates to extra cost for the final product. The Manufacturing WEG is intended to provide OEM and ODM partners with a roadmap of the ideal manufacturing process that brings together software and hardware on the factory floor. This WEG also provides opportunities to streamline the process and guidance for how to plan for and avoid common problems. Our manufacturing and deployment recommendations are meant to help you:

- Optimize the image disk footprint on desktops
- Enable Windows deployment on small capacity disks on desktops
- Shorten image deployment time
- Simplify the imaging process
- Simplify OEM Activation (OA3) injection/reporting process on desktops. Mobile devices do not require activation.
- Test and calibrate the device on the assembly line
- Support other key scenarios to build great devices

For this document, a generic version of the desktop manufacturing process would look like this:



The manufacturing process for mobile devices would look like this:



The Manufacturing WEG is not intended to communicate the [Windows Minimum Hardware Requirements](#) or OEM Policy Document (OPD). The WHCR and OPD documents take precedence over any information in the Manufacturing WEG. You must comply with WHCR and OPD.

## Overall considerations

### Manufacturing path

There are two general manufacturing paths you can take depending on your business—Build to Stock (BTS) and Build to Order (BTO). As you review the guidelines in this document, consider the manufacturing path for the device in order to prioritize investments in each phase and save as much time as possible for your customized process.

For a walkthrough using desktop devices, see our [manufacturing end-to-end lab](#).

For mobile devices, you must use the BTS manufacturing path.

### Build to order (BTO)

BTO devices start with a basic image and then receive the majority of their customizations during the manufacturing process.

The primary advantage is the flexible software bill of materials, which allows for late-breaking changes. The drawbacks include a more complex image creation and manufacturing process, extra time on the factory floor, and growing image sizes.

### Build to stock (BTS)

BTS devices have images that are customized almost entirely in the lab. BTS processes are simpler to plan and produce, are faster on the factory floor, have higher quality control, and have a controlled disk size. BTS devices still need to allow for late breaking changes on the factory floor. For desktop editions of Windows 10, many of these changes can be done using offline servicing.

### Push-button reset

The push-button reset tools no longer require a separate full-system recovery image on a separate partition. This can save several gigabytes of space. When users need to refresh or reset the device, they'll be able to keep their installed Windows updates, rather than downloading and installing them all again. They'll also keep all of the customizations that you've provided.

The new partition layout resembles the traditional partition layouts from Windows 8.1, except the Windows RE partition is now moved to the end of the drive, and there is no longer a need for a separate full-system recovery partition.

### Disk 0 default partition layout (UEFI-based PCs)



For more information, see [Push-button reset](#)

Push-button reset is not supported on mobile devices. Instead, you should do a factory reset.

## **Compact OS**

You can now run the entire operating system, including your preloaded Windows desktop applications, using compressed files, by using the Compact OS and single-instancing features. These features replace the WIM Boot feature from Windows 8.1 Update 1, and can help maintain a smaller disk footprint over time.

Although the Compact OS is supported for all devices, we recommend using Compact OS only on devices with solid-state drives, because of the slower performance of rotational drives.

For more information, see [Compact OS, single-instancing, and image optimization](#).

Compact OS is not supported on mobile devices.

## **Provisioning packages**

To save time while building images, you can now capture and apply desktop Windows applications during image deployment by using provisioning packages. This saves the time-consuming steps of generalizing and recapturing the entire image, and allows you to quickly deploy BTO devices.

## **Language packs**

Instead of adding full language packs, save space by adding the resources you need for the desktop device by choosing individual packages for display strings, handwriting, speech, and text-to-speech. Later, if your user needs additional language capabilities, Windows can download the packages as needed.

Language and regional SKU decisions can greatly impact the disk footprint and complexity of the image creation system. Care should be taken to limit the amount and types of language packs included with each image.

Mobile devices use a worldwide image so all languages are included in every image.

## **Driver co-installers**

Drivers are generally a very small portion of the disk footprint, however, the co-installers or desktop device apps that accompany the drivers can add hundreds of megabytes. Carefully consider if the device(s) require the accompanying Classic Windows application to be fully functional.

## **Hardware components**

Hardware decisions can also affect the manufacturing process. Besides the challenges to the physical assembly of the hardware, the inclusion or exclusion of certain devices can make the factory process more difficult. For example, if touch screens and sensors are included they must be calibrated on each device. If you exclude devices such as ethernet ports, you can't use PXE boot, which can mean extra costs.

## **Antimalware apps**

**Recommendation:** Configure your devices to avoid full scan of the disk during first sign-in. Please work with your Antimalware vendor to determine best practices for limiting this scan.

We have seen several instances where antimalware tools are doing a full disk scan during the user's first sign-in. The scanning competes with critical tasks occurring during the first sign-in process, resulting in very slow first sign-in, a degraded Start experience and slow system performance.

For Windows Defender, this can be configured by adding unique identifiers to your images. To learn more, see [Configure a Trusted Image Identifier for Windows Defender](#).

## **Pre-factory floor image updates**

Golden images are generally handed off to the ODM from the OEM before production begins. These images almost always require some updating. When you update the golden image, you won't have to perform the updates on each device. This leads to less time on the factory floor for each device and increases quality.

Updates to the image can include drivers, Windows Updates, software, OEM customizations, and app packages (.appx).

## Considerations

If you update images using offline servicing, you'll need to periodically maintain the images. The time saved on the factory floor should make it worthwhile.

## Goals

Reduce time spent per unit on the factory floor and decrease the amounts of errata on production devices.

## Implementation

On a BTO system, some optional drivers and some optional apps may need to be applied at the software download station to accommodate the device. These modifications should be minimized in order to decrease the likelihood of error and to improve manufacturing time.

### Image creation

The overall golden desktop image creation process in OEM image labs is similar to the existing process.



To avoid compatibility issues, use the new version of Windows PE when working on the reference device in the image creation lab.

### Region-specific policy for Skype removal on desktop

Windows-provided apps are included in all Windows images by default. These apps cannot be modified except where explicitly stated in the Windows OEM Policy Document (OPD).

If you are required to remove the inbox Skype app due to policy requirements, you can use the DISM.exe tool or the DISM Windows PowerShell cmdlets to remove the app. For more information about this policy requirement, see the most recent OEM Policy Document.

To remove Skype online in audit mode from Windows PowerShell:

```
get-provisionedappxpackage -online | where-object {$_._displayname -eq "Microsoft.SkypeApp"} | Remove-ProvisionedAppxPackage -online
```

To remove Skype offline with Windows PowerShell:

```
get-provisionedappxpackage -path c:\mount | where-object {$_._displayname -eq "Microsoft.SkypeApp"} | Remove-ProvisionedAppxPackage
```

To remove Skype offline using Dism.exe:

1. Get the full package name:

```
Dism.exe /image:<Windows_volume> /get-provisionedappxpackages
```

2. Remove the package, using the from the Microsoft.SkypeApp listing:

```
Dism.exe /image:<Windows_volume> /remove-provisionedappxpackage /PackageName:<PackageName>
```

We recommend using the Windows 10 version of Windows Preinstallation Environment (WinPE).

Note: If you use the Windows 8 version of WinPE, then after any servicing operation, you must update the timestamps of the files; otherwise you may not be able to activate the image using the OEM Activation 3.0 licensing method. In addition, the in-box licensing diagnostic tool, licensingdiag.exe, will report that the image has been tampered with.

To resolve this problem, after any servicing operation run from the Windows 8 version of WinPE, the OEM must run:

```
dir %windir%\System32\catroot\{F750E6C3-38EE-11D1-85E5-00C04FC295EE}
```

## Language pack updates

After installing a new language, you must reinstall any APPX bundles and inbox Windows apps to support the new languages. Otherwise, the APPX bundles won't include support for the new languages.

## Apps in Audit mode

After installing a new language, you must reinstall any APPX bundles and inbox Windows apps to support the new languages. Otherwise, the APPX bundles won't include support for the new languages.

To disable the app readiness service offline:

1. Create a .reg file where HKLM\Software\Microsoft\Windows\CurrentVersion\AppReadiness DisableInAuditMode is set to a value of 1.
2. Mount the Windows image. For example:

```
Dism /Mount-Image /ImageFile:"C:\Images\ModelSpecificImage.wim" /Name:"Fabrikam"
/MountDir:"C:\mount\windows" /Optimize
```

3. Load the registry hive. For example:

```
reg load hklm\LoadedHive C:\mount\Windows\System32\config\SYSTEM
```

4. Add the registry value. For example, using a .reg file from your USB stick:

```
regedit /s e:\registry\regFile.reg
```

5. Unload the hive.

```
reg unload hklm\LoadedHive
```

6. Unmount the Windows image, committing changes. For example:

```
Dism /Unmount-Image /MountDir:"C:\mount\windows" /Commit
```

It is recommended that you disable the service offline before entering audit mode. But you can also disable it online in audit mode. You must generalize the image after you disable the service. To disable the app readiness service online:

1. In audit mode, start `regedit`.
2. Navigate to HKLM\Software\Microsoft\Windows\CurrentVersion\AppReadiness DisableInAuditMode.
3. Set the value of the key to 1.

Run Sysprep generalize before continuing.

## SMT / Assembly phase

Devices need to be calibrated for the best customer experience and to pass the Windows Hardware Lab Kit tests.

### Implementation

- Calibrations
  - Sensors
  - Touchpad
  - Touchscreen
  - RF
  - Camera

Set the time to UTC and implement ACPI changes detailed below.

## Hardware test and run-in phase

Test on a full version of Windows. This lets you test final hardware/software interactions as the user will see them.

### Considerations

Using the full version of Windows that you will be shipping with allows testing and validation in the exact same environment that the end user will be seeing. Windows PE is not a supported operating system for test – it is designed to be used only as a deployment vehicle.

### Goals

Deliver the highest quality product possible, while keeping manufacturing times at the absolute minimum.

### Implementation

The installation of the test OS can be handled a variety of ways. Since the test OS has less churn than the shipping OS the image can be laid down on the disk at any time, which can reduce the amount of time spent applying the image on the factory floor. Options include having the image pre-flashed from the IHV, or using disk duplication on site.

**Important:** Disable TPM (Trusted Platform Module) auto-provisioning when booting into a test OS to ensure both good performance and to make sure the user's OS has ownership of the module. To do this in Windows, you need to set the following registry keys:

```
[HKLM\System\CurrentControlSet\Services\Tpm\WMI\NoAutoProvision] (REG_DWORD) to 1  
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TPM\WMI]  
"NoAutoProvision"=dword:00000001
```

## Software download phase

### Considerations

Care should be taken to minimize the amount of time spent on this phase. While some long durations are unavoidable (such as BTO customizations), we encourage our partners to calculate the costs vs. benefits of streamlining as much as possible.

### Goals

Create an efficient and resilient imaging system with a minimum amount of overhead. Move as many steps as possible to the Pre-Factory Floor Image Updates.

### Implementation

1. Boot the machine to WinPE. To deploy Compact OS, you'll need to use the Windows 10 version of WinPE.

You can boot WinPE in a variety of ways:

- Using PXE
- Using a USB stick
- Preinstalling WinPE on the hard disk

2. Create the hard drive partition structure using diskpart.

```
diskpart /s F:\CreatePartitions-UEFI.txt
```

For more information, see [UEFI/GPT-based hard drive partitions](#)

3. Apply the images that you created using DISM to the Windows partitions.

```
ApplyImage F:\Images\ThinImage.wim
```

For more information, see [Capture and Apply Windows, System, and Recovery Partitions](#).

Optional: Use the same image as a recovery image on a separate USB flash drive. This USB disk no longer needs to be sourced from an authorized replicator. For more info, see [Create media to run push-button reset features](#).

4. Boot the device into audit mode.

- Make any final image modifications. This is where many BTO modifications are made.
- If APPX apps were installed before additional language packs were added, reinstall the apps so they can support the new languages. This includes inbox applications.
- Microsoft strongly advises that OEMs run DISM with the /StartComponentCleanup /resetbase flags to gain additional free disk space.
- Ensure .NET Framework apps are compiled by running the following commands

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\ngen.exe update /queue  
C:\Windows\Microsoft.NET\Framework\v4.0.30319\ngen.exe eqi
```

On 64-bit machines, do the same for 64-bit CLR:

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\ngen.exe update /queue  
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\ngen.exe eqi
```

- Create the OA3 computer build report (CBR) using OAtool.exe, inject the key into firmware, and validate the provisioning.
- If you're using Windows Preinstallation Environment (WinPE) 5.x, fix the timestamps. (This step is not necessary if you're using WinPE for Windows 10).

```
dir %windir%\System32\catroot\{F750E6C3-38EE-11D1-85E5-00C04FC295EE}
```

- Run Sysprep /oobe to prepare the system for the end user.
- Enable Secure Boot (if implemented).

# Quality assurance phase

Quality assurance should be run on a sample of machines throughout the production run to ensure all customizations have been successfully installed. This QA pass should also be used to validate the OA3 implementation.

## Considerations

Once the system has gone through OOBE for this QA pass, it will need to be reset in order to assure the end-user experience is excellent.

## Goals

Assure a good customer experience and reduce time spent rebuilding machine.

## Implementation

The device should return to the production line to be re-imaged.

# Remanufacturing

Once the device has passed through OOBE (for any reason) the device needs to be reimaged. In the factory, the device can simply be added back onto the Software Download Station after the TPM has been cleared (if equipped).

If the device needs to be serviced in the field by a 3rd party, then Push Button Reset should be used. This will guarantee the machine is returned back to the full factory configuration while still remaining simple to execute.

## Considerations

While PBR has all the functions needed for factory remanufacturing, it does not provide the ability to script the actions and does not provide actionable error codes needed for automation that would be necessary in large scale production.

## Goals

Ensure a good customer experience, while protecting the user data.

## Implementation

To clear the TPM use the following commands:

```
$Tpm = Get-WmiObject -class Win32_Tpm -namespace "root\CMV2\Security\MicrosoftTpm"  
$Tpm.SetPhysicalPresenceRequest(22)
```

To use Windows PE, you'll need a customized Windows PE image with:

- SOC-specific drivers for fptm.
- Optional components: SecureStartup, WMI, PowerShell, and .NET Framework.

# Manufacturing checklist

## Windows 10 manufacturing task timeline

You can use this checklist to plan your manufacturing tasks.

TASK	PRE-EV PHASE	EV PHASE	DV PHASE	PV PHASE
Prerequisite manufacturing work:	Pre-EV	EV	DV	PV

TASK	PRE-EV PHASE	EV PHASE	DV PHASE	PV PHASE
ODM chosen?	✓	-	-	-
OEM access to Windows 10?	✓	-	-	-
ODM access to Windows 10?	✓	-	-	-
OEM access to Manufacturing WEG?	✓	-	-	-
ODM access to Manufacturing WEG?	✓	-	-	-
ODM points of contact identified?	✓	-	-	-
OEM points of contact identified?	✓	-	-	-
ODM kick-off?	✓	-	-	-
OEM kick-off?	✓	-	-	-
Regular manufacturing call?	-	✓	-	-
Deployment:	Pre-EV	EV	DV	PV
OEM understanding of deployment concepts	✓	-	-	-
ODM understanding of deployment concepts	✓	-	-	-
Using Windows 10 DISM	-	✓	-	-
Using Windows 10 Windows PE	-	✓	-	-
Extended attributes applied via DISM	-	✓	-	-
WinSxS check being run? /AnalyzeComponentStore	-	✓	-	-

Task	Pre-EV Phase	EV Phase	DV Phase	PV Phase
Image is cleaned up? (DISM /Cleanup-Image /StartComponentCleanup /ResetBase)	-	✓	-	-
Push-button reset	Pre-EV	EV	DV	PV
OEM understanding of push-button reset concepts	-	✓	-	-
ODM understanding of push-button reset concepts	-	✓	-	-
Recommended partition layout to use for Windows RE and push-button reset?	-	-	✓	-
If a non-standard partition layout is used, is bare-metal recovery configured?	-	-	✓	-
Recovery image ACL settings correct?	-	-	✓	-
Refresh/reset time is within guidelines?	-	-	-	✓
Windows RE:	Pre-EV	EV	DV	PV
OEM understanding of Windows RE concepts	-	✓	-	-
ODM understanding of Windows RE concepts	-	✓	-	-
Windows RE is enabled	-	-	✓	-
Windows RE location is correct	-	-	✓	-
BCD GUID for Windows RE matches Windows RE GUID entry in Reagent.xml	-	-	✓	-
Image index is correct	-	-	✓	-

Task	Pre-EV Phase	EV Phase	DV Phase	PV Phase
Manufacturing:	Pre-EV	EV	DV	PV
Windows RE partition size (MB) and position on selected disk	-	✓	-	-
Test partition uses full Windows	-	-	✓	-
Image deployment via [NIC] or [duplication]	-	✓	-	-
OPM key provisioning plan done?	-	✓	-	-
Language packs per SKU	-	-	✓	-
Secure boot:	Pre-EV	EV	DV	PV
OEM understanding of security concepts	✓	-	-	-
ODM understanding of security concepts	✓	-	-	-
Secure boot process tested with preproduction signing?	-	-	✓	-
Watermark off?	-	-	-	✓
Drivers signed from IHV/ISV?	-	-	-	✓
Drivers signed by Microsoft?	-	-	-	✓
Re-manufacturing process complete [factory]	-	-	✓	-
Re-manufacturing process complete [field service]	-	-	✓	-
Secure boot and debug policy plans vetted by FT	-	✓	-	-
OA 3.0:	Pre-EV	EV	DV	PV

Task	Pre-EV Phase	EV Phase	DV Phase	PV Phase
Using updated OA3tool.exe	-	-	✓	-
Validate image/key offline	-	-	✓	-
Injection done in [Windows PE] [Windows] Refer to the OEM Activation 3.0 section.	-	-	✓	-

## Appendix

### Small disk footprint optimization

The basic disk footprint of Windows 10 x86 with Office and 2GB of RAM will contain:

Windows (w/Office), Page file, hiberfile, swapfile, and two language packs	11.7GB
WinRE	500MB
System Partitions (MSR, ESP)	428MB
Total	~23GB
Available space for OEM customizations on a 32GB (29GB usable) device	~6GB

Assumptions:

- Disk footprint calculated using GetDiskFreeSpaceEx()
- Data is collected immediately after OS setup, prior to NGEN running, prior to Idle Tasks.
- Measurement includes pagefiles
- Windows Update is disabled
- Build has multiple runs; the max footprint is used in this report
- Drive capacity is converted into Base-2 sizes: 32GB == 32,000,000,000 bytes == 30518MiB (or 29GiB).

### Language packs

Language packs comprise some of the largest disk space additions an OEM is likely to make. A language with all its optional components included can be estimated at 275MB each (size varies based on language) while Office language packs can be estimated at 300MB (also depending on language).

Windows will trim unused language packs once the end-user selects a primary language during OOB. After they're removed, these languages won't be available through push-button reset.

In order to maintain a smaller disk footprint, consider reducing the amount of languages installed on each shipping SKU.

### Servicing

Adding Windows Update KB packages can add significantly to the size of the disk. To reduce the footprint of an

image after adding updates:

1. Install KB and reboot if prompted
2. From an elevated command prompt, run the following commands:

```
dism.exe /online /cleanup-image /startcomponent
```

3. Restart the device.

## Real Time Clock (RTC)

The RTC is a battery-backed time source that stores and maintains system time when a device is powered off. ACPI 5.0 defines the Time & Alarm device which abstracts the underlying hardware device which maintains platform time. The ACPI Time & Alarm device is the preferred way to set and query platform time in Windows, even on a system with a traditional CMOS based RTC. The ACPI interface provides the time zone bias for the time value obtained from or written to the RTC. This extra field of information addresses a longstanding issue with the CMOS based RTC, where an operating system does not know how to interpret the time read from the hardware clock.

### Factory floor considerations

Windows queries RTC to update the system time when:

- No time synchronization service is available.
- The machine enters sleep (S3) or hibernate (S4) power state.
- Kernel Debugger is enabled.

OEMs typically provision the RTC in Local Time (LT) for devices shipped with Windows 7. Windows 7 exclusively uses the CMOS time interface to get RTC time, which is interpreted as LT. In Windows 8, we added support for the ACPI Time & Alarm device, but Windows 8 also uses the CMOS RTC, if it is available, and treats the time returned from it as LT. This behavior (related to the CMOS RTC interface) is incompatible with most non-Windows operating systems. Also, hosting providers like Azure want to use UTC time in their virtualized hardware to simplify management and migration of guests that might have a number of different time zones.

To address these concerns, Microsoft will be transitioning away from using the CMOS RTC interface and primarily relying on the ACPI Time & Alarm device.

For OEMs, the guidance is:

- Implement the ACPI Time & Alarm device.
- Set "CMOS RTC Not Present" flag in Fixed ACPI Description Table (FADT). The underlying hardware can still be the CMOS backed RTC, however Windows will only use the ACPI Time & Alarm Device if this flag is set.
- It is not recommended for the platform firmware to update the RTC across a daylight saving boundary. If it does, however, the firmware needs to ensure coordinated universal time (UTC) can always be calculated by adding the time zone bias to the time value, i.e.,  $UTC = LT + TZ$ . Windows will ignore the DST field received from the \_GRT control method.
- Invalidate TZ (set to 0xFF) through firmware if RTC time is ever updated through CMOS RTC interface.

## Ensuring a good first sign-in experience

The Windows team has seen a number of issues blocking good performance of user's first experience with Windows, and the following guidance should address common issues when preparing OS Images for your customers.

### Antimalware tools scanning disk during first sign-in

We have seen several instances where antimalware tools are doing a full disk scan during the user's first sign-in. The scanning competes with critical tasks occurring during the first sign-in process, resulting in very slow first sign-in, a degraded Start experience and slow system performance.

**Recommendation:** Devices should be configured to avoid full scan of the disk during first sign-in. Specific guidance for AV solutions should be provided by the AV Vendor.

#### Windows Defender

Add unique identifiers to your images to prevent Windows Defender from re-scanning all of the files you provided in the original disk image. To learn more, see [Configure a Trusted Image Identifier for Windows Defender](#).

#### Running NGEN commands

Native image GENeration is a task compiling .NET framework's MSIL (virtual machine code) into native images (platform specific executables). Generally it improves CLR application startup time by more than an order of magnitude. See [The Performance Benefits of NGen](#) for more details.

**Recommendation:** Follow these instructions to ensure .NET Framework apps are compiled. Please run the following commands after installing all OS updates:

On 32-bit, x86 ,or ARM devices:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\ngen.exe update /queue  
C:\Windows\Microsoft.NET\Framework\v4.0.30319\ngen.exe eqi
```

On 64-bit devices, do this for both versions of the .NET framework:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\ngen.exe update /queue  
C:\Windows\Microsoft.NET\Framework\v4.0.30319\ngen.exe eqi  
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\ngen.exe update /queue  
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\ngen.exe eqi
```

#### Graphics drivers

A Windows device should ship with the correct DirectX Graphics Driver for the system hardware. Failing to install the correct driver results in a fallback to a software-based graphics driver. This results in degraded experiences with Windows, including slower first sign-in performance.

**Recommendation:** Install the correct graphics driver for your hardware while in audit mode.

#### Frequently asked questions

##### Windows PE

- **Question:** Can I use the new WinPE to deploy, maintain and service previous Windows versions?

**Answer:** Updates to WinPE will not affect the currently supported Windows versions. You can use the updated WinPE to deploy previous Windows versions including Windows 7.

- **Question:** Do I have to migrate to the new deployment tools available as part of Windows 10?

**Answer:** No you don't. You only need to update to the newer version of the deployment tools (WinPE, DISM) if you want to implement Compact OS.

- **Question:** How does disk footprint optimization impact my PXE environment?

**Answer:** You only need to update your PXE environment to the newer version of the deployment tools (WinPE, DISM) if you are executing an "apply" while in your PXE environment. If you are only executing a download (file copy from server to client), you don't have to update your PXE environment.

##### Push-button recovery (PBR) and Windows Recovery Environment (WinRE)

- **Question:** Will WinRE also be updated?

**Answer:** Yes, a new WinRE.wim is needed.

- **Question:** Can the user still create a PBR USB key?

**Answer:** Yes. If the default partition layout is used, then no additional setup is required by the OEM to enable this.

#### Storage

- **Question:** Do you only support solid state disks?

**Answer:** We support both solid-state and traditional rotational media. We recommend that single-instancing is only used on solid-state disks, due to performance issues.

- **Question:** Can I use single-instancing of provisioning packages on a dual disk configuration (HDD + SSD)?

**Answer:** Single-instancing can only be implemented on the same disk.

#### Disk footprint

- **Question:** How will you delete the language packs that the user does not choose during OOB?

**Answer:** The language packs are deleted from the device, and will no longer be available during push-button recovery operations.

- **Question:** How are you calculating disk size? For example, you report a disk size of 14.8 GB on a 16 GB disk.

**Answer:** The disk capacity is converted into Base-2. For example, 16,000,000,000 (billion bytes) is equal to ~14.8 GB.

- **Question:** Can I only use Compact OS on small devices, such as those with 1 GB RAM and 16 GB disk?

**Answer:** Compact OS can be applied to any 32-bit or 64-bit platform with >=16GB of solid state storage.

- **Question:** Do you recommend using 16 GB disk on a 64-bit platform running 64-bit Windows?

**Answer:** We recommend a minimum disk capacity of 32 GB for 64-bit Windows.

#### Imaging and deployment

- **Question:** What are the changes to the DISM command to support Compact OS?

**Answer:** You can use DISM /Apply-Image ... /Compact and /Apply-CustomDataImage. For more info, see [DISM Image Management Command-Line Options](#).

- **Question:** Does Compact OS support both GPT and MBR partition layout?

**Answer:** Yes.

- **Question:** Is an updated OA3 tool required with Windows 10?

**Answer:** Yes.

- **Question:** Can I still use Windows SIM, unattend answer files and settings with Windows 10?

**Answer:** Yes, though some settings may have changed. See [Changed answer file settings from Windows 8.1 and Windows Server 2012 R2](#).

#### Language Packs and apps

- **Question:** Can we use multiple language packs?

**Answer:** Yes, however, we strongly recommend that you validate the disk footprint impact of the number of languages (Windows, Office, drivers and apps) per image.

- **Question:** Is there a change in how I install language packs?

**Answer:** Yes, you'll apply the base lp.cab in the same way as you did before in order to get multiple UI options, but to be able to enter text or get support, you'll need to add optional language components. For more info, see [Add Language Packs to Windows](#).

- **Question:** Is there a change in how I install desktop or Microsoft Store apps?

**Answer:** There is no change in how you install desktop or Microsoft Store apps from Windows 8.1.

- **Question:** What is the user experience on a multi-language configuration or when a user adds an additional language pack?

**Answer:** Language packs will continue to work the same way they do in previous versions of Windows.

- **Question:** Are there compatibility concerns with desktop apps?

**Answer:** The type of apps listed below will need to be carefully validated.

- Full volume encryption tools should not encrypt WIM images to limit performance impact. Such tools should check integrity of the unencrypted WIM to prevent tampering.
- Any tool that writes system files can be affected:
  - Imaging applications should perform block-level backup and restore of ALL volumes.
  - Faulty/Incomplete restore-operations can render a system unbootable.
  - Encryption/Back-Up/Defrag tools may unintentionally inflate system files.

- **Question:** Is Compact OS also applicable to Windows Embedded?

**Answer:** The Compact OS implementation and feature design we shared is limited to Windows 10 for desktop editions (Home, Pro, and Enterprise) . However, you should contact your Windows Embedded representative and ask about their disk footprint optimization plan.

#### **Policy**

- **Question:** Is there a change to the existing 10 GB free disk space policy requirement?

**Answer:** Refer to the updated Windows Hardware Compatibility Requirements.

#### **Servicing**

- **Question:** How will upgrade work especially on the recovery image with disk footprint optimization?

**Answer:** Upgrade will continue to work.

# Windows 10 in S mode manufacturing overview

1/18/2019 • 2 minutes to read

Windows 10 in S mode is a mode that can be applied to a Windows edition to enable a productive Windows experience that's streamlined for security and performance. By exclusively using apps in the Microsoft Store and ensuring that you browse safely with Microsoft Edge, Windows 10 in S mode keeps you running fast and secure day in and day out. The same technology that makes Windows 10 in S mode secure also creates some differences when creating software images for Windows 10 devices.

In previous versions of Windows, Windows 10 S was a separate edition that required working with a different Windows image than non-S editions. Starting with Windows 10, version 1803, you can [Enable S mode](#) on supported Windows editions.

When you [plan your Windows 10 in S mode image and deployment](#), you'll have to ensure that your customizations will work with Windows 10 in S mode, as well as the manufacturing environment.

While the overall process is similar to building other Windows 10 devices, Windows 10 in S mode has some additional considerations.

## In this section

<a href="#">Planning a Windows 10 in S mode or Windows 10 S image</a>	Gives an overview of what to consider when planning a Windows 10 S image.
<a href="#">Manufacturing environment</a>	Describes how the manufacturing environment behaves with Windows 10 S.
<a href="#">Manufacturing mode</a>	Explains how to enable and disable manufacturing mode.
<a href="#">Enable S mode</a>	Covers how to use unattend to enable S mode.
<a href="#">Windows 10 S deployment lab</a>	Provides step by step instructions on how to create a Windows 10 S deployment.

# Planning a Windows 10 in S mode deployment

1/18/2019 • 7 minutes to read

Building a Windows 10 in S mode or Windows 10 S image is like building an image for any other desktop edition of Windows, with some key differences. You can add apps, drivers, and customizations to Windows 10 in S mode, but you'll have to make sure they're supported.

## Executables

When planning a deployment, make sure you understand what runs, and what is blocked in Windows 10 in S mode. Choose and test customizations that work with Windows 10 in S mode and won't interrupt your deployment. If you need to run unsigned code, you can [enable the manufacturing mode registry key](#) which allows you to run unsigned code, but once the PC ships the unsigned code will be blocked.

### What runs on Windows 10 in S mode?

Only run executable code that is signed with a **Windows**, **WHQL**, **ELAM**, or **Store** certificate from the [Windows Hardware Developer Center Dashboard](#). This includes companion apps for drivers.

Apps not signed with one of the certificates mentioned, including companion apps, are blocked. When a blocked app is run, the user is notified that the app cannot run.

### What is blocked in Windows 10 in S mode?

The following components are blocked from running in Windows 10 in S mode. Any script or application that calls one of these blocked components will be blocked. If your manufacturing process uses scripts or applications that rely on blocked components, you can temporarily [enable manufacturing mode](#) for configuring and testing, but you can't ship a PC with manufacturing mode enabled.

- bash.exe
- cdb.exe
- cmd.exe
- cscript.exe
- csi.exe
- dnx.exe
- fsi.exe
- hh.exe
- infdefaultinstall.exe
- kd.exe
- lcssmanager.exe
- msbuild.exe
- mshta.exe
- ntsd.exe
- powershell.exe
- powershell\_ise.exe
- rcsi.exe
- reg.exe
- regedit.exe
- regedt32.exe
- regini.exe

- syskey.exe
- wbemtest.exe
- windbg.exe
- wmic.exe
- wscript.exe
- wsl.exe

## Microsoft Office

For the best customer and end-user experience, preinstall Microsoft Office using the Office 16.5 OPK. This version of Microsoft Office is the same version that's available in the Microsoft Store, and runs on Windows 10 in S mode. We show you how to enable S mode and install the Office 16.5 OPK in the [OEM deployment of Windows 10 lab](#).

## Testing your app

For information on how to test your app, see [Test your Windows app for Windows 10 in S mode](#).

## Drivers

For Windows 10 in S mode driver guidelines and requirements, see [Windows 10 in S mode Driver Requirements](#).

## Customizations

Not all customizations are supported in Windows 10 in S mode. This section shows which customizations are supported, which customizations are not supported, and how to enable manufacturing mode that allows you to perform customizations in audit mode.

### Supported customizations

The following table shows customizations in Windows 10 in S mode, the mechanism to deploy the customizations, and the environment where you can deploy the customizations.

CUSTOMIZATION OR TASK	MECHANISM	ENVIRONMENT
Language Packs	DISM	Offline, WinPE, Audit Mode
Features on Demand	DISM	Offline, WinPE, Audit Mode
Start Menu Layout	layoutmodification.xml	N/A
OEM Taskbar tiles	taskbarlayoutmodification.xml	N/A
InkWorkstationTiles	InkWorkstationLayoutModification.xml	N/A
OOBE customizations	OOBE.xml, OOBE folder structure	OOBESystem pass
UWP apps	DISM	Offline, WinPE, Audit mode
Bridge apps	DISM	Offline, WinPE, Audit Mode
Drivers with no unsigned or win32 scripts/exe/binaries	DISM	Offline, WinPE, Audit Mode
Wallpaper	unattend.xml	N/A

CUSTOMIZATION OR TASK	MECHANISM	ENVIRONMENT
Command prompt from OOBE using <Shift + F10>	Manufacturing reg key	OOBE

## Unsupported customizations

The following tables shows customizations that are not supported in Windows 10 in S mode.

CUSTOMIZATION OR TASK	MECHANISM	ENVIRONMENT
Driver installation with setup.exe	Unsupported	Unsupported
Drivers with co-installers or dependent on scripts or cmd execution	Unsupported	Unsupported
Win32 apps	Unsupported	Unsupported
First logon commands	Unsupported	Unsupported

### IMPORTANT

Bing is set as the search default and Microsoft Edge is set as the default browser in Windows 10 in S mode. These settings can't be changed.

## Enable customizations in audit mode

To enable customizations in audit mode, you have to enable manufacturing mode by adding a registry key to your offline image. Manufacturing mode allows you to run unsigned code that is normally blocked. For instructions on how to add or remove the manufacturing registry key, see [Manufacturing registry key](#).

You'll also have to configure ScanState to exclude the registry key when capturing your recovery package. This ensures that the registry key doesn't get restored during reset or recovery scenarios. We'll cover how to exclude the key from recovery in the [Windows 10 in S mode deployment lab](#)

### IMPORTANT

Don't ship your Windows 10 in S mode PC with the registry in place. You'll have to remove the registry key prior to shipping the device.

## Upgrade and switch paths

### Upgrade paths

Windows 10 in S mode (Windows 10, version 1803 or later) allows the following upgrade paths:

UPGRADE PATHS
Windows 10 Home in S mode to Windows 10 Pro in S mode
Windows 10 Pro in S mode to Windows 10 Enterprise in S mode
Windows 10 Pro in S mode to Windows 10 Education in S mode

Windows 10 S allows the following upgrade paths:

## UPGRADE PATHS

Windows 10 S to Professional

Windows 10 S N to Professional N

Windows 10 S to Enterprise

Windows 10 S N to Enterprise N

Windows 10 S to Education

Windows 10 S N to Education N

Windows 10 S to Professional Education

Windows 10 S N to Professional Education N

For information on using DISM to change the a Windows image to a different edition, see [Change the windows image to a higher edition using dism](#).

## Switch paths

Windows 10 in S mode can be switched to a non-S edition. A user can switch modes through the Microsoft Store. The following shows the available switch paths:

### SWITCH PATHS

Windows 10 Home in S mode to Windows 10 Home

Windows 10 Pro in S mode to Windows 10 Pro

Windows 10 Enterprise in S mode to Windows 10 Enterprise

Windows 10 Education in S mode to Windows 10 Education

### NOTE

Starting with Windows 10, version 1803, switching from S mode doesn't require a reboot.

# Recovery

## Built-in recovery

Windows 10 S includes a recovery solution that enables a user to restore, refresh, or troubleshoot their PC. Recovery in Windows 10 S has some differences from other editions of Windows. These differences are:

- Third party recovery solutions are NOT supported
- Extensibility points for customizations documented in this section is supported
  - OEM tools in WinRE is not supported
  - CMD prompt in WinRE will be enabled but only allow execution of inbox WinRE binaries
  - Extensibility script must be in the form of a \*.CMD
  - Does not call any of the blocked inbox components except reg.exe and wmic.exe

## Recovery scenarios for S mode

**Note:** This applies to Windows 10 in S mode (Windows 10, version 1803 and later).

This table shows what Windows edition and mode will result from a user performs a system recovery:

PREINSTALLED OS	WAS WINDOWS SWITCHED TO NON-S MODE?	WAS WINDOWS UPGRADED TO A DIFFERENT EDITION?	PCS MODE/EDITION AT RECOVERY	PBR (BOTH "KEEP MY FILES" AND "REMOVE EVERYTHING")	BMR USING OEM FACTORY RECOVERY IMAGE	BMR USING USER-CREATED MEDIA
Home S	No	No	Home S	Home S	Home S	The edition and mode which the media was created
Home S	No	Yes (Pro S)	Pro S	Pro S	Home S	The edition and mode which the media was created
Home S	Yes (Home)	No	Home	Home	Home S	The edition and mode which the media was created
Home S	Yes (Home)	Yes (Pro)	Pro	Pro	Home S	The edition and mode which the media was created
Pro S	No	No	Pro S	Pro S	Pro S	The edition and mode which the media was created
Pro S	No	Yes (Enterprise S)	Enterprise S	Enterprise S	Pro S	The edition and mode which the media was created
Pro S	Yes (Pro)	No	Pro	Pro	Pro S	The edition and mode which the media was created
Pro S	Yes (Pro)	Yes (Enterprise or Workstations)	Enterprise or Workstations	Enterprise or Workstations	Pro S	The edition and mode which the media was created

- BMR: Bare metal recovery
- PBR: Push button reset

### **Validating recovery in your deployment**

After you configure your Windows 10 S PC for recovery scenarios, validate that it is working properly by verifying that these scenarios run successfully:

- Run refresh recovery and validate the user files are preserved and your factory desktop customizations are restored.
- Run reset recovery and validate the user files and profile are removed and your factory desktop customizations are restored.
- Validate extensibility scripts in the simulated RS3 enforcement level using the provided policy file.
- If you created a recovery package with ScanState, ensure that the manufacturing key was excluded from capture.

### **Retail Demo eXperience (RDX)**

In the Retail Demo Experience (RDX), Windows detects Windows 10 S editions and displays marketing messages with info about Windows 10 S for Windows and Office.

# Manufacturing environment

3/21/2019 • 2 minutes to read

## Overview

This topic covers the differences in the Windows 10 in S mode and Windows 10 S manufacturing environments from other Windows manufacturing environments.

## Code integrity policy

The code integrity policy (CI) blocks the execution of unsigned or improperly signed binaries. Using unsupported binaries is only recommended when performing lab or factory image customization, or during deployment where the execution environment is either WinPE or Audit Mode.

Once the CI policy is enabled on a system, it is enabled in two places:

1. Windows 10 in S mode, enforced at boot.
2. EFI firmware policy, enforced during firmware load and OS boot.

## WinPE

The Windows Preinstallation Environment (WinPE) behaves the same for Windows 10 in S mode and Windows 10 S as it does for Windows Home or Windows Professional.

For more information about WinPE, see [Windows PE](#).

## DISM

### Adding a Windows 10 S image to a WIM

If you want a single WIM that includes multiple Windows editions including Windows 10 S, you can add/append your Windows 10 S image to an existing WIM, which allows you to specify the Windows 10 S image index during DISM /apply.

To see more about adding/appending images to an existing WIM, see [Append, apply, and export volume images with a Windows Image \(.wim\) file](#).

### Detect Windows 10 S with DISM

You can use DISM to detect Windows 10 S (offline in WinPE or in Audit mode). In Audit mode, use

```
DISM /online /get-currentedition . If an image is Windows 10 S, the command should return S. In WinPE, use  
DISM /image:c:\ /get-currentedition .
```

See [DISM Windows edition-servicing command-line options](#) to see additional commands for working with Windows editions.

## Audit mode

Audit mode is available when manufacturing a Windows 10 S PC. By default, the [blocked inbox components](#) are blocked in audit mode. If you need to use blocked inbox components during the manufacturing process, you can [enable manufacturing mode](#). If you enable manufacturing mode, you'll have to make sure to [disable manufacturing mode](#) prior to shipping your PC.

To learn more about Audit Mode, see [Audit Mode overview](#).

## Factory device diagnostics

During factory testing, Win32-based diagnostic tools can be run by using one of the following options:

1. Windows 10 S in Audit Mode with Secure Boot turned off and the manufacturing registry key in place.

or

2. In a separate non-Windows 10 S test operating system.

# Enable manufacturing mode

1/24/2019 • 2 minutes to read

## Overview

To run scripts, installers, and diagnostic tools on the factory floor, Windows 10 in S mode and Windows 10 S have a manufacturing mode. This mode allows you to run unsigned code in Audit Mode. Enable manufacturing mode by adding a registry key to an offline image. Disable manufacturing mode by removing the registry key when booted into audit mode.

### IMPORTANT

Don't ship a Windows 10 in S mode PC with the registry in place. Remove the registry key prior to shipping the device.

Before shipping a Windows 10 in S mode PC, remove the manufacturing registry key and exclude it from recovery packages.

## Enable manufacturing mode

Here's how to enable manufacturing mode.

On your technician PC:

1. Mount your Windows 10 in S mode image.

```
Dism /Mount-Wim /WimFile:D:\sources\install.wim /index:1 /MountDir:C:\mount\windows
```

Where D: is your Windows 10 in S mode installation media.

2. Load the SYSTEM registry hive from your mounted image into regedit on your technician PC. We'll use a temporary hive called HKLM\Windows10S.

```
reg load HKLM\Windows10S C:\Mount\Windows\Windows\System32\Config\System
```

3. Add the manufacturing registry key.

```
reg add HKLM\Windows10S\ControlSet001\Control\CI\Policy /v ManufacturingMode /t REG_DWORD /d 1
```

4. Unload the registry hive from your technician PC.

```
reg unload HKLM\Windows10S
```

5. Unmount the image and commit changes.

```
Dism /Unmount-Image /MountDir:"C:\mount\windows" /Commit
```

The Windows 10 in S mode image now has the manufacturing key that will allow you to make changes in audit mode.

# Remove the manufacturing registry key

When you're finished making changes to your PC in audit mode, you'll remove the manufacturing registry key.

While still booted into audit mode:

1. Open Command Prompt.
2. Remove the registry key.

```
reg delete HKLM\SYSTEM\ControlSet001\Control\CI\Policy /v ManufacturingMode
```

The manufacturing registry key is now removed. You can check the Registry Editor to double check that the key has been removed.

On your Windows 10 in S mode PC in audit mode:

1. Open Registry Editor by clicking on the start menu and typing `regedit` and press enter.
2. Use the registry browser in the left pane to navigate to  
Computer\HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet001\Control\CI\Policy.
3. Under *policy*, you should not see a key called *manufacturingmode*.

## Exclude the manufacturing registry key from recovery

When you create a recovery package, exclude the manufacturing registry key. Create an exclusion file that tells scanstate to skip the registry key when it captures a recovery package.

1. Create an .xml file in a text editor.
2. Copy and paste the following code. This tells ScanState to not capture the registry key in the recovery package that it creates:

```
<?xml version="1.0" encoding="UTF-8"?>
<migration urlid="https://www.microsoft.com/migration/1.0/migxmltext/ExcludeManufacturingMode">
<component type="System">
    <displayName>Exclude manufacturing regkey</displayName>
    <role role="Settings">
        <rules context="System">
            <unconditionalExclude>
                <objectSet>
                    <pattern type="Registry">HKLM\SYSTEM\CurrentControlSet\Control\CI\Policy
[ManufacturingMode]</pattern>
                </objectSet>
            </unconditionalExclude>
        </rules>
    </role>
</component>
</migration>
```

3. Save the file as exclusion.xml.
4. When you use scanstate to generate a recovery package, add `/i:exclusion.xml` to the scanstate command to exclude the manufacturing key from the capture. This command creates a recovery package that excludes the manufacturing registry key, and places it into the recovery folder.

```
Scanstate.exe /config:T:\deploymenttools\Config_SettingsOnly.xml /o /v:13 /ppkg
C:\Recovery\Customizations\usmt.ppkg /i:exclusion.xml /l:C:\Scanstate.log
```

# Enable S mode

1/18/2019 • 2 minutes to read

This topic shows how to add S mode to a supported Windows 10 edition. The following information only applies to Windows 10, version 1803 or later.

## Overview

Beginning with Windows 10, version 1803, you can enable S mode on supported editions. This is a change from previous versions of Windows where Windows 10 S was its own edition with its own base media. To enable S mode, you'll create an unattend file, and then use DISM to apply the unattend file to a mounted Windows image.

We recommend enabling S mode and adding your manufacturing key at the same time. We show you how to do this in the [Windows 10 in S Mode deployment lab](#), as well as in the [OEM deployment of Windows 10 for desktop editions lab](#).

## Enable S Mode

To enable S mode in a Windows image, use an Unattend file that has a setting in Pass 2 - offlineServicing, and use DISM to apply it to our mounted Windows image.

1. Mount your Windows image.
2. Use Windows SIM to create or modify an unattend file.
3. Add the amd64\_Microsoft\_Windows\_CodeIntegrity component to Pass 2 offline Servicing.
4. Set amd64\_Microsoft\_Windows\_CodeIntegrity\SkuPolicyRequired to 1. The offline servicing pass in your unattend.xml file should look like this:

```
<settings pass="offlineServicing">
<component name="Microsoft-Windows-CodeIntegrity"
    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">
    <SkuPolicyRequired>1</SkuPolicyRequired>
</component>
</settings>
```

5. Save the answer file in the `Windows\Panther` folder of your mounted image as unattend.xml.
6. Use DISM to apply the unattend file and enable S Mode:

```
dism /image:C:\mount\windows /apply-unattend:C:\mount\windows\windows\panther\unattend.xml
```

**Note:** Only Pass 2 - offline Servicing is processed when an unattend file is applied with DISM.

S mode is now applied to the Windows image. When the PC boots, the same Code Integrity policy that is enforced in Windows 10 S will be enforced on your Windows installation. If you are going to boot the PC into Audit Mode,

you'll have to enable manufacturing mode. [Enable manufacturing mode](#) describes how to do that.

# Windows 10 in S mode deployment lab

1/25/2019 • 12 minutes to read

Starting with Windows 10 in S mode, creating a deployment of Windows 10 in S mode starts with a regular base Windows 10 image. S mode is applied by applying an unattend file to a mounted Windows image. When working with a PC that is in S mode, the manufacturing process has some differences when compared to other versions of Windows. When [planning your deployment](#), you have to make sure that your [drivers](#) and [apps](#) are supported by Windows 10 in S mode.

This lab walks you through the process of configuring a Windows 10 image in S mode for deployment. We'll customize an image, set S mode with unattend, add the manufacturing registry key in WinPE, and then remove the registry key in Audit Mode. Then we'll configure recovery and prepare the image for shipment.

## NOTE

If you're building a Windows 10 S image, we'll call out the differences between Windows 10 in S mode and Windows 10 S.

Let's get started.

## Get the tools you need

To start building an image for deployment, here's what you'll need:

- Windows 10 image
  - For Windows 10 in S mode, use a Windows Home or Windows Professional image **or**
  - For Windows 10 S, use a Windows 10 S image
- Technician PC running Windows 10, Version 1803 or later
- Reference PC where you can deploy your image
- The [latest version of the ADK](#) installed on your technician PC
- A USB key that you can format
- [Deployment scripts](#)
- Customizations such as drivers or language packs
- The latest General Distribution Release update from [the Microsoft Update Catalog](#)

## Format your USB key

To prepare your USB drive, you'll create separate FAT32 and NTFS partitions. The following creates two partitions on a USB drive; one 2GB FAT32 partition, and one NTFS partition that uses the rest of the available space on the drive. You want to make sure that your USB drive has enough free space for the 2GB WinPE partition and to hold large images on the NTFS partition:

1. On your technician PC, start the **Deployment and Imaging Tools Environment** as an administrator:
  - Click **Start**, type **Deployment and Imaging Tools Environment**. Right-click **Deployment and Imaging Tools Environment** and select **Run as administrator**.
2. Open diskpart.

```
diskpart
```

3. Select your your USB key's disk number, and run the `clean` command. This command will make any data on your USB key inaccessible. Make sure you've backed up any data that you want to keep.

```
list disk
select <disk number>
clean
```

Where `<disk number>` is the number of your USB drive

4. Create the FAT32 partiton for WinPE, label it "Windows PE" and mark it active.

```
create partition primary size=2000
format quick fs=fat32 label="Windows PE"
assign letter=P
active
```

5. Create the NTFS partition where you'll store your images and customizations.

```
create partition primary
format fs=ntfs quick label="Data"
assign letter=T
list vol
exit
```

## Make a bootable WinPE partition on your USB key

On your technician PC:

1. Open the Deployment and Imaging Tools Environment as administrator.
2. Copy the base WinPE files into a new folder:

```
copype amd64 C:\winpe_amd64
```

3. Copy the WinPE files to your FAT32 partition.

```
MakeWinPEMedia /UFD C:\winpe_amd64 P:
```

When prompted, press **Y** to format the drive and install WinPE.

For more information about how to create a WinPE drive, see [WinPE: Create USB bootable drive](#).

## Create Data USB partition

1. In File Explorer, open the deployment scripts zip and copy the scripts folder to the Data partition of your USB drive.
2. From the Deployment and Imaging Tools Environment use copydandi.cmd to copy deployment and imaging tools to your USB drive

```
copydandi amd64 T:\deploymenttools
```

3. Copy any other customizations you need for Audit mode.

## Mount install.wim and winre.wim

Mounting a Windows image is the same process that we used to mount the WinPE image earlier. When you mount your Windows image (install.wim), you'll be able to access a second image, WinRE.wim, which is the image that supports recovery scenarios. Updating install.wim and WinRE.wim at the same time helps you keep the two images in sync, which ensures that recovery goes as expected.

1. Mount the Windows 10 iso in File Explorer.
2. Create a temporary folder (c:\temp) and then copy install.wim from D:\Sources (Where D: is the drive letter of the mounted image) to the temporary folder.

```
md c:\temp  
copy d:\sources\install.wim c:\temp
```

3. Open the Deployment and Imaging Tools Environment as an administrator.
4. Create a folder for mounting images, and then mount install.wim.

```
Md C:\mount\windows  
Dism /Mount-Wim /WimFile:C:\temp\install.wim /index:1 /MountDir:C:\mount\windows
```

5. Create a mount folder for the Windows RE Image file from your mounted image, and then mount the WinRE image.

```
Md c:\mount\winre  
Dism /Mount-Wim /WimFile:C:\mount\windows\Windows\System32\Recovery\winre.wim /index:1  
/MountDir:C:\mount\winre
```

**Troubleshoot:** If winre.wim cannot be seen under the specified directory, use the following command to set the file visible:

```
attrib -h -a -s C:\mount\windows\Windows\System32\Recovery\winre.wim
```

**Troubleshoot:** If mounting operation fails, make sure the Windows 10 version of DISM is the one installed with the Windows ADK is being used and not an older version that might be on the Technician Computer. Don't mount images to protected folders, such as the User\Documents folder. If DISM processes are interrupted, consider temporarily disconnecting from the network and disabling virus protection.

For more information about mounting a Windows image, see [Mount and Modify a Windows Image Using DISM](#).

To learn about customizing WinRE, see [Customize Windows RE](#).

## Enable customizations

### Enable S mode

**Note:** This section doesn't apply if you're starting with Windows 10 S media. If this doesn't apply, [skip to the next section](#).

Before customizing an image, use the offline servicing pass of unattend to set a Windows PC to S mode.

1. Use Windows SIM to create an unattend file.
2. Add `SkuPolicyRequired` to the offlineServicing pass

3. Set `SkuPolicyRequired` to `1`

4. Save the file as `unattend.xml`

5. Copy `unattend.xml` to the mounted Windows image:

```
MkDir c:\mount\windows\Windows\Panther  
Copy unattend.xml C:\mount\windows\Windows\Panther\unattend.xml
```

6. Apply the `unattend` file to your mounted image:

```
DISM /Image=C:\mount\windows /Apply-Unattend=C:\mount\windows\Windows\Panther\unattend.xml
```

When the PC boots, it will boot into S mode with CI policies enforced. If you need to make customizations to the Windows image, you'll have to enable the manufacturing registry key. This will allow you to make changes in audit mode.

### Add the manufacturing registry key

Enabling manufacturing mode is a step you'll have to do when working with Windows 10 in S mode and Windows 10 S. To enable customizations during the manufacturing process, you'll have to add a registry key that gives you the ability to run unsigned code when booted into audit mode. This can help you build and test your image when getting a PC ready to ship.

We'll add the customization registry key to the mounted image by loading the mounted image's SYSTEM registry hive, and then adding a key. Then we'll configure ScanState to exclude the registry key when capturing your recovery package to ensure that the registry key doesn't get restored during reset or recovery scenarios.

#### IMPORTANT

Don't ship your PC with the registry in place. Remove the registry key prior to shipping the device.

1. Load the SYSTEM registry hive from your mounted image into regedit on your technician PC. We'll use a temporary hive called `HKLM\Windows10S`.

```
reg load HKLM\Windows10S C:\Mount\Windows\Windows\System32\Config\System
```

2. Add the following key to the registry have that you just mounted.

```
reg add HKLM\Windows10S\ControlSet001\Control\CI\Policy /v ManufacturingMode /t REG_DWORD /d 1
```

3. Unload the registry hive from your technician PC.

```
reg unload HKLM\Windows10S
```

The mounted image now has the manufacturing key that will allow you to make changes in audit mode. You'll have to remove it before shipping the PC.

To learn about the Windows 10 S manufacturing registry key, see [Windows 10 S manufacturing mode](#).

### Create exclusion.xml

Now we'll create a file that automates the exclusion of the customizations registry key when you capture settings for recovery. This ensures that your PC doesn't restore the customization registry key during the recovery process.

1. Create an xml file in a text editor.
2. Copy and paste the following code. This tells ScanState to not capture the registry key in the recovery package that it creates:

```
<?xml version="1.0"?> <encoding>UTF-8</encoding>
<migration urlid="https://www.microsoft.com/migration/1.0/migxml/ext/ExcludeManufacturingMode">
<component type="System">
<displayName>Exclude manufacturing regkey</displayName>
<role role="Settings">
<rules context="System">
<unconditionalExclude>
<objectSet>
<pattern type="Registry">HKLM\SYSTEM\CurrentControlSet\Control\CI\Policy
[ManufacturingMode]</pattern>
</objectSet>
</unconditionalExclude>
</rules>
</role>
</component>
</migration>
```

3. Save the file as exclusion.xml.

We'll use this config file when we capture a ScanState package for recovery later in the lab.

You can learn about excluding files and settings from a ScanState package at [Exclude Files and Settings](#).

## Add drivers

Like other versions of Windows, you can add drivers to a Windows 10 S image to ensure that hardware is setup and working the first time a user boots into Windows. Make sure that the drivers you add to your Windows 10 S are compatible with Windows 10 S and won't be blocked.

1. Add a single driver to your Windows and WinRE images from an .inf file. In this example, we're using a driver named media1.inf:

```
Dism /Add-Driver /Image:"C:\mount\windows" /Driver:"C:\Drivers\PnP.Media.V1\media1.inf"
Dism /Add-Driver /Image:"C:\mount\winre" /Driver:"C:\Drivers\PnP.Media.V1\media1.inf"
```

Where "C:\Drivers\PnP.Media.V1\media1.inf" is the .inf file for the driver you're adding.

```
Dism /Add-Driver /Image:"C:\mount\windows" /Driver:c:\drivers /Recurse
```

2. Verify that the drivers are part of the images:

```
Dism /Get-Drivers /Image:"C:\mount\windows"
Dism /Get-Drivers /Image:"C:\mount\winre"
```

Check the list of packages and verify that the list contains the drivers you added.

For more information about adding drivers to an offline Windows image, see [Add and Remove Drivers to an Offline Windows Image](#).

## Add a language (optional)

In this section, we'll add the German (de-de) language pack to the mounted Windows and WinRE images.

## 1. Add German language package to the Windows image.

Use the language packs from the 64-bit ISO:

```
Dism /Add-Package /Image:C:\mount\windows /PackagePath:"E:\x64\langpacks\Microsoft-Windows-Client-Language-Pack_x64_de-de.cab"
```

Where E: is the drive letter of the mounted language pack ISO.

## 2. Add the German language pack to Windows RE. Language packs are available as part of the ADK, and ensure that a user's language is available during recovery scenarios.

```
Dism /image:C:\mount\winre /add-package /packagepath:"E:\Windows Preinstallation Environment\x64\WinPE_0Cs\de-de\lp.cab"
```

See [Add and remove language packs offline using DISM](#) for more information.

## Add the latest General Distribution Release (GDR)

Install the latest GDR package that include the latest bug fixes and OS changes.

**Important:** Install GDR packages **after** you install language packs, AppX packages, and Features on Demand. If you install a GDR prior to adding these, you'll have to reinstall the GDR.

### 1. Download the GDR (KB 4020102) from the [Microsoft Update Catalog](#).

### 2. Use DISM /add package to add the GDR to the mounted images

```
dism /image:"C:\mount\windows" /add-package /packagepath:C:\temp\windows10.0-kb4020102-x64_9d406340d67caa80a55bc056e50cf87a2e7647ce.msu  
dism /image:"C:\mount\winre" /add-package /packagepath:C:\temp\windows10.0-kb4020102-x64_9d406340d67caa80a55bc056e50cf87a2e7647ce.msu
```

### 3. Use DISM to cleanup your image.

```
DISM /Cleanup-Image /Image=C:\mount\winre /StartComponentCleanup /ScratchDir:C:\Temp
```

See [Add or remove packages offline using DISM](#) for more information about adding packages to your Windows image.

## Unmount WinRE Image and make a copy

Now that you have made all of your offline customizations, you can unmount your images.

### 1. Close all applications that might access files from the images.

### 2. Commit the changes and unmount the WinRE and Windows images:

```
Dism /Unmount-Image /MountDir:"C:\mount\winre" /Commit  
Dism /Export-Image /SourceImageFile:c:\mount\windows\windows\system32\recovery\winre.wim /SourceIndex:1 /DestinationImageFile:c:\mount\winre-optimized.wim  
del c:\mount\windows\windows\system32\recovery\winre.wim  
copy c:\mount\winre-optimized.wim c:\mount\windows\windows\system32\recovery\winre.wim
```

## Unmount install.wim

```
Dism /Unmount-Image /MountDir:"C:\mount\windows" /Commit
```

## Copy install.wim and winre.wim to your USB drive

```
copy c:\temp\install.wim t:\  
copy c:\temp\winre-optimized.wim t:\
```

## Deploy the image to reference PC

1. Boot your reference PC to WinPE.
2. Use the deployment scripts to apply your modified install.wim image.

```
T:\Deployment\walkthrough-deploy.bat t:\install.wim
```

## Boot to audit mode and make changes

1. Boot your reference PC if it's not already booted.
2. When the device boots to OOBE, press Ctrl+Shift+F3 to enter Audit mode.
3. The PC will restart into audit mode.
4. Make changes to the PC. See the table on [Planning a Windows 10 in S mode image](#) to see which customizations are available in audit mode.

To learn about audit mode, see [Audit mode overview](#). To learn about Audit mode's behavior with Windows 10 S, see [Audit mode in Windows 10 S manufacturing environment](#).

## Capture your audit mode changes for the recovery tools

Now that you've customized your image in Audit mode, you can use ScanState to capture the package so the customizations are available in recovery scenarios.

1. Use ScanState that you copied to your USB key to capture customizations into a provisioning package. Use the exclusion.xml file that you created earlier to ensure that the manufacturing registry key is not restored during recovery.

```
md c:\Recovery\Customizations  
T:\deploymenttools\scanstate /config:T:\deploymenttools\Config_SettingsOnly.xml /o /v:13 /ppkg  
c:\recovery\customizations\usmt.ppkg /i:exclusion.xml /l:C:\Scanstate.log
```

2. When the capture completes successfully, delete the ScanState logfile: `del c:\scanstate.log`.

## Remove the manufacturing registry key

When you're finished customizing your PC in audit mode, you have to remove the manufacturing registry key that allows you to run unsigned code on Windows 10 S.

To remove the registry key, run the following command as administrator when booted into audit mode on the reference PC:

```
reg delete HKLM\SYSTEM\ControlSet001\Control\CI\Policy /v ManufacturingMode
```

## Add WinRE back into your captured image

To ensure that your WinRE image is captured for your final deployment, copy your exported WinRE-optimized.wim image to your Windows 10 S image.

```
xcopy t:\winre-optimized.wim c:\windows\system32\recovery\winre.wim
```

## Sysprep and shut down the PC

1. Open Command Prompt.
2. Run sysprep to reseal the PC and make it ready for capture.

```
c:\windows\system32\sysprep\sysprep /generalize /oobe /shutdown
```

## Capture the image

1. Boot the reference PC into WinPE.
2. Identify the drive letter of the Windows partition in diskpart:

```
diskpart  
list volume  
exit
```

3. Use DISM to capture the Windows partition.

```
dism.exe /capture-image /ImageFile:"T:\Images\Windows10S.wim" /capturedir:C:\ /Name:"Windows10S"
```

Where C:\ is the Windows partition.

See [Capture and apply Windows system and recovery partitions](#) for more information.

## Deploy your image and verify customizations and recovery

### Apply your image

1. Boot your reference PC into WinPE.
2. Apply your Windows 10 in S mode image (Windows10S.wim) to the PC. This will overwrite any existing Windows installations.

```
T:  
cd Deployment  
T:\Deployment\applyimage.bat T:\images\Windows10S.wim
```

### Verify customizations

1. Boot the reference PC. This is the first time booting the PC with your new Windows image.
2. If you installed additional languages, verify that these preinstalled languages appear and can be selected by the user during OOBE.

3. Validate the desktop customizations you made correctly after OOBE is complete.

### Verify recovery

To verify recovery is working as expected, perform the following validation tasks:

- Run refresh recovery and validate the user files are preserved and your factory desktop customizations are restored.
- Run reset recovery and validate the user files and profile are removed and your factory desktop customizations are restored.
- Validate extensibility scripts in the simulated RS3 enforcement level using the provided policy file.
- If you created a recovery package with ScanState, ensure that the manufacturing key was excluded when the package was captured.

## Ship the PC

Now that you have an image, you are ready to build and ship Windows 10 in S mode PCs. Make sure that the manufacturing registry key is removed and Secure Boot is enabled on shipped PCs.

# IoT Enterprise overview

1/25/2019 • 2 minutes to read

Windows 10 IoT Enterprise is a full version of Windows 10 that delivers enterprise manageability and security to IoT solutions.

Because it's a binary equivalent to Windows 10 Enterprise, Windows 10 IoT Enterprise shares all the benefits of the worldwide Windows ecosystem, including the same familiar development and management tools as client PCs and laptops. However, when it comes to licensing and distribution, the desktop version and IoT versions differ.

Windows 10 IoT Enterprise offers both LTSC and SAC options, and OEMs can choose the option they need for their devices.

This section covers how to build, customize and deploy a Windows 10 IoT Enterprise image with the following labs:

## LAB

[Get the tools you need to create an IoT Enterprise image](#)

[Lab 1a: Create a basic image](#)

[Lab 1b: Customize a reference device in Audit mode](#)

[Lab 2: Configure device lockdown features](#)

[Lab 3: Configure policy settings on IoT Enterprise devices](#)

[Lab 4: Sysprep, capture, and deploy](#)

[Lab 5: Configure Shell Launcher or Assigned Access](#)

## NOTE

We also have guidance for a more automated solution. If you want to try script-based image customization and deployment, visit our [GitHub repo](#).

# Get the tools needed to customize Windows 10 IoT Enterprise

4/18/2019 • 2 minutes to read

## What is Windows 10 IoT Enterprise?

Windows 10 IoT Enterprise is a full version of Windows 10 that delivers enterprise manageability and security to IoT solutions. Windows 10 IoT Enterprise shares all the benefits of the worldwide Windows ecosystem. It is a binary equivalent to Windows 10 Enterprise, so you can use the same familiar development and management tools as client PCs and laptops. However, when it comes to licensing and distribution, the desktop version and IoT versions differ. Note that Windows 10 IoT Enterprise offers both LTSC and SAC options, and OEMs can choose the one they need for their devices.

## Getting started

Before reaching out to an Embedded/IoT Distributor, we recommend working with a device that meets the [Windows 10 Hardware requirements](#). You can load your PC or recommended device with an evaluation copy of Windows 10 Enterprise in order to begin prototyping right away.

In order to start your journey in manufacturing with Windows 10 IoT Enterprise, you'll need to reach out to a distributor from [this list](#).

## What you need to build IoT Enterprise images

You'll need the following tools to go through any of the labs in this section and create OEM images using the Windows 10 IoT Enterprise Operating System.

## PCs and devices

Here's how we'll refer to them:

- Technician PC: Your work PC. This PC should have at least 15GB of free space for installing the software and for modifying IoT Enterprise images. We recommend using either Windows 10 or Windows 8.1 with the latest updates. The minimum requirement is Windows 7 SP1, though this may require additional tools or workarounds for tasks such as mounting .ISO images.
  - Configure the technician PC so it has the following:
    - [Windows ADK](#) with Deployment Tools, Configuration Designer, and the Windows PE add-on installed
    - Windows 10 IoT Enterprise 2019 LTSC OPK
    - Feature on Demand ISO
    - Language Pack ISO
- IoT device: A test device or board that represents all of the devices in a single model line. Depending on the device you will most likely need a keyboard, mouse and a monitor.
- A USB key that's at least 8GB in size and that can have all information removed from it

## Next steps

Now you have what you need to build a custom image. Lab 1a shows how to get started.

[Go to lab 1a](#)

# Lab 1a: Create a basic image

1/25/2019 • 4 minutes to read

To get started, we'll walk through installing a basic Windows 10 IoT Enterprise image onto a device. This ensures that everything's working before we move on to adding customizations. This lab is a prerequisite for completing all of the other labs in this section, as the other labs build on this first basic image.

## What you need to complete this series of labs

See [Get the tools you need to create an IoT Enterprise image](#) and make sure you have everything you need. Once you have everything, you can move on to the next section.

## Create your basic image

### Install Windows onto a reference device

This section covers how to create bootable USB Windows installation media, and then install it onto a device so it's ready to be customized.

#### Create a bootable USB key

The typical way to install Windows is to create a bootable USB key, and then copy the Windows installation files onto the USB key. Once you have the files on the USB key, you can insert it into the IoT device and boot from the USB key. See [Install Windows from a USB flash drive](#) to learn more.

Here's how to prepare the installation USB key:

1. Insert a USB key into your Technician PC
2. Open an Administrative Command Prompt and run diskpart:

```
diskpart
```

3. Use diskpart to list the disks so you can identify the USB drive:

```
list disk
```

You should see something like the following:

Disk #	Status	Size	Free	Dyn	Gpt
Disk 0	Online	238 GB	0 B		*
Disk 1	Online	3822 MB	0 B		

In this example Disk 1 is our USB drive, because the size represents the size of the USB drive that we're using.

4. When you've identified the disk number of your USB drive, use diskpart to prepare the drive so you can use it as a bootable installation drive:

## WARNING

The following commands will erase everything on the USB drive.

Enter the following commands from within Diskpart, where Disk 1 is the USB drive:

```
Select disk 1
clean
create partition primary
select partition 1
active
Format fs=fat32 quick
assign
exit
```

5. Copy the entire contents of the Windows 10 IoT Enterprise 2019 LTSC ISO or DVD onto the root of the USB drive. You can use File explorer to manually copy the files.

### Install Windows from the USB drive onto your IoT device and boot into Audit mode

This section covers how to use the Windows installation USB key to install Windows onto your IoT device using Windows Setup.

We recommend not having your device connected to any network during Windows Setup. Network connectivity could cause it to come out of the deferred activation state.

#### Boot the device to Windows Setup

1. Move the USB key from the Technician PC to the powered down IoT device.
2. Turn on your IoT device and enter the device's boot menu. Your device will have a specific button combination or keyboard key to press to get to the boot menu. If you don't know how to get to your device's boot menu, consult your hardware documentation.
3. From the boot menu select the USB key to boot from. Your device will boot from the USB key and enter into the Windows Setup.

#### Install Windows with Windows Setup

1. Step through the Windows Setup menus, providing the requested information. Choose the settings, such as language, time and currency, and keyboard options that apply to your device and click to the next screen.
2. On the "Install now" screen click on **Install now**.
3. On the Activate Windows screen insert a valid product key. If you don't have a valid product key select "I don't have a product key".
4. On the Application notices and license terms screen, if the terms are acceptable check the checkbox that you accept the license terms and then click **Next**.
5. On the "Which type of installation do you want" screen, select **Custom: Install Windows only**. This starts a clean installation
6. In the "Where do you want to install Windows?" screen, if the device has existing partitions, we recommend deleting the partitions so you have a single block on unallocated space to start from, then click **Next** to start the installation.
7. While the Operating System is being installed the device may restart a couple of time. Wait until the IoT device has entered OOBE (Out Of Box Experience) and is showing a screen that says "Let's start with region. Is this right?".

#### Enter Audit Mode

1. From the first OOBE screen, use the **CTRL+SHIFT+F3** combination on your keyboard to enter Audit mode.
2. Your device should restart in Audit mode. You'll know you're in Audit mode when you see a System Preparation Tool window. click **Cancel** on the System Preparation Tool to close it.
3. Every time you reboot the system you will see the System Preparation Tool, also called Sysprep. In a future lab

you will be invoking the Sysprep utility to get the system out of Audit mode. Refer to lab 1b for more information on Audit mode.

## Next steps

Windows is installed on your reference device and you have a basic image that's ready to be customized. In lab 1b, we'll take the basic image and use Audit mode to customize it.

[Go to lab 1b](#)

# Lab 1b: Customize the reference device in Audit Mode

1/25/2019 • 8 minutes to read

In [Lab 1a](#), you installed Windows 10 IoT Enterprise onto an IoT device and booted into audit mode. In this lab, we'll show you how to customize your device from audit mode.

## TIP

Most customizations in this lab can be made to an offline mounted Windows image, as well as in Audit mode. See [Modify a Windows image using DISM](#) for more information.

## What is Audit Mode?

[Audit Mode](#) allows you to customize Windows to image capture. Common audit mode customizations include installing Features on Demand (FODs), Drivers, Language Packs, and OEM software. This lab describes how to complete some of these common audit mode customizations.

Audit mode isn't necessarily the only way to implement these customizations. If the examples below do not fit into your workflow, explore the desktop deployment documentation for other alternatives.

For a fully automated approach to these steps consider using the [Windows 10 IoT Enterprise deployment framework](#).

## Prerequisites

The image that you created in [Lab 1a: Create a basic image](#) installed on an IoT device.

## Customize the device

The steps in this lab are optional. Most OEM devices will require at least one of the customizations in this lab, but if you don't see the need to make any audit mode customizations, you don't have to.

This section covers how to add:

- [Features on Demand](#)
- [Device drivers](#)
- [Languages](#)
- [Updates](#)
- [OEM software](#)

### Add a Feature on Demand (FOD) in Audit Mode

Features on Demand (FODs) are Windows feature packages that can be added at any time. Common features include language resources like handwriting recognition or other features like the .NET Framework (.NetFx3).

Device partners will often include FODs in Windows images. A commonly added feature is .NET Framework 3.5 to support scenarios where the device is running an OEM application and that needs .NET Framework 3.5 support.

To add a Feature on Demand in audit mode, you'll need the FOD ISO either on a USB drive, or copied to your IoT

device. Once you've finished installing FODs, you can remove the ISO from your IoT device or remove the USB drive.

1. Mount the Feature on Demand (FOD) ISO on the Technician PC.
2. Locate the cab file for the FOD you're going to install. In this example we'll use .NET Framework 3.5. The cab is named `Microsoft-Windows-NetFx3-OnDemand-Package~31bf3856ad364e35~amd64~~.cab`. You can view all the FOD .cab names at [Available Features on Demand](#).
3. Copy the cab file to the IoT device in a folder called C:\FoD.
4. Add the FOD. From an Administrative Command Prompt:

```
DISM.exe /online /add-package /packagepath:C:\FoD\Microsoft-Windows-NetFx3-OnDemand-
Package~31bf3856ad364e35~amd64~~.cab
```

5. Verify that the FOD is part of the image:

```
DISM.exe /online /get-capabilities /format:table
```

The output indicates the installation status for all FODs. Verify that the FODs you installed show as **Installed**.

Capability Identity	State
...	...
NetFX3~~~	Installed
...	...

See [Features on Demand](#) to learn more about Features on Demand, including how to add them to an offline mounted image.

### Install drivers in Audit Mode

Device partners may need to install additional drivers for Windows in order to support the hardware of the IoT device. There are numerous ways to install drivers. Two options below show how to do a simple installation using the driver vendors supplied setup package and an advanced method to add the driver using DISM.

To add a driver, you'll have to have a driver supplied from a hardware vendor. The driver package could be distributed as an .msi, .exe, or .inf file. The process of adding a driver depends on how the driver is distributed.

#### Simple method - manual installation

Use this method if the driver supplied by the independent hardware vendor (IHV) is a simple MSI or EXE package. If you want auto driver installation, you can use unattend files or through scripting. The steps below outline a simple installation.

1. Gather the driver installer package provided by the IHV. This is often an installation MSI or EXE package.
2. Copy the package to a temporary location on the IoT device. In audit mode, the system is logged in locally as the local Administrator account. Run the installation MSI or EXE and follow the prompts.
3. **Optional** Remove the installation package from the temporary location.

#### Advanced method

To use this method, the driver supplied by the IHV has to already be extracted out into INF, SYS, CAT, etc. files, or be an MSI or EXE package that can be extracted. This method can also be used to [add drivers to an offline mounted image](#).

1. If the driver is distributed as an MSI or EXE, copy the driver package provided by the IHV into a folder on the IoT device (we'll use C:\Drivers in our example). If the driver package is an .msi or .exe, extract the contents into a folder.
2. Open an Administrative Command Prompt and use DISM to add all the drivers in the folder.

```
Dism /online /add-driver /driver:C:\Drivers /recurse
```

The `/recurse` option will add all the drivers located in the C:\Drivers folder and its subfolders.

3. Reboot the device if prompted. When the PC reboots, make sure it reboots into audit mode.

### Add a language in Audit Mode

Device partners may need to add additional languages to an image to enable a user to change languages. This is particularly important for devices that may not have a persistent internet connection to download and install a language with the Settings app.

You can add additional languages to your custom image by using DISM to install a language pack and the related Features on Demand. You can add languages in audit mode or to an offline mounted image. See [Languages overview](#) for more information.

1. Mount the Feature on Demand ISO on your Technician PC. This might still be mounted if you added a FOD earlier in the lab.
2. Mount the Language Pack ISO on your Technician PC.
3. Add a language pack to your image. In this example, we'll use French (fr-FR). From an Administrative Command Prompt:

```
Dism /Add-Package /online /packagepath:"E:\x64\langpacks\Microsoft-Windows-Client-Language-Pack_x64_fr-fr.cab
```

Where E: is the mounted Language Pack ISO

4. Install the language FODs for your language pack.

```
DISM /online /add-package /packagepath:D:Microsoft-Windows-LanguageFeatures-Basic-fr-fr-Package~31bf3856ad364e35~amd64~~.cab /packagepath:D:Microsoft-Windows-LanguageFeatures-OCR-fr-fr-Package~31bf3856ad364e35~amd64~~.cab /packagepath:D:Microsoft-Windows-LanguageFeatures-Handwriting-fr-fr-Package~31bf3856ad364e35~amd64~~.cab /packagepath:D:Microsoft-Windows-LanguageFeatures-TextToSpeech-fr-fr-Package~31bf3856ad364e35~amd64~~.cab /packagepath:D:Microsoft-Windows-LanguageFeatures-Speech-fr-fr-Package~31bf3856ad364e35~amd64~~.cab
```

Where D: is the mounted FOD ISO

### Add a cumulative update in Audit Mode

Device partners may need to update the OS image with the latest cumulative update (LCU) as part of the initial image build process. The update can be applied offline using DISM or online using DISM or running the MSU package directly. Two options below show how to do a simple installation using the MSU or an advanced installation using DISM.

To add an update, you first download the most recent LCU from the Microsoft Update Catalog: <https://www.catalog.update.microsoft.com/Home.aspx>, and then install it. You can install the update through the GUI or the command line.

Below, we'll show you how to install an LCU using an .msu from the Microsoft Update catalog.

## Download an update

These steps can be performed on the Technician PC if the IoT device does not have internet connectivity, or if the device scenario requires never connecting to the internet.

1. Visit [Windows 10 Update History](#) to see which updates are available for your Windows image.
2. In the upper left of the page select your Windows 10 build. Click on, for example, Windows 10, version 1809.
3. In the left-hand navigation, you'll see a section called **In this release**. This section shows the most recent LCU's KB number. Click on the latest KB name, which will take you to a KB article with some information about the release.
4. On the KB article page, locate the link for the Microsoft Update Catalog and click the link. This will take you to the download page in the catalog.
5. Download the MSU package from the catalog and save it to C:\Packages on the IoT device.

## Install an update, simple method

After you've downloaded an update, double click the update in File Explorer to start the installation.

## Install an update, advanced method

You can install an LCU using DISM. This can be helpful if you're scripting the installation of the update. You can also use this method to add the update to an offline mounted image. See [Add updates to a Windows image](#) for more information.

Use DISM to install the LCU:

From an Administrative Command Prompt:

```
Dism /online /add-package /packagepath:C:\Packages\<package.msu>
```

## Install OEM software in Audit Mode

Device partners may need to install software in audit mode. This software might be Line of Business applications, tools, utilities, or any type of software that needs to be on the device prior to shipping. You can use Audit Mode to install software using methods that are available from the Windows desktop, and device partners should use the method that best aligns with their workflow. See

Some things to consider:

- If an installed application is to become the shell experience for the device, follow the steps in lab #5 to setup Shell Launcher or Assigned Access. The features used depend on the type of application that will become the shell.
  - Shell Launcher is used if a Win32 or .NET application will be used as the shell.
  - Assigned Access is used if a UWP application will be used as the shell.
- If the device experience is more like a customized desktop experience (for example, a hotel kiosk) where users are able to have access to the desktop, there are customization steps that can make it easier to ensure your device layout is preserved. For example, icon layout on the desktop and start menu can be preserved as part of the Sysprep process.
- This type of installation has to be done in audit mode, and can't be done on an offline mounted image.

## Next steps

With your image customized in audit mode, you can further customize the device experience. Lab 2 covers how to enable device lockdown features.

[Go to lab 2](#)

# Lab 2: Device Lockdown Features

1/25/2019 • 5 minutes to read

In labs [1a](#) and [1b](#) we installed the OS onto a reference device and made customizations in audit mode. This lab describes several ways lock down your device using device lockdown features that are built in to Windows. The device lockdown features aren't listed in any particular order, and you can enable some, all, or none of the features, depending on the device you're building.

## NOTE

This lab is optional. You can build an IoT Enterprise device without enabling any of the features described in this lab. If you aren't implementing any of these features, you can continue to [Lab 3](#).

For a fully automated approach to these steps consider using the [Windows 10 IoT Enterprise deployment framework](#).

## Prerequisites

Complete Lab 1a: Create a basic image.

## Keyboard filter

### Keyboard filter overview

The [Keyboard Filter](#) enables controls that you can use to suppress undesirable key presses or key combinations. Normally, a customer can alter the operation of a device by using certain key combinations like Ctrl+Alt+Delete, Ctrl+Shift+Tab, Alt+F4, etc. The Keyboard filter will prevent users from using these key combinations, which is helpful if your device is intended for a dedicated purpose.

The Keyboard Filter feature works with physical keyboards, the Windows on-screen keyboard, and the touch keyboard. Keyboard Filter also detects dynamic layout changes, such as switching from one language set to another, and continues to suppress keys correctly, even if the location of suppressed keys has changed on the keyboard layout.

Keyboard filter keys are stored in the Registry at **HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows Embedded\KeyboardFilter**.

### Enable the Keyboard filter

There are several methods to enable the Keyboard Filter, we are providing instructions for one of those methods in this lab.

## NOTE

See [Keyboard Filter](#) for more information.

1. Enable the Keyboard Filter feature by running the following command from an Administrative Command Prompt:

```
DISM /online /enable-feature /featurename:Client-DeviceLockdown /featurename:Client-KeyboardFilter
```

2. You'll be prompted to restart the reference device, type **Y** to reboot. The device will reboot into audit mode.

Once you've enabled the keyboard filter, see [Keyboard filter PowerShell script samples](#) to learn about blocking key combinations.

3. For this lab we are going to provide a demo on blocking the CTRL+ALT+DEL key. In an administrative PowerShell command window copy and paste the below commands.

```
$key = "Ctrl+Alt+Del"
$setkey = Get-WMIObject -class WEKF_PredefinedKey -computer localhost -namespace
root\standardcimv2\embedded | where {$_.Id -eq "$key"};
$setkey.Id = $key
$setkey.Enabled = 1;
$setkey.Put() | Out-Null;
```

4. Restart the reference device and then note the CTRL+ALT+DEL key is blocked.

## Unified Write Filter (UWF)

### Unified Write Filter overview

[Unified Write Filter \(UWF\)](#) is a Windows 10 device lockdown feature that helps to protect your device's configuration by intercepting and redirecting any writes to the drive (app installations, settings changes, saved data) to a virtual overlay. This overlay can be deleted by rebooting or, in certain configurations, the overlay can be retained until the Unified Write Filter is disabled.

### Enable the UWF

1. Enable the Unified Write Filter feature by running the following command from an Administrative Command Prompt:

```
DISM /online /enable-feature /featureName:Client-DeviceLockdown /featureName:Client-UnifiedWriteFilter
```

2. Restart the reference device
3. Configuring and enabling the overlay and protection is best done through scripting but for this lab we will configure using command line

See [Unified write filter](#) for more information about the UWF, including sample scripts.

4. At an Adminstrative Command prompt run the following commands

```
ufwmgr volume protect c:
ufwmgr filter enable
```

5. Restart the reference device
6. Now all writes will be redirected to the RAM overlay and will not be retained when the reference device is rebooted.
7. To disable the Unified Write Filter, at an Administrative Command prompt run the following command and then reboot the device.

```
ufwmgr filter disable
```

#### **NOTE**

When using the Unified Write Filter you must take into consideration the Operating System product activation. Product activation must be done with the Unified Write Filter disabled. Also, when cloning the image to other devices the image needs to be in a Sysprep state and the filter disabled prior to capturing the image.

## Unbranded boot

### **Unbranded boot overview**

**Unbranded boot** allows you to suppress Windows elements that appear when Windows starts or resumes, and can suppress the crash screen when Windows encounters an error that it cannot recover from.

### **Enable Unbranded boot**

1. Enable the Unbranded boot feature by running the following command in an Administrative Command Prompt:

```
DISM /online /enable-Feature /featureName:Client-DeviceLockdown  
DISM /online /Enable-Feature /FeatureName:Client-EmbeddedBootExp
```

2. Restart the reference device

### **Configure Unbranded Boot settings at runtime using BCDEdit**

You can customize Unbranded boot from an Administrative Command prompt in the following ways:

- Disable the F8 key during startup to prevent access to the Advanced startup options menu:

```
bcdedit.exe -set {globalsettings} advancedoptions false
```

- Disable the F10 key during startup to prevent access to the Advanced startup options menu:

```
bcdedit.exe -set {globalsettings} optionsedit false
```

- Suppress all Windows UI elements (logo, status indicator, and status message) during startup:

```
bcdedit.exe -set {globalsettings} bootuxdisabled on
```

#### **NOTE**

Anytime you rebuild the BCD information, for example using bcdboot, you'll have to re-run the above commands.

## Custom Logon

You can use the [Custom Logon](#) feature to suppress Windows 10 UI elements that relate to the Welcome screen and shutdown screen. For example, you can suppress all elements of the Welcome screen UI and provide a custom logon UI. You can also suppress the Blocked Shutdown Resolver (BSDR) screen and automatically end applications while the OS waits for applications to close before a shutdown.

See [Custom logon](#) for more information.

#### **NOTE**

Custom Logon feature will not work on images that are using a blank or evaluation product key. You must use a valid Product Key to see the changes made with the below commands.

1. Enable the Custom Logon feature by running the following command at an Administrative Command Prompt:

```
DISM /online /enable-feature /featurename:Client-DeviceLockdown /featurename:Client-EmbeddedLogon
```

2. If prompted to restart choose No.
3. Next at an Administrative Command prompt modify the following registry entries. If prompted to overwrite choose Yes.

```
Reg add "HKLM\SOFTWARE\Microsoft\Windows Embedded\EmbeddedLogon" /v BrandingNeutral /t REG_DWORD /d 1  
Reg add "HKLM\SOFTWARE\Microsoft\Windows Embedded\EmbeddedLogon" /v HideAutoLogonUI /t REG_DWORD /d 1  
Reg add "HKLM\SOFTWARE\Microsoft\Windows Embedded\EmbeddedLogon" /v HideFirstLogonAnimation /t  
REG_DWORD /d 1  
Reg add "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonU" /v AnimationDisabled /t  
REG_DWORD /d 1  
Reg add "HKLM\SOFTWARE\Policies\Microsoft\Windows\Personalization" /v NoLockScreen /t REG_DWORD /d 1  
Reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v UIVerbosityLeve /t REG_DWORD  
/d 1
```

4. Restart the reference device. You should no longer see the Windows UI elements that relate to the Welcome screen and shutdown screen.

## Next steps

Your device now has device lockdown features in place. You can use group policies to further customize your device's user experience. Lab 3 covers how to conigure policy settings.

[Go to lab 3](#)

# Lab 3: Configure policy settings on IoT Devices

1/25/2019 • 11 minutes to read

In lab 2 we enabled device lockdown features on our custom image. In addition to the Windows IoT Enterprise lockdown features, device partners can use a mix of Group Policies and feature customizations to achieve the desired user experience.

In this lab, we'll recommend some common configuration settings that IoT device partners tend to use. Consider whether each individual configuration setting applies to your device scenario.

## Control Windows Updates

One of the most common requests from device partners is centered around controlling automatic updates on Windows 10 IoT devices. The nature of IoT devices is such that unexpected disruptions, through something like an unplanned update, can create a bad device experience. Questions that you should ask when considering how to control Windows updates:

- Is the device scenario such that any disruption of the workflow is unacceptable?
- How are updates validated prior to deployment?
- What is the update user experience on the device itself?

If you have a device where disruption of the user experience isn't acceptable, you should consider limiting updates to only certain hours, disabling automatic updates, or deploying updates either manually or through a controlled 3rd party solution.

### Limit reboots from updates

You can use the Active Hours Group Policy, MDM, or registry setting to limit updates to only certain hours.

1. Open the Group Policy Editor (gpedit.msc) and navigate to **Computer Configuration\Administrative Templates\Windows Components\Windows Update** and open the **Turn off auto-restart for updates during active hours** policy setting. **Enable** the policy so you can set the start and end times for active hours.
2. Set the **Start** and **End** time to the Active Hours window. For example, set Active Hours to start at 2:00AM and end 4:00AM. This allows the system to reboot from updates between the hours of 2:00 AM and 4:00 AM.

### Control UI notifications from the Windows Update client

A device can be configured in a way to hide the UI experience for Windows Update while letting the service itself run in the background and update the system. The Windows Update client still honors the policies set for configuring Automatic Updates, this policy controls the UI portion of that experience.

1. Open the Group Policy Editor (gpedit.msc) and navigate to **Computer Configuration\Administrative Templates\Windows Components\Windows Update\Display options for update notifications**
2. Set the policy to enabled.
3. Set **Specify the update notifications display options** to 1 or 2.

#### NOTE

Set the value to 1 to hide all notifications except restart warnings, or to 2 to hide all notifications, including restart warnings.

### Completely disable automatic Windows Updates

Security and stability are at the core of a successful IoT project, and Windows Update provides updates to ensure

Windows 10 IoT Enterprise has the latest applicable security and stability updates. You might, however, have a device scenario where updating Windows has to be handled completely manually. For this type of scenario, we recommend disabling automatic updating through Windows Update. In previous versions of Windows device partners could stop and disable the Windows Update service, but this is no longer the supported method for disabling automatic updates. Windows 10 has a number of policies that allow you to configure Windows Updates in several ways.

To completely disable automatic updating of Windows 10 with Windows Update:

1. Open the Group Policy Editor (gpedit.msc) and navigate to **Computer Configuration\Administrative Templates\Windows Components\Windows update\Configure Automatic Updates**.
2. Explicitly set the policy to **Disabled**. When this setting is set to Disabled, any available updates from Windows Update must be downloaded and installed manually, which you can do in the Settings app under **Update & security > Windows Update**.

#### Disable access to the Windows Update user experience

In some scenarios, configuring Automatic Updates isn't enough to preserve a desired device experience. For example, an end user may still have access to the Windows Update settings, which would allow manual updates via Windows Update. You can configure Group policy to prohibit access to Windows Update through settings.

To prohibit access to Windows update:

1. Open the Group Policy Editor (gpedit.msc) and navigate to **Computer Configuration\Administrative Templates\Windows Components\Windows update\Remove access to use all Windows update features**.
2. Set this policy to **Enabled** to prevent the "Check for updates" option for users. Note: Any background update scans, downloads, and installations will continue to work as configured. This policy simply prevents the user from accessing the manual check through settings. Use the steps in the [previous section](#) to also disable scans, downloads and installations.

#### IMPORTANT

Be sure to have a well-designed servicing strategy for your device. Disabling Windows Update capabilities leaves the device in a vulnerable state if your device isn't getting updates in another way.

#### Prevent drivers from being installed via Windows Update

Sometimes drivers installed from Windows Update can cause issues with a device experience. The steps below prohibit Windows Update from downloading and installing new drivers on the device.

1. Open the Group Policy Editor (gpedit.msc) and navigate to **Computer Configuration\Administrative Templates\Windows Components\Windows update\Do not include drivers with Windows Updates**.
2. **Enable** this policy, which tells Windows to not include drivers with Windows quality updates.

#### Windows Update Summary

You can configure Windows Update in several ways, and not all policies are applicable to all devices. As a general rule, IoT devices require special attention to the servicing and management strategy to be used on the devices. If your servicing strategy is to disable all Windows Update features through policy, the steps below provide a combined list of policies to configure.

1. Open the Group Policy Editor (gpedit.msc) and navigate to **Computer Configuration -> Administrative Templates -> System -> Device Installation** and set the following policies:
  - a. **Specify the search server for device driver updates** to **Enabled**, with **Select update server** set to **Search Managed Server**
  - b. **Specify search order for device driver source locations** to **Enabled**, with **Select search order** set

- to **Do not search Windows Update**
2. In the Group Policy Editor, navigate to **Computer Configuration -> Administrative Templates -> Windows Components -> Windows Update** and set the following policies:
    - a. **Configure Automatic Updates** to **Disabled**
    - b. **Do not include drivers with Windows Updates** to **Enabled**
  3. In the Group Policy Editor, navigate to **Computer Configuration -> Administrative Templates -> System -> Internet Communication Management -> Internet Communication settings** and set **Turn off access to all Windows Update features** to **Enabled**
  4. In the Group Policy Editor, navigate to **Computer Configuration -> Administrative Templates -> Windows Components-> Windows Update -> Display options for update notifications** and set the policy to **Enabled** with **Specify the update notifications display options** to **2**

## Configure the system to hide blue screens

Bugchecks on the system (Blue Screen or BSOD) can happen for many reasons. For IoT devices it is important to hide these errors if they occur. The system can still collect a memory dump for debugging, but the user experience should avoid showing the bugcheck error screen itself. You can configure the system to replace "blue screen" with a blank screen for OS errors.

1. Open Registry Editor on the IoT device and navigate to  
`HKLM\SYSTEM\CurrentControlSet\Control\CrashControl`
2. Add a new registry value named `DisplayDisabled` as DWORD (32-bit) type with a value of 1.

## Configure notifications, toasts, and popups

IoT devices typically suppress common Windows dialogs that make sense in PC scenarios but could disrupt the user experience of an IoT device. The simplest way to disable unwanted dialogs is to use a custom shell using Shell Launcher or Assigned Access. If custom shell isn't the right choice, you can set a combination of policies, settings, and registry tweaks that can disable unwanted popups and notifications.

### Notifications

In some scenarios disabling individual notifications is beneficial. For example, if the device is a tablet device, the Battery Saver notification may be something the user should see, while other notifications such as OneDrive or Photos should be hidden. You might also decide that your device should suppress all notifications, regardless of the OS component that's providing them.

#### Hide all notifications

One method to disable notifications is to use Windows' Quiet Hours feature. Quiet Hours works similarly to features found on many smartphones that suppress notifications during certain hours, usually during the overnight hours. In Windows 10, Quiet Hours can be set to 24x7 so that notifications are never shown.

#### Enable 24x7 Quiet Hours:

1. Open the Group Policy Editor (gpedit.msc) and navigate to **User Configuration -> Administrative Templates -> Notifications**
2. Enable the policy for **Set the time Quiet Hours begins each day**, and set the value to **0**
3. Enable the policy for **Set the time Quiet Hours ends each day**, and set the value to **1439** (there are 1440 minutes per day)

#### NOTE

There are other policies in User Configuration -> Administrative Templates -> Notifications that allow you to get more granular on the exact notifications to disable. These options may be useful in some device scenarios.

### **Message Box Default Reply**

This is a registry change that will disable MessageBox class boxes from popping up, by having the system automatically click the default button on the dialog (usually OK or Cancel). This can be useful if 3rd party applications, which the device partner doesn't control, show MessageBox style dialogs. You can learn about this registry value at [Message Box Default Reply](#).

#### **Disable MessageBox class boxes**

1. Open the Registry Editor as administrator
2. Create a new Dword registry value under **HKLM\System\CurrentControlSet\Control\Error Message Instrument**, with a value named **EnableDefaultReply**
3. Set the data for the EnableDefaultReply value to 0
4. Test the scenario to ensure it is working as expected

## Security Baseline

Starting with the first release of Windows 10, an accompanying set of policies called the Security Baseline have been provided with each Windows release. A security baseline is a group of Microsoft-recommended configuration settings based on feedback from Microsoft security engineering teams, product groups, partners, and customers. The Security Baseline is a good way to quickly enable recommended security settings on IoT devices.

Note: Devices requiring certification such as STIG would benefit from using the Security Baseline as a starting point. The security baseline is delivered as part of the [Security Compliance Toolkit](#)

You can download the [Security Compliance Toolkit](#) from the Download Center.

1. Click **Download** on the link above. Select the Windows 10 Version xxxx Security Baseline.zip and the LGPO.zip. Be sure to choose the version that matches the version of Windows 10 you are deploying.
2. Extract the Windows 10 Version xxxx Security Baseline.zip file and the LGPO.zip file on the IoT device.
3. Copy LGPO.exe to the Local\_Script\Tools folder of the Windows 10 Version xxxx Security Baseline. LGPO is needed by the security baseline installation script but must be downloaded separately.
4. From an Administrative Command Prompt run:

```
Client_Install_NonDomainJoined.cmd
```

or, if the IoT device will be part of an Active Directory domain:

```
Client_Install_DomainJoined.cmd
```

5. Press Enter when prompted to run the script and then reboot the IoT device.

### **What you can expect**

Many settings are included as part of the security baseline. In the Documentation folder you'll find an Excel spreadsheet which outlines all the policies set by the baseline. You'll immediately notice that user account password complexity has been changed from its default, so you may need to update user account passwords on the system or as part of your deployment. Additionally, policies are configured for USB drive data access. Copying data from the system is protected by default now. Continue to explore the other settings added by the security baseline.

## Windows Defender

Anti-virus protection is required in many IoT device scenarios, especially devices that are more fully featured and

running an operating system like Windows 10 IoT Enterprise. For devices such as kiosks, retail POS, ATM, etc. Windows Defender is included and enabled by default as part of the Windows 10 IoT Enterprise installation. You may have a scenario where you want to modify the default Windows Defender user experience. For example, disabling notifications about scans performed, or even disabling scheduled deep scans in favor of only using real-time scanning. The policies below are useful for preventing unwanted UI to be created by Windows Defender.

1. Open the Group Policy Editor (gpedit.msc) and navigate to **Computer Configuration -> Administrative Templates -> Windows Components -> Windows Defender Antivirus -> Scan** and set:
  - a. **Check for the latest virus and spyware definitions before running a schedule scan** to **Disabled**
  - b. **Specify the maximum percentage of CPU utilization during a scan** to **5**
  - c. **Turn on catch up full scan** to **Disabled**
  - d. **Turn on catch up quick scan** to **Disabled**
  - e. **Create a system restore point** to **Disabled**
  - f. **Define the number of days after which a catch-up scan is forced** to **20** (this is a "just in case" setting" and shouldn't be needed if catch-up scans are enabled)
  - g. **Specify the scan type to use for a scheduled scan** to **Quick scan**
  - h. **Specify the day of the week to run a scheduled scan** to **0x8 (never)**
2. In Group Policy Editor navigate to **Computer Configuration -> Administrative Templates -> Windows Components -> Windows Defender Antivirus -> Signature Updates** and set:
  - a. **Define the number of days before spyware definitions are considered out of date** to **30**
  - b. **Define the number of days before virus definitions are considered out of date** to **30**
  - c. **Turn on scan after signature update** to **Disabled**
  - d. **Initiate definition update on startup** to **Disabled**
  - e. **Specify day of the week to check for definition updates** to **0x8 (never)**
  - f. **Define the number of days after which a catch-up definition update is required** to **30**

**Windows Components -> Windows Defender Antivirus** has additional policies. check each setting description to see if it applies to your IoT device.

## Next steps

Now that you've created an image that's tailored to your desired user experience, you can capture your image so it can be deployed to as many devices as you'd like. Lab 4 covers how to prepare an image for capture, and then deploy it to a device.

[Go to lab 4](#)

# Lab 4: Sysprep, capture, and deploy

1/25/2019 • 6 minutes to read

Now that the reference IoT device has been customized with software and settings, the system will be prepared for mass deployment using Sysprep and then captured to a WIM using DISM. This is the WIM image used during manufacturing to deploy to new systems.

## NOTE

The steps below use a combination of WinPE and DISM to complete the capture process. These tools are freely available from Microsoft. Some tools, like DISM, are included with all Windows installations. Many 3rd party tools also offer image capture and deployment that might work better for your deployment strategy. Choose the tool that is right for your device scenario.

For a fully automated approach to these steps consider using the [Windows 10 IoT Enterprise deployment framework](#).

## Prerequisites

- Complete at least [Lab 1a](#). This lab covers how to capture the image you created.
- The ADK with the WinPE add-on installed on your technician PC. See [Get the tools you need](#) for more information.

## Run Sysprep to complete the Audit process

In [Lab 1a](#) you put the system into audit mode, which is a special setup mode where a device maker can pre-install software and configure settings on a reference IoT device. When you've made your customizations in audit mode, you can capture an image of your customized reference device. While audit mode isn't specifically required, it does provide a scenario where the IoT device can be customized online prior to OOBE.

### Sysprep the reference IoT device

While booted into audit mode on the reference IoT device, run Sysprep from an Administrative Command Prompt to prepare the image for capture:

```
C:\Windows\System32\Sysprep\Sysprep.exe /generalize /oobe /shutdown
```

After Sysprep prepares the image, the reference device will shut down. The next time the device boots, it will boot into OOBE.

#### Caution

Don't power the reference IoT device back on until you're ready to capture an image. If the device boots, you'll have to go through the Sysprep process again.

## Capture your device image

### Create a WinPE USB drive

In this section, we'll show you how to create a bootable WinPE USB drive. You can use this USB key for both capturing and deploying your image.

The process outlined in the next steps can be scripted to make capturing and deploying images easier. For a fully automated scenario, see the [Windows 10 IoT Enterprise deployment framework](#)

1. Insert a USB drive into the Technician PC.

**NOTE**

Use a drive that's at least 16GB

2. On the technician PC, open the **Deployment and Imaging Tools Environment** as administrator. You can find a shortcut to the Deployment and Imaging Tools under Windows Kits in the Start Menu.
3. Copy the WinPE files to a working folder:

```
copype amd64 C:\WinPE
```

This command copies the 64bit WinPE files to C:\WinPE. Note: The destination folder will be created automatically.

4. Copy the WinPE files to your USB key.

```
makewinpemedia /ufd C:\WinPE D:
```

Where D: is the USB drive. This command will format the USB drive and erase any data that's on it.

5. Move the USB flash drive from the Technician PC to the reference IoT device.

### Boot the IoT reference device to WinPE and capture the Windows 10 IoT Enterprise OS image

The following steps will capture a WIM image from the reference IoT device's hard drive. This WIM can be used in development or in production. It's common to capture OS images during different stages of the development process. For example, the steps below could be followed to capture a base image of the OS with default apps installed. A later image could be captured with additional end customer apps installed.

1. Boot the reference IoT device from the bootable WinPE USB flash drive. The sequence to select which media to boot from is different from device to device. Consult the documentation for the IoT reference device in order to determine which key to press during boot in order to select the USB flash device as the boot target.

**IMPORTANT**

Don't boot your device until you know which key brings up the device's boot menu. The imaged IoT device is in a Sysprepped state and should not be allowed to boot back into Windows 10 IoT Enterprise.

The system will boot to the WinPE, where you'll see a Command prompt.

2. From the WinPE Command prompt, use DISM to capture an image of the Windows partition:

```
DISM /capture-image /imagefile:C:\IoTOS.wim /CaptureDir:C:\ /Name:"Windows 10 IoT Enterprise"
```

DISM will capture an image of the OS partition and store it on C: drive.

**NOTE**

Your device will have more than one partition, but you only need to capture the Windows partition. Refer to the deployment lab steps on how to re-create the system partition dynamically during deployment.

**Why are we capturing the image to the C:\ drive?**

- We temporarily capture the OS image to the C:\ drive because of the size of the captured image. Our USB drive is formatted as Fat32 which has a 4GB file size limit and the OS image is likely to be larger than 4GB. In the next step we'll split the image into smaller files so you can fit your entire image on a single FAT32-partitioned USB drive.

Windows supports multiple partitions on a flash drive which could allow you to have a drive with one Fat32 WinPE partition, and one NTFS partition where you could store a larger captured WIM file. Keep in mind, the default compression for DISM is fast, which speeds up the capture process however results in a larger WIM file.

If the image being captured is close to 4GB, try using DISM's `/Compress:max` option, which may remove the need to have the intermediate step of capturing the WIM to the OS partition and then using `/Split-image` to split it.

3. Split the WIM file captured to the C:\ drive into 4GB files and copy to the USB flash drive. From the WinPE command prompt:

```
Dism /Split-Image /ImageFile:C:\IoTOS.wim /SWMFile:D:\IoTOS.swm /FileSize:4000
```

This command splits the OS image into 4GB chunks.

Once the OS image is captured to the USB flash drive, it can be combined back into a single WIM on the Technician PC or kept as individual \*.swm files. In this lab, we'll use the .SWM files to deploy the image from the USB drive.

## Deploy the captured WIM image from WinPE

In this section, we show you how to deploy a WIM image from WinPE. The reference IoT device that we've been creating in these labs should already be in a deployed state; it's been captured in a Sysprepped state and, when deployed, will boot into OOBE. Use the steps below image a clean system. For this lab series, you can move on to [Lab 5](#) from here, since the reference IoT device will be powered on and OOBE completed.

### Use the WinPE USB drive to deploy to new systems

1. Boot the reference IoT device from the USB flash drive. The button or keypress sequence to select which media to boot differs from device to device. Consult your IoT device's documentation to determine which key to press during boot in order to select the USB flash device as the boot target.
2. Format the device. From the WinPE command prompt:

```
select disk 0
clean
convert gpt
create partition efi size=100
format quick fs=fat32 label="System"
assign letter="S"
create partition msr size=16
create partition primary
format quick fs=ntfs label="Windows"
assign letter="W"
```

**NOTE**

The above Diskpart commands don't create a recovery partition. If you need to configure a recovery partition, see [Configure UEFI/GPT-based hard drive partitions](#).

3. Deploy the WIM image to the W: drive created in the previous step. From the WinPE command prompt:

```
DISM /Apply-Image /ImageFile:D:\IoTOS.swm /SWMFile:IoTOS*.swm /ApplyDir:W:\ /Index:1 and press Enter
```

4. Configure the default BCD on the system. This is a required step as the disk was freshly partitioned and formatted which requires a new BCD. From the WinPE Command Prompt:

```
W:\Windows\System32\bcdboot W:\Windows /s S:
```

5. Reboot the system by typing wpeutil reboot at the WinPE Command Prompt. The IoT device will reboot into OOB.

## Next steps

Now that you've captured and deployed a Windows image, you can configure Windows to launch to a custom Shell. Lab 5 covers how to configure Shell launcher or Assigned Access.

[Go to lab 5](#)

# Lab 5: Configure Shell Launcher or Assigned Access

1/25/2019 • 3 minutes to read

Many IoT device scenarios require a custom user experience. This can be done by either automatically launching an application at Windows startup, or a custom shell experience. Using a custom shell experience enables the OEM to create a controlled user experience where the Windows UI is hidden and the OEM application is the focus.

Windows 10 IoT Enterprise has two custom shell features that enable this custom user experience.

- **Shell Launcher** enables OEMs to set a classic, non-UWP, app as the system shell. The advantage to using Shell Launcher is the ability to provide custom actions based on the exit code of the OEM application. For example if the OEM application exits with a specific exit code, the system can be configured to automatically restart the application, reboot or shutdown the device, etc.
- **Assigned Access** enables OEMs to set a UWP application as the system shell. Similar to Shell Launcher, Assigned Access can automatically restart the application when it's closed, keeping the device in the intended user experience.

For a fully automated approach to enabling these features, consider using the [Windows 10 IoT Enterprise deployment framework](#).

## Prerequisites

Complete [Lab 4](#): You should have a basic image that's been sysprepped and ready to be captured.

## Complete the OOB process on the IoT device

In lab 4 we used Sysprep to get the system ready for capture and deployment. The following steps assume you're using the image from lab 4. The steps will still work on system that hasn't been Sysprepped, but the OOB experience will already be completed.

### Complete the OOB process on the reference system

1. Turn on the reference IoT device and boot to the OS partition. The OS was in a Sysprep state and OOB should begin.
2. Complete the OOB experience. Choose the settings that match your device requirements.

#### NOTE

The OOB experience can be fully automated using an Answer File along with Sysprep to answer the OOB questions in advance. For more details refer to the example Answer Files in the [Windows 10 IoT Enterprise deployment framework](#).

## Enable and configure Shell Launcher

### Enable Shell Launcher

Once the device is booted to the desktop, enable the Shell Launcher. From an Administrative Command Prompt:

```
DISM /online /Enable-Feature /FeatureName:Client-EmbeddedShellLauncher
```

## Configure Shell Launcher to run an OEM application

With Shell Launcher enabled, you can set an application as the Windows Shell. Below, we'll show you how to use notepad.exe as the shell for the current user. In your device, you'll use a different application in place of Notepad.exe to configure the system to use the OEM application as the shell, but the steps are the same. See [Shell launcher](#) to learn more.

To set Notepad.exe as your custom shell:

1. From PowerShell run:

```
$ShellLauncherClass = [wmiclass]"\\localhost\root\standardcimv2\embedded:WESL_UserSetting"  
  
$ShellLauncherClass.SetDefaultShell("notepad.exe",1)  
  
$ShellLauncherClass.SetEnabled($TRUE)
```

2. Reboot the reference IoT device. When the system reboots the Notepad application will start as the default system shell.

## Enable and configure Assigned Access

The lab steps below provide links on how to install a UWP application suitable for Assigned Access and to configure the system to launch the application automatically at startup. The UWP application must be able to run above the lock screen in order to work correctly with assigned access.

### NOTE

See details on the UWP application requirements for Assigned Access at [Create a kiosk app for Assigned Access](#).

### Sideload a UWP application and configure Assigned Access to run it

In this lab, you'll add a UWP app to your image by sideloading it onto the system. For production scenarios follow the guidance on deploying signed UWP applications.

1. Compile the UWP application and build the APPX package following the steps at [Packaging UWP apps](#).
2. Sideload the UWP application following the steps at [Sideload your app package](#)
3. Follow the steps at [Set up a kiosk using Windows PowerShell](#) to complete the process.

### NOTE

For scenarios where multiple apps are needed, follow the steps at [Set up a multi-app kiosk](#)

# Boot and install Windows

1/18/2019 • 2 minutes to read

This section covers how to Boot a PC and install Windows.

## In this section

TOPIC	DESCRIPTION
<a href="#">Boot to WinPE</a>	How to create WinPE media, and boot a PC into WinPE
<a href="#">Boot to UEFI Mode or Legacy BIOS mode</a>	How to boot into UEFI or legacy BIOS mode
<a href="#">Windows Setup: Installing using the MBR or GPT partition style</a>	How to choose the partition style when installing Windows with Windows Setup
<a href="#">Boot from a DVD</a>	How to boot a PC to a DVD
<a href="#">Install Windows from a USB Flash Drive</a>	How to install Windows from a USB flash drive
<a href="#">Deploy a Custom Image</a>	Learn how to deploy a custom image
<a href="#">Deploy Windows with a VHD (Native Boot)</a>	Learn how to deploy a virtual hard drive that a PC can boot to
<a href="#">Windows Setup Installation Process</a>	How Windows Setup works

# Install Windows from a USB Flash Drive

1/18/2019 • 2 minutes to read

Here's how to create a bootable Windows installation USB drive starting with a Windows .iso file or a Windows Setup DVD.

## NOTE

For new Windows 10 installations, we've got a tool that does this for you. See [Download Windows 10](#).

## What you need

- Windows 10 install .iso or DVD
- USB flash drive with at least 5GB free space. This drive will be formatted, so make sure it doesn't have any important files on it.
- Technician PC - Windows PC that you'll use to format the USB flash drive
- Destination PC - A PC that you'll install Windows on

### Step 1 - Format the drive and set the primary partition as active

1. Connect the USB flash drive to your technician PC.
2. Open Disk Management: Right-click on **Start** and choose **Disk Management**.
3. Format the partition: Right-click the USB drive partition and choose **Format**. Select the **FAT32** file system to be able to boot either BIOS-based or UEFI-based PCs.
4. Set the partition as active: Right-click the USB drive partition and click **Mark Partition as Active**.

## NOTE

If **Mark Partition as Active** isn't available, you can instead use [diskpart](#) to select the partition and mark it active.

### Step 2 - Copy Windows Setup to the USB flash drive

1. Use File Explorer to copy and paste the entire contents of the Windows product DVD or ISO to the USB flash drive.
2. Optional: add an unattend file to automate the installation process. For more information, see [Automate Windows Setup](#).

### Step 3 - Install Windows to the new PC

1. Connect the USB flash drive to a new PC.
2. Turn on the PC and press the key that opens the boot-device selection menu for the computer, such as the Esc/F10/F12 keys. Select the option that boots the PC from the USB flash drive.  
Windows Setup starts. Follow the instructions to install Windows.
3. Remove the USB flash drive.

### Troubleshooting: file copy fails

This can happen when the Windows image file is over the FAT32 file size limit of 4GB. When this happens:

1. Copy everything except the Windows image file (sources\install.wim) to the USB drive (either drag and drop, or use this command, where D: is the mounted ISO and E: is the USB flash drive.)

```
robocopy D: E: /s /max:3800000000
```

2. Split the Windows image file into smaller files, and put the smaller files onto the USB drive:

```
Dism /Split-Image /ImageFile:D:\sources\install.wim /SWMFile:E:\sources\install.swm /FileSize:3800
```

Note, Windows Setup automatically installs from this file, so long as you name it install.swm.

## Related topics

[Windows Setup Technical Reference](#)

# Boot from a DVD

1/18/2019 • 3 minutes to read

The simplest way to install Windows on new hardware is to start directly from the Windows product DVD by using an answer file that is named Autounattend.xml. This method provides flexibility when network access is not available or when you are building only a few computers. You can use this same method to build an initial image in an image-based deployment scenario, typically known as a *master installation*.

By using the answer file, you can automate all or parts of Windows Setup. You can create an answer file by using Windows System Image Manager (Windows SIM). For more information, see [Create or Open an Answer File](#).

## Prerequisites

To complete this walkthrough, you need the following:

- An answer file on removable media (CD or DVD-ROM) or a USB flash drive. The answer file must be named Autounattend.xml. The answer file must be located at the root of the media.
- A Windows product DVD.

### To install Windows from the Windows product DVD

1. Turn on the new computer.

**Note** This example assumes that the hard disk drive is blank.

2. Insert both the Windows product DVD and the removable media that contains your answer file into the new computer.

**Note** When you use a USB flash drive, insert the drive directly into the primary set of USB ports for the computer. For a desktop computer, this is typically in the back of the computer.

3. Restart the computer by pressing the CTRL+ALT+DEL keys. Windows Setup (Setup.exe) starts automatically.

By default, Windows Setup searches at the root of a drive and other locations, such as removable media, for an answer file that is named Autounattend.xml. This occurs even if you do not explicitly specify an answer file. For more information, see "Implicitly Searching for an Answer File" and "Implicit Answer File Search Order" in [Windows Setup Automation Overview](#).

4. After the Setup program is finished, validate that Windows applied all customizations, and then reseal the computer by using the **sysprep** command together with the **/generalize** option.

The Sysprep tool removes all system-specific information and resets the computer. The next time that the computer starts, your customers can accept the Microsoft Software License Terms and add user-specific information.

Optional: To automatically run the Sysprep tool after the installation, set the Microsoft-Windows-Deployment | Reseal component setting in your answer file (Autounattend.xml) as follows:

```
ForceShutdownNow = true, Mode =OOBE
```

Optional: To run the Sysprep tool manually from a running operating system, type the following at a command prompt:

```
c:\windows\system32\sysprep /oobe /shutdown
```

For more information, see [Sysprep \(System Preparation\) Overview](#).

## Next Steps

This walkthrough illustrates a basic unattended installation that requires no user input. You can manually add more customizations to the newly installed operating system. If this is a master installation or an installation that you will use for image deployment, shut down the computer. Then, capture an image of the installation by using the Deployment Image Servicing and Management (DISM) tool or any third-party imaging software.

**Important** You must run the **sysprep /generalize** command before you move a Windows image to a new computer by any method. These methods include imaging, hard disk duplication, and other methods. Moving or copying a Windows image to a different computer without running the **sysprep /generalize** command is not supported, even if the new computer has the same hardware configuration. Generalizing the image removes unique information from the Windows installation so that you can apply that image on different computers.

The next time that you boot the Windows image, the **specialize** configuration pass runs. During this configuration pass, many components perform actions that must occur when you boot a Windows image on a new computer. For more information, see [How Configuration Passes Work](#).

## Related topics

[Windows Setup Technical Reference](#)

[Use a Configuration Set with Windows Setup](#)

[Deploy a Custom Image](#)

[Boot Windows to Audit Mode or OOB](#)

[Add Device Drivers to Windows During Windows Setup](#)

[Add a Custom Script to Windows Setup](#)

# Boot to WinPE

1/18/2019 • 2 minutes to read

This section covers how to create WinPE media and boot a PC into WinPE.

## In this section

TOPIC	DESCRIPTION
<a href="#">Download WinPE (Windows PE)</a>	How to get WinPE
<a href="#">Create bootable WinPE media</a>	How to create bootable WinPE media
<a href="#">WinPE: Install on a Hard Drive (Flat Boot or Non-RAM)</a>	How to install WinPE on a hard drive

# Download WinPE (Windows PE)

1/18/2019 • 2 minutes to read

Before you can use [WinPE](#), you'll have to create a bootable WinPE USB flash drive, CD, DVD, or virtual hard drive.

The files you need to create WinPE media are included in the [Winpe Add-on](#) to the [Windows Assessment and Deployment Kit](#). To create WinPE media, you'll have to [install the ADK](#) with the **Deployment tools** option, and then install the WindowsPE addon kit.

## Download and Install the Windows ADK

### For the ADK for Windows 10, version 1809

To start working with WinPE, download and install both the Windows Assessment and Deployment Kit (ADK) and the WinPE Add-ons.

[Download the Windows ADK for Windows 10, version 1809](#)

During installation, select **Deployment Tools**.

[Download the Windows PE add-on for the ADK](#)

### For the ADK for Windows 10, version 1803 or earlier

In previous versions, WinPE is included in the [Windows ADK](#).

During installation, select the following features:

- **Deployment Tools**: includes the **Deployment and Imaging Tools Environment**.
- **Windows Preinstallation Environment**: includes the files used to install Windows PE.

## Next Steps - create a bootable USB, CD, or DVD:

After you've downloaded and installed the ADK, you can create bootable WinPE media.

- To learn how to create a bootable WinPE USB drive, see [WinPE: Create USB Bootable drive](#)
- To learn how to create a bootable WinPE CD, DVD, ISO, or VHD, see [WinPE: Create a Boot CD, DVD, ISO, or VHD](#)

## Related topics

[WinPE for Windows 10](#)

[WinPE: Mount and Customize](#)

# Create bootable WinPE media

1/18/2019 • 3 minutes to read

The Windows Assessment and Deployment Kit (ADK) includes the [CopyPE](#) and [MakeWinPEMedia](#) command line utilities. When run from the Deployment and Imaging Tools Environment, **CopyPE** creates a working set of WinPE files, that **MakeWinPEMedia** can use to create bootable WinPE media.

**MakeWinPEMedia** can create bootable WinPE USB drives, virtual hard disks, or ISOs that allow you to boot a VHD or burn to a DVD or CD.

**CopyPE** and **MakeWinPEMedia** are installed when you choose the **Deployment tools** and **Windows Preinstallation Environment** options when [installing the ADK](#). If you're using the ADK for Windows 10, version 1809, Windows PE is a separate add-on that you download and install after you install the ADK. You can [download the WinPE add-on from the Download Center](#).

## Step 1: Create working files

No matter what type of media you're going to create, the first thing to do is create a working set of WinPE files on your technician PC.

1. Start the **Deployment and Imaging Tools Environment** as an administrator.
2. Run **copype** to create a working copy of the Windows PE files. For more information about **copype**, see [Copype command line options](#).

```
copype amd64 C:\WinPE_amd64
```

## Step 2: Customize WinPE (Usually not needed)

Note, when you add more packages to WinPE, it slows WinPE performance and boot time. Only add additional packages when necessary.

### Common customizations

- **Add an update.** If you're going to be capturing an FFU at the end of the lab, apply [KB4048955](#) to your WinPE image. To learn more, see: [WinPE: mount and customize](#).
- **Add a video or network driver.** (WinPE includes generic video and network drivers, but in some cases, additional drivers are needed to show the screen or connect to the network.). To learn more, see [WinPE: Add drivers](#).
- **Add PowerShell scripting support.** To learn more, see [WinPE: Adding Windows PowerShell support to Windows PE](#). PowerShell scripts are not included in this lab.
- **Set the power scheme to high-performance.** Speeds deployment. Note, our sample deployment scripts already set this scheme automatically. See [WinPE: Mount and Customize: High Performance](#).
- **Optimize WinPE:** Recommended for devices with limited RAM and storage (for example, 1GB RAM/16GB storage). After you add drivers or other customizations to Windows PE, see [WinPE: Optimize and shrink the image](#) to help reduce the boot time.

## Step 3: Create bootable media

Now that you now have a set of working files, you can use **MakeWinPEMedia** to build bootable WinPE media.

### Create a bootable WinPE USB drive

1. Attach a USB drive to your technician PC.
2. Start the **Deployment and Imaging Tools Environment** as an administrator.
3. **Optional** You can format your USB key prior to running MakeWinPEMedia. MakeWinPEMedia will format your WinPE drive as FAT32. If you want to be able to store files larger than 4GB on your WinPE USB drive, you can create a multipartition USB drive that has an additional partition formatted as NTFS. See [Create a multipartition USB drive](#) for instructions.
4. Use **MakeWinPEMedia** with the `/UFD` option to format and install Windows PE to the USB flash drive, specifying the USB key's drive letter:

```
MakeWinPEMedia /UFD C:\WinPE_amd64 P:
```

#### WARNING

This command reformats the partition.

See [MakeWinPEMedia command line options](#) for all available options.

The bootable WinPE USB drive is ready. You can use it to [boot a PC into WinPE](#).

### Create a WinPE ISO, DVD, or CD

1. Use **MakeWinPEMedia** with the `/ISO` option to create an ISO file containing the Windows PE files:

```
MakeWinPEMedia /ISO C:\WinPE_amd64 C:\WinPE_amd64\WinPE_amd64.iso
```

2. **Optional** Burn a DVD or CD: In Windows Explorer, right-click the ISO file, and select **Burn disc image** > **Burn**, and follow the prompts.

### Create a WinPE VHD to use with Hyper-V

Even though you can create a bootable When running Windows PE in Hyper-V, consider using an ISO file format instead of a VHD, to enable quick setup of the virtual PC.

To install Windows PE to a VHD:

1. Create a virtual hard drive (.vhdx):

```
diskpart
create vdisk file="C:\WinPE.vhdx" maximum=1000
attach vdisk
create partition primary
assign letter=V
format fs=ntfs quick
exit
```

2. Prepare the drive by using **MakeWinPEMedia**:

```
MakeWinPEMedia /UFD C:\WinPE_amd64 V:
```

3. Detach the drive:

```
diskpart
select vdisk file="C:\WinPE.vhdx"
detach vdisk
exit
```

## Troubleshooting

1. If Windows PE doesn't appear, try the following workarounds, rebooting the PC each time:
  - To boot a PC that supports UEFI mode: In the firmware boot menus, try manually selecting the boot files: \EFI\BOOT\BOOTX64.EFI.
  - If your PC requires storage or video drivers to boot, try adding those same drivers to the Windows PE image. For more information, see [WinPE: Mount and Customize](#).
2. If the PC doesn't connect to network locations, see [WinPE Network Drivers: Initializing and adding drivers](#).

## Related topics

[WinPE for Windows 10](#)

[WinPE: Install on a Hard Drive \(Flat Boot or Non-RAM\)](#)

[WinPE: Mount and Customize](#)

[WinPE: Boot in UEFI or legacy BIOS mode](#)

[Windows Setup Supported Platforms and Cross-Platform Deployments](#)

# WinPE: Install on a hard drive (Flat boot or Non-RAM)

1/18/2019 • 3 minutes to read

Windows Preinstallation Environment (Windows PE) is a minimal operating system where you can prepare a PC for installation, deployment, and servicing of Windows. Here's how to download and install it to an internal or external hard drive.

These instructions show how to set up a basic Windows PE installation that runs from the drive. This can sometimes give you better performance than booting from memory, and can help you run Windows PE on PCs or virtual environments with low memory. This procedure is also known as a *non-RAMDISK boot*, or a *flat boot*.

**Note** When Windows PE is running from the drive, you must turn off the PC before disconnecting the drive to avoid losing your work.

## Install the Windows ADK

- Get the [Windows Assessment and Deployment Kit \(Windows ADK\) Technical Reference](#), including the Windows PE feature. Note that starting with Windows 10, version 1809, WinPE is a separate add-on that you'll have to [download and install](#) after you install the ADK.

## Create a working directory for Windows PE files

- Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
- From the **Deployment and Imaging Tools Environment**, create a working directory for the Windows PE files.

```
copyce amd64 C:\WinPE_amd64
```

## Install Windows PE to the media

- Use DiskPart to prepare the partitions.

**Note** The following commands prepare a USB hard drive that can boot on either a BIOS-based or UEFI-based PC.

On UEFI-based PCs, Windows PE requires a boot partition formatted using the FAT32 file format, which only supports file sizes up to 4 GB. In this example, we create a separate partition on the drive, formatted using NTFS, that can store Windows images and other large files. To learn more, see .

```
diskpart
list disk
select <disk number>
clean
rem === Create the Windows PE partition. ===
create partition primary size=2000
format quick fs=fat32 label="Windows PE"
assign letter=P
active
rem === Create a partition for images ===
create partition primary
format fs=ntfs quick label="Images"
assign letter=I
list vol
exit
```

where *<disk number>* is the listed number of the external USB hard drive.

2. Apply the Windows PE image to the hard drive.

```
dism /Apply-Image /ImageFile:"C:\WinPE_amd64\media\sources\boot.wim" /Index:1 /ApplyDir:P:\
```

3. Set up the boot files.

```
BCDboot P:\Windows /s P: /f ALL
```

**Note** Ignore any warning messages that say "Warning: Resume application not found."

## Boot to Windows PE

1. Connect the device (internal or external USB hard drive) into the PC you want to work on.
2. Turn on the PC, and use the boot menus to select the Windows PE drive. Typically this requires pressing a hardware button or a key, such as the Esc key.

**Note** For UEFI-based PCs, you might need to find an option to manually select the UEFI boot files, for example, USBDrive01\EFI\BOOT\BOOTX64.EFI.

Windows PE starts automatically. After the command window appears, the wpeinit command runs automatically. This might take a few minutes.

3. Once WinPE is booted, you can [identify the drive letters with a script](#) or with diskpart.

## Troubleshooting

1. If the PC does not boot, try the following steps in sequence, and try to boot the PC after each step:
  - a. For external USB drives, try inserting the drive into a different USB port. Avoid using USB hubs or cables, because they might not be detected during the boot sequence. Avoid USB 3.0 ports if the firmware does not contain native support for USB 3.0.
  - b. If your PC requires drivers to boot, such as storage drivers or video drivers, or if your driver requires changes to the registry, add the driver to the Windows PE image. For more info, see [WinPE: Mount and Customize](#).
  - c. Update the firmware of the PC to the latest version.
2. For tips on connecting to a network, see [WinPE Network Drivers: Initializing and adding drivers](#).

## **Running Windows Setup from Windows PE**

- See [Windows Setup Supported Platforms and Cross-Platform Deployments](#) for tips on installing Windows on UEFI PCs that support both UEFI and legacy BIOS firmware modes, and for using the 32-bit (x86) version of Windows PE to install a 64-bit version of Windows.

## Related topics

[WinPE for Windows 10](#)

[WinPE: Create USB Bootable drive](#)

[WinPE: Mount and Customize](#)

[WinPE: Boot in UEFI or legacy BIOS mode](#)

[Windows Setup Supported Platforms and Cross-Platform Deployments](#)

# Boot to UEFI Mode or legacy BIOS mode

1/24/2019 • 3 minutes to read

Choose UEFI or legacy BIOS modes when booting into Windows PE (WinPE) or Windows Setup. After Windows is installed, if you need to switch firmware modes, you may be able to use the [MBR2GPT](#) tool.

In general, [install Windows using the newer UEFI mode](#), as it includes more security features than the legacy BIOS mode. If you're booting from a network that only supports BIOS, you'll need to boot to legacy BIOS mode.

After Windows is installed, the device boots automatically using the same mode it was installed with.

## To boot to UEFI or BIOS:

1. Open the firmware menus. You can use any of these methods:

- Boot the PC, and press the manufacturer's key to open the menus. Common keys used: **Esc**, **Delete**, **F1**, **F2**, **F10**, **F11**, or **F12**. On tablets, common buttons are **Volume up or Volume down** ([find more common keys and buttons](#)). During startup, there's often a screen that mentions the key. If there's not one, or if the screen goes by too fast to see it, check your manufacturer's site.
- Or, if Windows is already installed, from either the Sign on screen or the Start menu, select **Power** (  ) > hold **Shift** while selecting **Restart**. Select **Troubleshoot** > **Advanced options** > **UEFI Firmware settings**.

2. From the firmware menus, boot to drive or network while in UEFI or BIOS mode:

On the boot device menu, select the command that identifies both the firmware mode and the device. For example, select **UEFI: USB Drive** or **BIOS: Network/LAN**.

You might see separate commands for the same device. For example, you might see **UEFI USB Drive** and **BIOS USB Drive**. Each command uses the same device and media, but boots the PC in a different firmware mode.

Some devices only support one mode (either UEFI or BIOS). Other devices will only allow you to boot to BIOS mode by manually disabling the UEFI security features. To disable the security features, go to **Security** > **Secure Boot** and disable the feature.

### NOTE

Some older PCs (Windows 7-era or earlier) support UEFI, but require you to browse to the boot file. From the firmware menus, look for the option: "Boot from file", then browse to \EFI\BOOT\BOOTX64.EFI on Windows PE or Windows Setup media.

## UEFI and BIOS modes in WinPE

### Detect if WinPE is booted into BIOS or UEFI Mode

Query the registry to determine which mode the device is in. You can do this from the command line:

```
reg query HKLM\System\CurrentControlSet\Control /v PEFirmwareType
```

RETURN CODE	FIRMWARE MODE
0x1	BIOS
0x2	UEFI

Use it in a script:

```
wpeutil UpdateBootInfo
for /f "tokens=2* delims=      " %%A in ('reg query HKLM\System\CurrentControlSet\Control /v PEFirmwareType')
DO SET Firmware=%&B
:: Note: delims is a TAB followed by a space.
if %Firmware%==0x1 echo The PC is booted in BIOS mode.
if %Firmware%==0x2 echo The PC is booted in UEFI mode.
```

Note that between `delims=` and `" %%A` is a tab, followed by a space.

## Make sure you boot into the right mode every time

Here are a couple of ways you can make sure you're booted into the right firmware mode every time you start your PC.

### Use preformatted hard drives, and use a method that doesn't automatically format the drive.

If you want to ensure that your drive boots into a certain mode, use drives that you've preformatted with the GPT file format for UEFI mode, or the MBR file format for BIOS mode. When the installation starts, if the PC is booted to the wrong mode, Windows installation will fail. To fix this, restart the PC in the correct firmware mode.

### Remove the UEFI or BIOS boot files

If you want a PC to only boot into a certain mode, you can remove the files that Windows PE or Windows Setup use to boot in UEFI or BIOS mode. Remove the following files, depending on the mode you want to boot to.

#### Boot only when in UEFI mode

Remove the **bootmgr** file from the root of the Windows PE or Windows Setup media. This prevents the device from starting in BIOS mode.

#### Boot only when in BIOS mode

Remove the **efi** folder from the root of the Windows PE or Windows Setup media. This prevents the device from starting in UEFI mode.

## Related topics

[WinPE: Create USB Bootable drive](#)

# Deploy a Custom Image

1/18/2019 • 4 minutes to read

In this topic you create a reference installation, capture an image of the installation, and rerun Windows Setup with an answer file that points to your custom image. Deploying a custom image using Windows Setup provides several benefits over applying an image using an image capture tool.

Setup supports the following:

- Applying another answer file for additional customizations during deployment.
- Reconfiguring disk configuration.
- Adding additional drivers.
- Replacing a product key.
- Selecting a different language to install.
- Selecting from a list of images to install, if your image file contains more than one image.
- Installing to a different drive location.
- Upgrading an existing Windows installation.
- Configuring the computer to dual-boot operating systems.
- Ensuring that the hardware can support Windows.

There are some limitations to installing a custom image using Windows Setup. For more information, see [Windows Setup Scenarios and Best Practices](#).

## Prerequisites

To complete this walkthrough, you need the following:

- A technician computer that has the Windows Assessment and Deployment Kit (Windows ADK) tools installed.
- A Windows product ISO.
- A master computer on which you will install and capture your custom image.
- Bootable Windows PE media. There are several types of Windows PE media that you can create. For more information about these options, see [WinPE for Windows 10](#).
- Access to a network share to store your custom image and Windows Setup source files.

## Step 1: Copy the Windows product DVD source files to a network share

On your technician computer, copy the entire content of the Windows product DVD to a network share. For example:

```
net use N: \\server\share\  
xcopy D: N:\WindowsDVD\ /s
```

where D: is the DVD-ROM drive on your local computer.

## Step 2: Create a master installation

1. Create a master installation by using one of the following methods:

- [Boot from a DVD](#)
- [Use a Configuration Set with Windows Setup](#)

2. After the installation is complete, shut down the computer.

## Step 3: Capture an image of the installation

In this step, you'll capture an image of the reference installation by using DISM and then store the custom image on a network share.

1. Boot the reference computer by using your bootable Windows PE media.
2. At a command prompt, capture an image of the installation. You specify a name and description as part of your image capture. All values are required by Windows Setup. If a .wim file does not include these values, then the image will not install correctly. For example:

```
Dism /Capture-Image /ImageFile:C:\myimage.wim /CaptureDir:c:\ /Compress:fast /CheckIntegrity  
/Name:"x86_Ultimate" /Description:"x86 Ultimate Compressed"
```

3. Replace the default Install.wim on the network share with your custom image. The image must be called Install.wim. For example:

```
net use N: \\server\share\  
copy C:\myimage.wim N:\WindowsDVD\sources\install.wim
```

If necessary, provide network credentials for appropriate network access.

For more information, see [DISM Image Management Command-Line Options](#).

## Step 4: Create a custom answer file

In this step, you will create an answer file that points to your custom image. This step assumes that you have already built an answer file and have a working catalog.

1. On your technician computer, open **Windows System Image Manager**.
2. On the **File** menu, click **New Answer File**.
3. In the **Windows Image** pane of Windows SIM, expand the **Components** node to display available settings.
4. Add the following components to your answer file by right-clicking the component and then selecting the appropriate configuration pass.

COMPONENT	CONFIGURATION PASS
<b>Microsoft-Windows-Setup\DiskConfiguration\Disk\CreatePartitions\CreatePartition</b>	<b>windowsPE</b>
<b>Microsoft-Windows-Setup\DiskConfiguration\Disk\ModifyPartitions\ModifyPartition</b>	<b>windowsPE</b>
<b>Microsoft-Windows-Setup\ImageInstall\OSImage\InstallTo</b>	<b>windowsPE</b>

#### Note

Expand the component list until you see the lowest setting listed in the previous table, and then add that setting to your answer file. This shortcut will add the setting and all parent settings to your answer file in one step.

5. All of the settings that you added must appear in the **Answer File** pane. Select and configure each setting as specified in the following table.

COMPONENT	VALUE
<b>Microsoft-Windows-Setup\DiskConfiguration</b>	<code>WillShowUI = OnError</code>
<b>Microsoft-Windows-Setup\DiskConfiguration\Disk</b>	<code>DiskID = 0</code> <code>WillWipeDisk = true</code>
<b>Microsoft-Windows-Setup\DiskConfiguration\Disk\CreatePartitions\CreatePartition</b>	<code>Extend = false</code> <code>Order = 1</code> <code>Size = 300</code>
	<code>Type = Primary</code>
<b>Microsoft-Windows-Setup\DiskConfiguration\Disk\CreatePartitions\CreatePartition</b>	<code>Extend = true</code> <code>Order = 2</code>
	<code>Type = Primary</code>

COMPONENT	VALUE
<b>Microsoft-Windows-Setup\DiskConfiguration\Disk\ModifyPartitions\ModifyPartition</b>	<pre>Active = true Extend = false Format = NTFS Label = System Letter = S Order = 1 PartitionID = 1</pre>
<b>Microsoft-Windows-Setup\DiskConfiguration\Disk\ModifyPartitions\ModifyPartition</b>	<pre>Extend = false Format = NTFS Label = Windows Letter = C Order = 2 PartitionID = 2</pre>
<b>Microsoft-Windows-Setup\Image\Install\OSImage&lt;/strong&gt;</b>	<pre>WillShowUI = OnError</pre>
<b>Microsoft-Windows-Setup\Image\Install\OSImage\InstallTo</b>	<pre>DiskID = 0 PartitionID = 2</pre>

6. In a command prompt window copy the answer file to a network location. For example:

```
net use N: \\server\share\
md N:\AnswerFiles
copy C:\deploy_unattend.xml N:\AnswerFiles\
```

If necessary, provide network credentials for appropriate network access.

## Step 5: Deploy the image by using Windows Setup

In this step, you will deploy your custom image from a network share onto a destination computer.

1. Boot the destination computer by using your bootable Windows PE media.
2. Connect to the network share that you specified in [Step 4: Create a custom answer file](#), and then run Setup with your answer file. For example:

```
net use N: \\server\share
N:\WindowsDVD\setup /unattend:N:\AnswerFiles\deploy_unattend.xml
```

If necessary, provide network credentials for appropriate network access.

## Next Steps

You can further customize your answer file to include additional options. You can also build a DVD deployment media that contains the same content that you put on the network share. A single deployment DVD provides a portable installation solution that requires no network or any additional resources. The process includes building a configuration set and recapturing all source files into a single DVD.

### IMPORTANT

The DVD media that you create is for internal deployment use only. You cannot redistribute this media.

## Related topics

[Windows Setup Technical Reference](#)

[Boot from a DVD](#)

[Use a Configuration Set with Windows Setup](#)

[Boot Windows to Audit Mode or OOBEmode](#)

[Add Device Drivers to Windows During Windows Setup](#)

[Add a Custom Script to Windows Setup](#)

# Deploy Windows with a VHDX (Native Boot)

1/18/2019 • 5 minutes to read

Native boot enables Windows 10 virtual hard disks (VHDXs) to run on a computer without a virtual machine or *hypervisor*. A hypervisor is a layer of software under the operating system that runs virtual machines. Native boot for Windows 10 requires the .vhdx format, not the .vhf format.

## What Is VHDX with Native Boot?

A virtual hard disk can be used as the running operating system on designated hardware without any other parent operating system, virtual machine, or hypervisor. Windows disk-management tools, the DiskPart tool and the Disk Management Microsoft Management Console (Diskmgmt.msc), can be used to create a VHDX file. A supported Windows image (.wim) file can be applied to a VHD, and the VHDX can be copied to multiple systems. The Windows boot manager can be configured to boot directly into the VHD.

The VHDX can also be connected to a virtual machine for use with the Hyper-V Role in Windows Server.

Native-boot VHDXs are not designed or intended to replace full image deployment on all client or server systems. Enterprise environments already managing and using .vhdx files for virtual machine deployment will get the most benefit from the native-boot VHDX capabilities. Using the .vhdx file as a common image container format for virtual machines and designated hardware simplifies image management and deployment in an enterprise environment.

For more information about virtualization in Windows, see [this Microsoft Web site](#). For more information about how to use VHDs with native boot, see [this Microsoft Web site](#).

## Common Scenarios

- Using disk-management tools to create and *attach* a VHDX for offline image management. You can attach a VHDX by using the **Attach vdisk** command which activates the VHDX so that it appears on the host as a disk drive instead of as a .vhf file.
- Mounting reference VHDX images on remote shares for image servicing.
- Maintaining and deploying a common reference VHDX image to execute in either virtual or physical computers.
- Configuring VHDX files for native boot without requiring a full parent installation.
- Configuring a computer to boot multiple local VHDX files that contain different application workloads, without requiring separate disk partitions.
- Using Windows Deployment Services (WDS) for network deployment of VHDX images to target computers for native boot.
- Managing desktop image deployment.

## Requirements

- The local disk must have at least two partitions: a system partition that contains the Windows boot-environment files and Boot Configuration Data (BCD) store, and a partition to store the VHDX file. The .vhf file format is supported for native boot on a computer with a Windows 7 boot environment, but you will have to update the system partition to a Windows 8 or Windows 10 environment to use the .vhdx file

format. For more information about how to add a Windows 8 or Windows 10 boot environment for native VHDX boot, see [Boot to VHDX \(Native Boot\): Add a Virtual Hard Disk to the Boot Menu](#).

- The local disk partition that contains the VHDX file must have enough free disk space for expanding a dynamic VHDX to its maximum size and for the page file created when booting the VHD. The page file is created outside the VHDX file, unlike with a virtual machine where the page file is contained inside the VHD.

## Benefits

- Using the same image-management tools for creating, deploying, and maintaining system images to be installed on designated hardware or on a virtual machine.
- Deploying an image on a virtual machine or a designated computer, depending on capacity planning and availability.
- Deploying Windows for multiple boot scenarios without requiring separate disk partitions.
- Deploying supported Windows images in a VHDX container file for faster deployment of reusable development and testing environments.
- Replacing VHDX images for server redeployment or recovery.

## Limitations

- Native VHDX disk management support can attach approximately 512 VHDX files concurrently.
- Native VHDX boot does not support hibernation of the system, although sleep mode is supported.
- VHDX files cannot be nested in other VHDX files.
- Native VHDX boot is not supported over Server Message Block (SMB) shares.
- Windows BitLocker Drive Encryption cannot be used to encrypt the host volume that contains VHDX files that are used for native VHDX boot, and BitLocker cannot be used on volumes that are contained inside a VHD.
- The parent partition of a VHDX file cannot be part of a volume snapshot.
- An attached VHDX can't be a *dynamic disk*. A dynamic disk provides features that basic disks do not, such as the ability to create volumes that span multiple disks (spanned and striped volumes), and the ability to create fault-tolerant volumes (mirrored and RAID-5 volumes). All volumes on dynamic disks are known as dynamic volumes.
- The parent volume of the VHDX cannot be configured as a dynamic disk. Store the VHDX on a basic disk.

## Types of Virtual Hard Disks

Three types of VHDX files can be created by using the disk-management tools:

- **Fixed size.** A fixed size VHDX is a file that is allocated to the size of the virtual disk. For example, if you create a virtual hard disk that is 2 gigabytes (GB) in size, the system will create a host file approximately 2 GB in size. Fixed hard-disk images are recommended for production servers and working with customer data. Recommended for production servers, to increase performance and help protect user data
- **Dynamically expanding.** A dynamically expanding VHDX is a file that is as large as the actual data written to it at any given time. As more data is written, the file dynamically increases in size. For example, the size of a file backing a virtual 2 GB hard disk is initially around 2 megabytes (MB) on the host file system. As data is written to this image, it grows with a maximum size of 2 GB.

Dynamically expanding VHDXs are recommended only for development and testing environments because they can be more easily corrupted by a catastrophic system shutdown, such as a power outage.

- **Differencing.** A differencing hard disk image describes a modification of a parent image. This type of hard-disk image is not independent; it depends on another hard-disk image to be fully functional. The parent hard disk image can be any of the mentioned hard-disk image types, including another differencing hard-disk image.

Differencing hard disk images are recommended only for development and testing environments.

## Related topics

[Deploy Windows with a VHD \(Native Boot\)](#)

# Windows Setup Installation Process

1/18/2019 • 2 minutes to read

Windows® Setup is the program that installs Windows or upgrades an existing Windows installation. It is also the basis for the following installation and upgrade methods:

- Interactive Setup
- Automated installation
- Windows Deployment Services

In this topic:

- [Windows Setup Installation Types](#)
- [Windows Setup Process](#)

## Windows Setup Installation Types

Windows Setup can perform both clean and upgrade installations. However, it does not perform computer-to-computer migrations. Instead, you must use Windows Easy Transfer, the User State Migration Tool (USMT), or another migration tool to move data from a previous installation to the new operating system.

- **Custom installations.** Windows Setup can perform a custom installation, also known as a clean installation, which saves your previous Windows installation but does not migrate your settings. The previous Windows installation will not boot after a clean installation.
- **Upgrade installations.** Windows Setup can perform an installation that retains your settings and preferences while upgrading your operating system.

## Windows Setup Process

The Windows Setup program starts and restarts the computer, gathers information, copies files, and creates or adjusts configuration settings. The following table shows the overall process for Windows Setup:

WINDOWS SETUP PHASE	SETUP ACTIONS
---------------------	---------------

WINDOWS SETUP PHASE	SETUP ACTIONS
<b>Downlevel</b> (for custom installations and upgrades) - or - <b>Windows PE</b> (for booting the Windows DVD or booting a custom Windows PE image)	<ol style="list-style-type: none"> <li>Specify Windows Setup configurations by using either the Windows Setup dialog boxes (interactive) or an answer file (unattended), or a combination of the two. Windows Setup configurations include adding a product key and configuring a disk.</li> <li>Apply answer file settings in the <a href="#">windowsPE</a> configuration pass to configure the installation behavior and user experience.</li> <li>Configure the disk.</li> <li>Copy the Windows image to the disk.</li> <li>Prepare boot information.</li> <li>Process answer file settings in the <a href="#">offlineServicing</a> configuration pass. The settings are applied to the Windows image before that Windows image boots. When the computer first boots, any optional components, drivers, updates, or language packs are processed.</li> </ol>
<b>Online configuration</b>	Create specific configurations, making the Windows installation unique.
<b>Windows Welcome</b>	<ol style="list-style-type: none"> <li>Apply answer file settings in the <a href="#">oobeSystem</a> configuration pass.</li> <li>Apply content file settings from the Oobe.xml file.</li> <li>Start Windows Welcome.</li> </ol>

## Related topics

[Windows Setup Technical Reference](#)

[Automate Windows Setup](#)

[Settings for Automating OOBE](#)

[Windows Setup Scenarios and Best Practices](#)

[Windows Setup Automation Overview](#)

[Audit Mode Overview](#)

[Windows Setup Configuration Passes](#)

[Windows Setup Supported Platforms and Cross-Platform Deployments](#)

# Install Windows 10 using a previous version of Windows PE

1/18/2019 • 2 minutes to read

To use some DISM features in WinPE, such as [siloed provisioning packages](#), you may run the latest version of DISM from a separate location.

Each time you boot WinPE and want to use these features, you'll need to install and configure the drivers needed for DISM, including the wimmount.sys and wofadk.sys drivers.

The CopyDandI.cmd script copies the version of DISM from your local installation of the ADK to a folder which you can use in WinPE.

## Option 1: Run DISM from a separate location

You'll need the Windows 10, version 1607 version of the Deployment and Imaging Tools from the ADK.

**Important** Don't overwrite the existing DISM files on the WinPE image.

1. Start the Deployment and Imaging Tools Environment as an administrator.
2. From the technician PC, copy the Deployment and Imaging Tools from the Windows ADK to the storage USB key.

```
CopyDandI.cmd amd64 E:\ADKTools\amd64
```

## Option 2: Add DISM to the WinPE RAMDisk.

Note: this will add roughly 4MB to the size of your DISM image, which may affect performance.

1. On your technician PC, install the Windows ADK for Windows 10.
2. Mount WinPE. For WinPE 3.x, mount the file: \sources\winpe.wim. For WinPE 4.x and 5.x, mount the file: \sources\boot.wim.

```
md "C:\WinPE_amd64\mount"

Dism /Mount-Image /ImageFile:"C:\WinPE_amd64\media\sources\boot.wim" /index:1
/MountDir:"C:\WinPE_amd64\mount"
```

3. Copy the DISM folder from the Windows ADK into a new folder in the mounted WinPE image.

```
md C:\WinPE_amd64\mount\DISM

robocopy "C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Deployment
Tools\amd64\DISM" C:\WinPE_amd64\mount\DISM
```

**Important** Don't overwrite the existing DISM files from the **system32** folder in the WinPE image. Instead, create a new folder on the host computer to copy the Windows ADK files into.

4. Unmount WinPE.

```
Dism /Unmount-Image /MountDir:"C:\WinPE_amd64\mount" /commit
```

5. Create WinPE bootable media, or replace the WinPE image file on your existing removable media.

```
MakeWinPEMedia /UFD C:\WinPE_amd64 F:
```

## Use the new version of DISM

1. Boot the reference PC to WinPE.
2. Find the drive letter of the storage drive (`diskpart, list volume, exit`).
3. Install and configure DISM's required drivers by using either **wimmountadksetupamd64.exe /Install** or **wimmountadksetupx86.exe /Install**.

```
W:\ADKTools\amd64\wimmountadksetupAmd64.exe /Install /q
```

For the default (RAMDisk) version of WinPE, you'll need to run this command each time you boot WinPE. To learn how to run this command automatically when WinPE boots, see [Wpeinit and Startnet.cmd: Using WinPE Startup Scripts](#).

4. Verify the new version of DISM:

```
W:\ADKTools\amd64\DISM.exe /?
```

The output shows the build number, for example:

```
Deployment Image Servicing and Management tool
Version: 10.0.14939.0
```

5. Use the new version of DISM. Example:

```
W:\ADKTools\amd64\DISM.exe /Apply-Image /ImageFile:install.wim /Index:1 /ApplyDir:W: /Compact
W:\ADKTools\amd64\DISM.exe /Apply-SiloedPackage /ImagePath:W:\ /PackagePath:"e:\SPPs\fabrikam-id.spp"
/PackagePath:"D:\SPPs\office16_base.spp" /PackagePath:"D:\SPPs\office16_fr-fr.spp"
/PackagePath:"D:\SPPs\office16_de-de.spp"
```

## Related topics

[DISM Supported Platforms](#)

[WinPE: Mount and Customize](#)

[Lab 10: Add desktop applications and settings with siloed provisioning packages \(SPPs\)](#)

# Windows Setup Automation Overview

2/21/2019 • 11 minutes to read

## Use Setupconfig.ini to install Windows

### What is a setupconfig file?

Setupconfig is a configuration file that is used to pass a set of flags or parameters to Windows setup.exe. Use this file as an alternative to passing parameters to Windows setup on a command line. This functionality is available in Windows 10, version 1511 and later.

IT pros can use the setupconfig file to add parameters to Windows Setup from Windows Update and Windows Server Update Services.

The different parameters that can be used with Windows 10 Setup.exe are described in this topic.

Setupconfig.ini files can contain single parameters, or parameters and value pairs. Do not include "/" characters, and with parameter and value pairs, include "=" between the two.

For example, you create a Setupconfig.ini with the following. Note that the header `[SetupConfig]` is required.

```
[SetupConfig]
NoReboot
ShowOobe=None
Telemetry=Enable
InstallDrivers=<path of folder containing INF and SYS drivers>
ReflectDrivers=<path of folder containing INF and SYS files for the encryption drivers>
```

This is equivalent to the following command line:

```
Setup /NoReboot /ShowOobe None /Telemetry Enable
```

### How does Windows Setup use Setupconfig.ini?

#### Using media/ISO file

If you are running Windows setup from media or an ISO file, you must include the location to the setupconfig file on the command line ("/`ConfigFile <path>`") when running setup.exe. For example:

```
Setup.exe /ConfigFile <path to Setupconfig.ini>
```

If you include a parameter on the command line and the same parameter in the setupconfig file, the setupconfig file parameter and value has precedence.

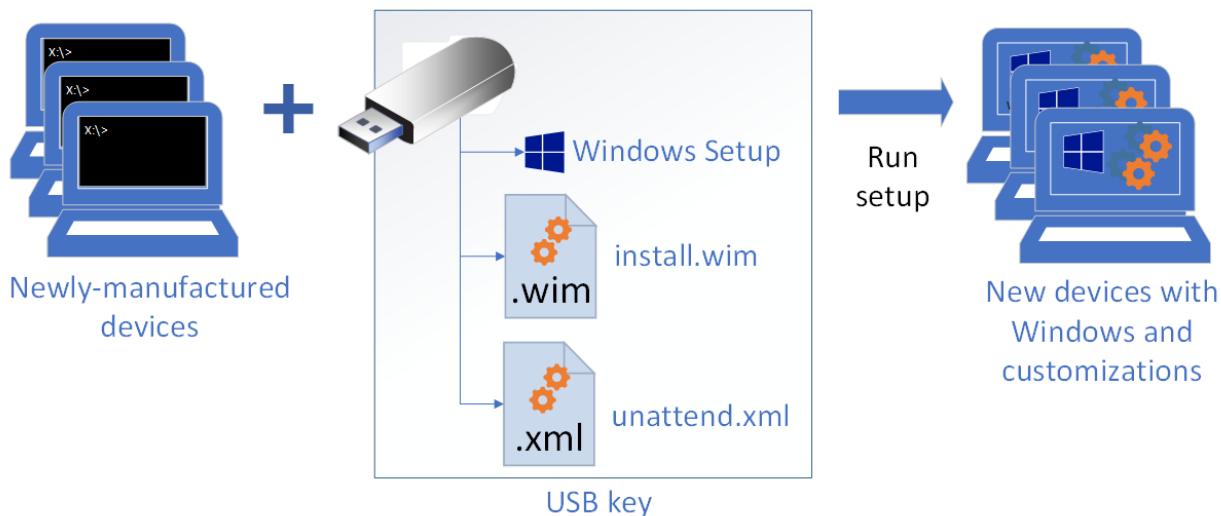
#### Using Windows Update

If the update is delivered through Windows Update, Windows Setup searches in a default location for a setupconfig file. You can include the setupconfig file here:

```
"%systemdrive%\Users\Default\AppData\Local\Microsoft\Windows\WSUS\SetupConfig.ini"
```

## Use an answer file while installing Windows

You can automate Windows installation by using an answer file:



### Use a USB flash drive

1. Use a sample answer file or create your own with Windows System Image Manager (Windows SIM).
2. Save the file as **Autounattend.xml** on the root of a USB flash drive.
3. On a new PC, put in the Windows product DVD and the USB flash drive, and then boot the PC. When no other answer file is selected, Windows Setup searches for this file.

### Select an answer file

- You can select a specific answer file during installation by booting to the Windows Preinstallation Environment, and using the **setup.exe** command with the **/unattend:filename** option. For more information, see [WinPE: Create USB Bootable drive](#).

For sample answer files and a list of settings used to automate installation, see [Automate Windows Setup](#).

## Modify an existing installation

Because reboots are required during Setup, a copy of the answer file is cached to the %WINDIR%\Panther directory of the Windows installation. You can modify this file to do any of the following:

- Update system and control panel settings without booting the image.
- Update an image by preparing the PC to boot to audit mode (see Microsoft-Windows-Deployment\Reseal\Mode).
- Update the order in which drivers or packages are installed. (Packages with dependencies may require installation in a certain order.)

### Replace the answer file in an offline image

1. Create a custom answer file in Windows System Image Manager (Windows SIM).
2. Open an elevated command prompt.
3. Mount the Windows image.

```
Dism /Mount-Image /ImageFile:"C:\images\CustomImage.wim" /Index:1 /MountDir:C:\mount
```

4. Modify or replace the file: \Windows\Panther\unattend.xml in the mounted image.

```
Copy CustomAnswerFile.xml C:\mount\Windows\Panther\unattend.xml
```

**Note** The answer file in the image may contain settings that have not yet been processed. If you want these settings to get processed, edit the existing file rather than replacing it.

5. Unmount the image.

```
Dism /Unmount-Image /MountDir:C:\mount /Commit
```

6. Test the image by deploying it to a new PC, without specifying an answer file. When Windows Setup runs, it finds and uses this answer file.

## Implicit Answer File Search Order

Windows Setup searches for answer files at the beginning of each configuration pass, including the initial installation and after applying and booting an image. If an answer file is found, and it contains settings for the given configuration pass, it processes those settings.

Windows Setup identifies and logs all available answer files, depending on the search order. The answer file that has the highest precedence is used. The answer file is validated and then cached to the computer. Valid answer files are cached to the \$Windows.~BT\Sources\Panther directory during the [windowsPE](#) and [offlineServicing](#) configuration passes. After the Windows installation is extracted to the hard disk, the answer file is cached to %WINDIR%\panther.

The following table shows the implicit answer file search order.

SEARCH ORDER	LOCATION	DESCRIPTION
1	Registry  HKEY_LOCAL_MACHINE\System\Setup\UnattendFile	Specifies a pointer in the registry to an answer file. The answer file is not required to be named Unattend.xml.
2	%WINDIR%\Panther\Unattend	The name of the answer file must be either Unattend.xml or Autounattend.xml.  <b>Note</b> Windows Setup searches this directory only on downlevel installations. If Windows Setup starts from Windows PE, the %WINDIR%\Panther\Unattend directory is not searched.

SEARCH ORDER	LOCATION	DESCRIPTION
3	%WINDIR%\Panther	<p>Windows Setup caches answer files to this location for use in subsequent stages of installation. For example, when a computer reboots, Setup can continue to apply the settings in an answer file. If you explicitly specify an answer file by using Windows Setup or Sysprep, the answer file cached to this directory is overwritten with the explicitly specified answer file.</p> <div style="border: 1px solid black; padding: 5px;"> <p><b>Important</b></p> <p>Do not use, modify, or overwrite the answer file in this directory. The answer file in this directory is annotated by Windows Setup during installation. This answer file cannot be reused in Windows SIM or any other Windows installations.</p> </div>
4	Removable read/write media in order of drive letter, at the root of the drive.	<p>Removable read/write media in order of drive letter, at the root of the drive.</p> <p>The name of the answer file must be Autounattend.xml, and the answer file must be located at the root of the drive.</p>
5	Removable read-only media in order of drive letter, at the root of the drive.	<p>Removable read-only media in order of drive letter, at the root of the drive.</p> <p>The name of the answer file must be Autounattend.xml, and must be located at the root of the drive.</p>
6	<p>windowsPE and offlineServicing configuration passes:</p> <ul style="list-style-type: none"> <li>• \Sources directory in a Windows distribution</li> </ul> <p>All other passes:</p> <ul style="list-style-type: none"> <li>• %WINDIR%\System32\Sysprep</li> </ul>	<p>In the windowsPE and offlineServicing configuration passes, the name of the answer file must be Autounattend.xml.</p> <p>For all other configuration passes, the file name must be Unattend.xml.</p>
7	%SYSTEMDRIVE%	<p>The answer file name must be Unattend.xml or Autounattend.xml</p>

SEARCH ORDER	LOCATION	DESCRIPTION
8	Drive from where Windows Setup (setup.exe) is running, at the root of the drive.	The name of the answer file must be Unattend.xml or Autounattend.xml, and must be located at the root of the Windows Setup folder path.

## Sensitive Data in Answer Files

Setup removes sensitive data in the cached answer file at the end of each configuration pass.

### Important

Because answer files are cached to the computer during Windows Setup, your answer files will persist on the computer between reboots. Before you deliver the computer to a customer, you must delete the cached answer file in the %WINDIR%\panther directory. There might be potential security issues if you include domain passwords, product keys, or other sensitive data in your answer file. However, if you have unprocessed settings in the [oobeSystem](#) configuration pass that you intend to run when an end user starts the computer, consider deleting the sections of the answer file that have already been processed. One option when you run the [sysprep /oobe](#) command might be to use a separate answer file that only contains settings in the oobeSystem configuration pass.

However, if an answer file is embedded in a higher precedence location than the cached answer file, then the cached answer may be overwritten at the beginning of each subsequent configuration pass, if the embedded answer file matches the implicit search criteria. For example, if an answer file is embedded at %WINDIR%\Panther\Unattend\Unattend.xml, the embedded answer file will replace the cached answer file at the beginning of each configuration pass. For example, if the embedded answer file specifies both the [specialize](#) and [oobeSystem](#) configuration passes, then the embedded answer file is discovered for the [specialize](#) configuration pass, cached, processed, and sensitive data is cleared. The embedded answer file is discovered again during the oobeSystem configuration pass and cached again. As a result, the sensitive data for the specialize configuration pass is no longer cleared. Sensitive data for previously processed configuration passes will not be cleared again. Unless the cached answer file must be overridden, embed the answer files at a location that has a lower precedence.

### Important

Because answer files are cached to the computer during Windows Setup, your answer files will persist on the computer between reboots. Before you deliver the computer to a customer, you must delete the cached answer file in the %WINDIR%\panther directory. There might be potential security issues if you include domain passwords, product keys, or other sensitive data in your answer file. However, if you have unprocessed settings in the [oobeSystem](#) configuration pass that you intend to run when an end user starts the computer, consider deleting the sections of the answer file that have already been processed. One option when you run the [sysprep /oobe](#) command might be to use a separate answer file that only contains settings in the oobeSystem configuration pass.

You can add a command to the Setupcomplete.cmd command script that deletes any cached or embedded answer files on the computer. For more information, see [Add a Custom Script to Windows Setup](#).

## Windows Setup Annotates Configuration Passes in an Answer File

After a configuration pass is processed, Windows Setup annotates the cached answer file to indicate that the pass has been processed. If the configuration pass is run again and the cached answer file has not been replaced or updated in the interim, the answer file settings are not processed again. Instead, Windows Setup will search for implicit Unattend.xml files that are at a lower precedence location than the cached Unattend.xml file.

For example, you can install Windows with an answer file that contains Microsoft-Windows-Deployment/**RunSynchronous** commands in the **specialize** configuration pass. During installation, the specialize configuration pass runs and the **RunSynchronous** commands execute. After installation, run the **sysprep** command with the **/generalize** option. If there is no answer file in a higher precedence than the cached answer file or an answer file was not explicitly passed to the Sysprep tool, Setup runs the specialize configuration pass the next time that the computer boots. Because the cached answer file contains an annotation that the settings for that configuration pass were already applied, the **RunSynchronous** commands do not execute.

## Implicit Answer File Search Examples

The following examples help describe the behavior of implicit answer file searches.

### Answer Files Named Autounattend.xml are Automatically Discovered by Windows Setup

1. Create an answer file that is named Autounattend.xml that includes settings in the [windowsPE](#) configuration pass.
2. Copy Autounattend.xml to a removable media device.
3. Configure the BIOS of your computer to boot from CD or DVD.
4. Boot the Windows product DVD.
5. Insert the removable media device when Windows is booting. This example assumes that the removable media is assigned the drive letter D:\.

Windows Setup starts and automatically identifies Autounattend.xml as a valid answer file. Because the answer file uses a valid file name (Autounattend.xml), is located in one of the valid search paths (the root of D), and includes valid settings for the current configuration pass ([windowsPE](#)), this answer file is used.

The answer file is cached to the computer. If there are no additional answer files discovered in later passes, the cached answer file is used throughout Windows Setup.

### Answer Files are Discovered in Order of Precedence in Predefined Search Paths

1. Install Windows with an answer file by using the steps in the previous scenario. The answer file that is used to install Windows is cached to the system in the %WINDIR%\Panther directory.
2. Copy an Unattend.xml file to the %WINDIR%\System32\Sysprep directory.

This answer file has settings in the **generalize** configuration pass.

3. Run the **sysprep** command with the **/generalize** option to create a reference image.

Because the %WINDIR%\System32\Sysprep directory is in the implicit search paths, the answer file copied to this directory is found. However, an answer file that was used to install Windows is still cached on the computer and contains settings for the **generalize** configuration pass. This cached answer file has a higher precedence than the one copied to the Sysprep directory. The cached answer file is used.

### Note

The Sysprep tool can be run as a command-line tool or as a GUI tool. If you run the Sysprep tool as a GUI tool, you can select the **Generalize** check box.

To use the new answer file, you can copy it to a directory of a higher precedence than the cached answer file, or you can specify the answer file by using the \*\*/unattend\*\* option. For example:

```
...
sysprep /generalize /unattend:C:\MyAnswerFile.xml
...
```

## **Answer Files Must Include a Valid Configuration Pass**

1. Copy an Unattend.xml file to a removable media device.

The Unattend.xml file has settings only for the [auditSystem](#) and [auditUser](#) configuration passes.

2. On an installed Windows operating system, run the **sysprep /generalize /oobe** command.

Even though the answer file is available in one of the implicit search paths, the Unattend.xml file is ignored because it does not contain a valid pass for the [generalize](#) configuration pass.

## Additional Resources

See the following topics for more information about answer files and configuration passes:

- [Best Practices for Authoring Answer Files](#)
- [Create or Open an Answer File](#)
- [Configure Components and Settings in an Answer File](#)
- [Validate an Answer File](#)
- [Hide Sensitive Data in an Answer File](#)
- [How Configuration Passes Work](#)

## Related topics

[Windows Setup Scenarios and Best Practices](#)

[Windows Setup Installation Process](#)

[Automate Windows Setup](#)

[Audit Mode Overview](#)

[Windows Setup Configuration Passes](#)

[Windows Setup Supported Platforms and Cross-Platform Deployments](#)

# Automate Windows Setup

1/18/2019 • 4 minutes to read

You can prevent some or all of the user interface (UI) pages from Windows Setup from being displayed during installation. The default behavior of Windows Setup is to display the Setup UI if any of the required settings are incorrect or empty.

## Use an answer file while installing Windows

You can automate Windows installation by using an answer file:

### Use a USB flash drive

1. Use an existing answer file or [create your own with Windows System Image Manager \(Windows SIM\)](#).
2. Save the file as **Autounattend.xml** on the root of a USB flash drive.
3. On a new PC, insert a Windows installation USB flash drive, as well as the flash drive that contains **Autounattend.xml** and then boot the PC. When no other answer file is selected, Windows Setup searches for this file.

### Select an answer file

- You can select a specific answer file during installation by booting to the Windows Preinstallation Environment, and [using the setup.exe command with the /unattend:filename option](#).

## List of settings

The following is a list of the settings used in these answer files:

- Windows Setup language settings: Microsoft-Windows-International-Core-WinPE\UILanguage and Microsoft-Windows-International-Core-WinPE\SetupUILanguage\UILanguage.
- Product key: Microsoft-Windows-Setup\UserData\ProductKey\Key.

## Automating Windows Setup

To automate Windows Setup, add settings for each of the following Windows Setup pages to your unattended Setup answer file. When a setting for a Windows Setup page is configured, Windows Setup skips that page.

### Language, Region, and Input Method Selection Page

SETTING	DESCRIPTION
Microsoft-Windows-International-Core-WinPE   <a href="#">UILanguage</a>	Specifies the default language to use on the installed Windows operating system.
Microsoft-Windows-International-Core-WinPE   <a href="#">SetupUILanguage</a>   <a href="#">UILanguage</a>	Specifies the default language to use during Windows Setup. During installation, Windows Setup displays installation progress in the selected language.

**Note** When you use an Autounattend.xml file with Windows Setup and rely on an implicit answer-file search, the language selection page in Setup is not displayed, even if you explicitly do not configure language settings in

your answer file. For more information about implicit answer files, see [Windows Setup Automation Overview](#).

### Type your Product Key for Activation Page

The product key must match the Windows edition you intend to install. For more information, see [Work with Product Keys and Activation](#).

SETTING	DESCRIPTION
Microsoft-Windows-Setup   UserData   ProductKey   <a href="#">Key</a>	Specifies the product key used to install Windows.
Microsoft-Windows-Setup   ImageInstall   OSImage   InstallFrom   MetaData   ( <a href="#">Key</a> and <a href="#">Value</a> ).	<p>Use Key and Value together to select a specific Windows image to install. Required for some Windows Server® 2012 editions.</p> <p>You can get the image information by using the DISM /Get-ImageInfo command. For more information, see <a href="#">Image Management Command-Line Options</a>.</p>

### Accept Microsoft Software License Terms Page

SETTING	DESCRIPTION
Microsoft-Windows-Setup   UserData   AcceptEula	Specifies whether to accept Microsoft License Software Terms during Windows Setup.

### Select Upgrade or Custom Installation Page

By default, when an answer file is used, this page does not appear and Windows is configured as a new installation. To configure Windows as an upgrade, add the following setting:

SETTING	DESCRIPTION
Microsoft-Windows-Setup   UpgradeData   <a href="#">Upgrade</a>	Specifies that the present installation is an upgrade from a previous version of Windows.

### Specify Where to Install Windows Page

You can either specify the exact disk ID and partition ID, or you can install Windows to the first available partition. To preconfigure your partitions, you may also need to configure your drive partitions. For full XML examples and recommended partition configurations, see [How to Configure UEFI/GPT-Based Hard Disk Partitions](#) or [How to Configure BIOS/MBR-Based Hard Disk Partitions](#).

SETTING	DESCRIPTION
Microsoft-Windows-Setup   ImageInstall   OSImage   InstallTo   <a href="#">DiskID</a>	Specifies the disk where Windows will be installed.
Microsoft-Windows-Setup   ImageInstall   OSImage   InstallTo   <a href="#">PartitionID</a>	Specifies the partition where Windows will be installed.

-or-

SETTING	DESCRIPTION
Microsoft-Windows-Setup   ImageInstall   OSImage   <a href="#">InstallToAvailablePartition</a>	Specifies to install Windows on the first available partition.

## Settings to Use with Unattended Windows Deployment Services

When deploying Windows using Windows Deployment Services, add each of the settings in the following sections to your unattended-Setup answer file. These are the only settings required for an unattended installation.

### Select a Language and Locale Page

SETTING	DESCRIPTION
Microsoft-Windows-International-Core-WinPE   SetupUILanguage   <a href="#">UILanguage</a>	Specifies the default language to use during Windows Setup.

### Provide Windows Deployment Services Credentials Page

SETTING	DESCRIPTION
Microsoft-Windows-Setup   WindowsDeploymentServices   <a href="#">Login</a>	Specifies the credentials used for Windows Deployment Services logon, and specifies in what circumstances the UI is displayed for logon.

### Select an Image to Install Page

SETTING	DESCRIPTION
Microsoft-Windows-Setup   WindowsDeploymentServices   <a href="#">ImageSelection</a>	Specifies the image to be installed and the location where it is installed, as well as whether the UI is displayed.

### Specify Where to Install Windows Page

These settings assume that you are installing to a partitioned disk drive.

SETTING	DESCRIPTION
Microsoft-Windows-Setup   WindowsDeploymentServices   ImageSelection   InstallTo   <a href="#">DiskID</a>	Specifies the disk ID of the disk to which the image is to be installed.
Microsoft-Windows-Setup   WindowsDeploymentServices   ImageSelection   InstallTo   <a href="#">PartitionID</a>	Specifies the partition ID of the partition to which the image is to be installed.

## Related topics

[Settings for Automating OOB](#)

[Windows Setup Technical Reference](#)

# Windows Setup: Installing using the MBR or GPT partition style

1/18/2019 • 3 minutes to read

When installing Windows on UEFI-based PCs using Windows Setup, your hard drive partition style must be set up to support either UEFI mode or legacy BIOS-compatibility mode.

For example, if you receive the error message: "Windows cannot be installed to this disk. The selected disk is not of the GPT partition style", it's because your PC is booted in UEFI mode, but your hard drive is not configured for UEFI mode. You've got a few options:

1. Reboot the PC in legacy BIOS-compatibility mode. This option lets you keep the existing partition style. For more info, see [Boot to UEFI Mode or Legacy BIOS mode](#).
2. Configure your drive for UEFI by using the GPT partition style. This option lets you use the PC's UEFI firmware features.

You can preserve your data and convert the drive using the [MBR2GPT tool](#). You can also choose to reformat the drive using the instructions below. Reformatting will erase all the data on the drive.

## Why should I convert my drive?

Many PCs now include the ability to use the UEFI version of BIOS, which can speed up boot and shutdown times and can provide additional security advantages. To boot your PC in UEFI mode, you'll need to use a drive formatted using the GPT drive format.

Many PCs are ready to use UEFI, but include a compatibility support module (CSM) that is set up to use the legacy version of BIOS. This version of BIOS was developed in the 1970s and provides compatibility to a variety of older equipment and network configurations, and requires a drive that uses the MBR drive format.

However, the basic MBR drive format does not support drives over 4TB. It's also difficult to set up more than four partitions. The GPT drive format lets you set up drives that are larger than 4 terabytes (TB), and lets you easily set up as many partitions as you need.

## Reformatting the drive using a different partition style

### To wipe and convert the drive by using Windows Setup

1. Turn off the PC, and put in the Windows installation DVD or USB key.
2. Boot the PC to the DVD or USB key in UEFI mode. For more info, see [Boot to UEFI Mode or Legacy BIOS mode](#).
3. When choosing an installation type, select **Custom**.
4. On the **Where do you want to install Windows?** screen, select each of the partitions on the drive, and select **Delete**. The drive will show a single area of unallocated space.
5. Select the unallocated space and click **Next**. Windows detects that the PC was booted into UEFI mode, and reformats the drive using the GPT drive format, and begins the installation.

### To manually wipe a drive and convert it to GPT:

1. Turn off the PC, and put in the Windows installation DVD or USB key.

2. Boot the PC to the DVD or USB key in UEFI mode. For more info, see [Boot to UEFI Mode or Legacy BIOS mode](#).
3. From inside Windows Setup, press **Shift+F10** to open a command prompt window.
4. Open the diskpart tool:

```
diskpart
```

5. Identify the drive to reformat:

```
list disk
```

6. Select the drive, and reformat it:

```
select disk <disk number>
clean
convert gpt
exit
```

7. Close the command prompt window.

8. Continue the Windows Setup installation.

When choosing an installation type, select **Custom**. The drive will appear as a single area of unallocated space.

Select the unallocated space and click **Next**. Windows begins the installation.

## Make sure Windows Setup boots to the correct firmware mode

To automate this process, you'll need to run Windows Setup through Windows PE, and use a script to detect which mode you're in before installing Windows. For more info, see [WinPE: Boot in UEFI or legacy BIOS mode](#).

## Related topics

[Boot to UEFI Mode or Legacy BIOS mode](#)

# Add languages to Windows Setup

1/18/2019 • 4 minutes to read

To support multiple languages with Windows Setup:

- Add languages to the Windows Setup image (boot.wim index:2) to help technicians or users install Windows images through Windows Setup.
- When you add languages to a Windows image (install.wim), add a list of languages (lang.ini) so that during installation, the technician or user can select which language is installed as the default language.

You can include a different set of languages for your Windows images and Windows Setup.

## Prerequisites

- A technician computer that has the Windows Assessment and Deployment Kit (Windows ADK) installed with the **Deployment tools** option, including the **WinPE add-on**.

**CopyPE** and **MakeWinPEMedia** are installed when you choose the **Deployment tools** and **Windows Preinstallation Environment** options when [installing the ADK](#). If you're using the ADK for Windows 10, version 1809, Windows PE is a separate add-on that you download and install after you install the ADK. You can [download the WinPE add-on from the Download Center](#).

- Windows installation media for all languages that you are creating media.
- The Windows language pack ISO.

## Step 1. Copy installation files to a working folder

On your technician PC, copy the Windows installation files from the Windows installation media to a local directory.

If you are creating media for use with a customized image, you must use the Windows media that corresponds to the version of your customized image. For example, if you are building a custom Windows 10 Setup image, you must use the original Windows 10 product media.

```
md C:\my_distribution  
xcopy /E D: C:\my_distribution
```

Where *D:* is the location of the Windows installation media.

## Step 2. Add languages to the default Windows Setup boot image (index:2)

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. Mount the second image (index 2) in Boot.wim.

```
md C:\mount\boot  
Dism /mount-image /imagefile:C:\my_distribution\sources\boot.wim /index:2 /mountdir:C:\Mount\boot
```

3. Add language packs from the Windows ADK into your mounted image using **Dism /Add-Package** for each language you want to support. Add *lp.cab*, *WinPE-setup\_<language>.cab*. For Windows 10, also add *WinPE-Setup-client\_<language>.cab*. For Windows Server, also add *WinPE-Setup-server\_<language>.cab*  
Windows PE language packs are available in the Windows ADK.

#### For Windows 10:

```
Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\lp.cab"

Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-Setup_fr-fr.cab"

Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-Setup-Client_fr-fr.cab"
```

#### For Windows Server:

```
Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\lp.cab"

Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-Setup_fr-fr.cab"

Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-Setup-Server_fr-fr.cab"
```

4. For Japanese (ja-JP), Korean (ko-KR), and Chinese (zh-HK, zh-CN, zh-TW), add font support to your image.  
For example, to add Japanese font support:

```
Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-FontSupport-JA-JP.cab"
```

5. Change the Windows Setup default language:

```
Dism /image:C:\mount\boot /Set-SetupUILang:fr-FR /distribution:C:\my_distribution
```

For more information about specifying different international settings, see [DISM Languages and International Servicing Command-Line Options](#).

## Step 3: Add localized Windows Setup resources to the Windows distribution

In this step you copy the language-specific Setup resources from each language-specific Windows distribution to the Sources folder in your Windows distribution. For example, mount the fr-FR Windows installation media as drive E: and copy the Fr-FR sources folder to your Windows distribution.

Note, language interface packs (LIP) are not supported on Windows Setup.

- Copy the localized Windows Setup files to your Windows distribution.

```
xcopy E:\sources\fr-fr C:\my_distribution\sources\fr-fr /cherkyi
```

Where *E:* is the location of the Windows installation media that contains the localized Windows Setup resources.

## Step 4. Add language packs to the Windows image

To modify the languages in the Windows image, see [Add Language Packs to Windows](#).

After you do this, update the list of available Windows languages (Lang.ini):

1. Recreate the Lang.ini file to reflect the additional languages.

```
Dism /image:C:\mount\windows /gen-langINI /distribution:C:\my_distribution
```

2. Copy the lang.ini file in the Windows distribution to the boot folder.

```
Xcopy C:\my_distribution\sources\lang.ini C:\mount\boot\sources\lang.ini
```

## Step 6: Commit the changes to the Windows images

```
...
Dism /unmount-image /mountdir:C:\mount\boot /commit
Dism /unmount-image /mountdir:C:\mount\windows /commit
...
```

## Step 7: Create the media

### USB flash drive

Use **MakeWinPEMedia** with the `/UFD` option to format and install Windows PE to the USB flash drive, specifying the USB key's drive letter:

```
...
MakeWinPEMedia /UFD C:\my_distribution F:
...
```

To learn more, see [WinPE: Create USB Bootable drive](#).

### DVD or CD

1. Create a **boot order file**. This is required for DVD or CD images larger than 4.5 GB to make sure that boot files are located at the beginning of the image.

For example, copy this into a text file called C:\Temp\BootOrder.txt:

```
boot\bcd
boot\boot.sdi
boot\bootfix.bin
boot\bootsect.exe
boot\etfsboot.com
boot\memtest.efi
boot\memtest.exe
boot\en-us\bootsect.exe.mui
boot\fonts\chs_boot.ttf
boot\fonts\cht_boot.ttf
boot\fonts\jpn_boot.ttf
boot\fonts\kor_boot.ttf
boot\fonts\wg14_boot.ttf
sources\boot.wim
```

2. Create the ISO file, using the boot order file:

```
Oscdimg -m -n -yo C:\temp\BootOrder.txt -bC:\winpe_amd64\Efisys.bin C:\winpe_amd64\winpeamd64.iso
```

3. Right-click the ISO file and select **Burn disc image**.

## Related topics

[Windows Setup Technical Reference](#)

[DISM Image Management Command-Line Options](#)

[DISM Windows PE Servicing Command-Line Options](#)

[Oscdimg Command-Line Options](#)

[WinPE: Mount and Customize](#)

[WinPE: Install on a Hard Drive \(Flat Boot or Non-RAM\)](#)

# Answer files (unattend.xml)

1/18/2019 • 6 minutes to read

Answer files (or Unattend files) can be used to modify Windows settings in your images during Setup. You can also create settings that trigger scripts in your images that run after the first user creates their account and picks their default language.

Windows Setup will automatically search for [answer files in certain locations](#), or you can specify an unattend file to use by using the `/unattend:` option when running Windows Setup (setup.exe).

## Windows settings overview

While you can set many Windows settings in audit mode, some settings can only be set by using an answer file or Windows Configuration Designer, such as adding manufacturer's support information. A full list of answer file settings (also known as Unattend settings) is in the [Unattended Windows Setup Reference](#).

Enterprises can control other settings by using Group Policy. For more info, see [Group Policy](#).

## Answer file settings

You can specify which configuration pass to add new settings:

- **1 windowsPE:** These settings are used by the Windows Setup installation program. If you're modifying existing images, you can usually ignore these settings.
- **4 specialize:** Most settings should be added here. These settings are triggered both at the beginning of audit mode and at the beginning of OOBE. If you need to make multiple updates or test settings, generalize the device again and add another batch of settings in the Specialize Configuration pass.
- **6 auditUser:** Runs as soon as you start audit mode.

This is a great time to run a system test script - we'll add [Microsoft-Windows-Deployment\RunAsynchronousCommand](#) as our example. To learn more, see [Add a Custom Script to Windows Setup](#).

- **7 oobeSystem:** Use sparingly. Most of these settings run after the user completes OOBE. The exception is the Microsoft-Windows-Deployment\Reseal\Mode = Audit setting, which we'll use to bypass OOBE and boot the PC into audit mode.

If your script relies on knowing which language the user selects during OOBE, you'd add it to the oobeSystem pass.

- To learn more, see [Windows Setup Configuration Passes](#).

### NOTE

These settings could be lost if the user resets their PC with the built-in recovery tools. To see how to make sure these settings stay on the device during a reset, see [Sample scripts: Keeping Windows settings through a recovery](#).

## Create and modify an answer file

### Step 1: Create a catalog file

1. Start **Windows System Image Manager**.
2. Click **File > Select Windows Image**.
3. In **Select a Windows Image**, browse to and select the image file (D:\install.wim). Next, select an edition of Windows, for example, Windows 10 Pro, and click **OK**. Click **Yes** to create the catalog file. Windows SIM creates the file based on the image file, and saves it to the same folder as the image file. This process can take several minutes.

The catalog file appears in the **Windows Image** pane. Windows SIM lists the configurable components and packages in that image.

**Troubleshooting:** If Windows SIM does not create the catalog file, try the following steps:

- To create a catalog file for either 32-bit or ARM-based devices, use a 32-bit device.
- Make sure the Windows base-image file (**\Sources\Install.wim**) is in a folder that has read-write privileges, such as a USB flash drive or on your hard drive.

## Step 2: Create an answer file

- Click **File > New Answer File**.

The new answer file appears in the **Answer File** pane.

**Note** If you open an existing answer file, you might be prompted to associate the answer file with the image. Click **Yes**.

## Step 3: Add new answer file settings

1. Add OEM info:

In the **Windows Image** pane, expand **Components**, right-click **amd64\_Microsoft-Windows-Shell-Setup\_(version)**, and then select **Add Setting to Pass 4 specialize**.

In the **Answer File** pane, select **Components\4 specialize\amd64\_Microsoft-Windows-Shell-Setup\_neutral\OEMInformation**.

In the **OEMInformation Properties** pane, in the **Settings** section, select:

- Manufacturer=
- Model=
- Logo=

Create a 32-bit color with a maximum size of 120x120 pixels, save it as  file on your local PC, or use the sample from the USB-B key: .

We'll copy the logo into the Windows image in a few steps.

2. Set the device to automatically [boot to audit mode](#):

In the **Windows Image** pane, expand **Components**, right-click **amd64\_Microsoft-Windows-Deployment\_(version)**, and then select **Add Setting to Pass 7 oobeSystem**.

In the **Answer File** pane, select **Components\7 oobeSystem\amd64\_Microsoft-Windows-Deployment\_neutral\Reseal**.

In the **Reseal Properties** pane, in the **Settings** section, select Mode= .

3. Prepare a [script](#) to run after Audit mode begins.

In the **Windows Image** pane, right-click **amd64\_Microsoft-Windows-Deployment\_(version)** and then click **Add Setting to Pass 6 auditUser**.

In the **Answer File** pane, expand **Components\6\_auditUser\amd64\_Microsoft-Windows-Deployment\_neutral\RunAsynchronous**. Right-click **RunAsynchronousCommand Properties** and click **Insert New AsynchronousCommand**.

In the **AsynchronousCommand Properties** pane, in the **Settings** section, add the following values:

```
Path = C:\Fabrikam\SampleCommand.cmd
```

```
Description = Sample command to run a system diagnostic check.
```

```
Order = 1 (Determines the order that commands are run, starting with 1.)
```

4. Add a registry key. In this example, we add keys for the OEM Windows Store program. Use the same process as adding a script, using `CMD /c REG ADD`.

For Windows 10 Customer Systems, you may use the OEM Store ID alone or in combination with a Store Content Modifier (SCM) to identify an OEM brand for the OEM Store. By adding a SCM, you can target Customer Systems at a more granular level. For example, you may choose to target commercial devices separately from consumer devices by inserting unique SCMs for consumer and commercial brands into those devices.

Add RunAsynchronousCommands for each registry key to add. (Right-click **RunAsynchronousCommand Properties** and click **Insert New AsynchronousCommand**).

```
Path = CMD /c REG ADD HKEY_LOCAL_MACHINE\Software\OEM /v Value /t REG_SZ ABCD
```

```
Description = Adds a OEM registry key
```

```
Order = 2
```

```
RequiredUserInput = false
```

See the [Unattended Windows Setup Reference](#) for a full list of configuration options.

#### Step 4: Save the answer file

- Save the answer file, for example: **D:\AnswerFiles\BootToAudit-x64.xml**.

**Note** Windows SIM will not allow you to save the answer file into the mounted image folders. **Step 5: Create a script**

Since we specified a script to run in Step 3, let's create that script now.

- Copy the following sample script into Notepad, and save it as **D:\AnswerFiles\SampleCommand.cmd**.

```
@rem Scan the integrity of system files
@rem (Required after removing the base English language from an image)
sfc.exe /scannow

@rem Check to see if your drivers are digitally signed, and send output to a log file.
md C:\Fabrikam
C:\Windows\System32\dxdiag /t C:\Fabrikam\dxdiag-testlogfiles.txt
```

## Add the answer file and script to the image

#### Step 6: Mount an image and add the answer file

1. Use DISM to mount a Windows image. To learn how to mount an image, see [Mount and modify a Windows image using DISM](#)
2. Copy the answer file into the image into the **\Windows\Panther** folder, and name it `unattend.xml`. The Panther folder is one of the folders where Windows searches for an answer file. Create the folder if it

doesn't exist. If there's an existing answer file, replace it or use Windows System Image Manager to edit/combine settings if necessary.

```
Mkdir c:\mount\windows\Windows\Panther  
Copy D:\AnswerFiles\BootToAudit-x64.xml C:\mount\windows\Windows\Panther\unattend.xml  
Mkdir c:\mount\windows\Fabrikam  
Copy D:\AnswerFiles\Fabrikam.bmp C:\mount\windows\Fabrikam\Fabrikam.bmp  
Copy D:\AnswerFiles\SampleCommand.cmd C:\mount\windows\Fabrikam\SampleCommand.cmd
```

3. Unmount the image, committing the changes. For example:

```
Dism /Unmount-Image /MountDir:"C:\mount\windows" /Commit
```

where C is the drive letter of the drive that contains the mounted image.

This process may take several minutes.

When you [apply your image](#), Windows will process your unattend file and will configure your settings based on what you specified in the unattend.

# Run custom actions during feature update

1/18/2019 • 2 minutes to read

## Overview

Starting with Windows 10, version 1803, enterprise IT admins can run their own custom actions by running scripts during an upgrade or feature update.

## Running custom actions

### About custom actions

Custom actions are .cmd scripts that run during the feature update process. These can be run during two phases of an upgrade:

1. Pre-install phase: This phase is when Setup starts, but prior compatibility checks. Actions during this phase are specified in `preinstall.cmd`.
2. Pre-commit phase: This phase is prior to the upgrade being applied and the system rebooting. Actions during this phase are specified in `precommit.cmd`.

### Custom actions folder structure

Custom action scripts run from subfolders in `%windir%\System32\update\`, depending on whether you want scripts to run for a single upgrade or on all future upgrades. Windows Setup automatically looks for these scripts in the following locations, and will run them if they are present:

FOLDER	DESCRIPTION
<code>%windir%\System32\update\run</code>	Windows Setup will migrate scripts in these folders so they are run in future upgrades.
<code>%windir%\System32\update\runonce</code>	Scripts in this folder will only run in one upgrade and won't be migrated for future upgrades.
<code>%windir%\System32\update\run\&lt;GUID&gt;</code> <code>%windir%\System32\update\runonce\&lt;GUID&gt;</code>	Create a unique GUID to clearly identify each script you run. Use this GUID to name a folder within the <code>run</code> and <code>runonce</code> folders.
<code>%windir%\System32\update\run\&lt;GUID&gt;\reflectdrivers</code> <code>%windir%\System32\update\runonce\&lt;GUID&gt;\reflectdrivers</code>	Boot-critical drivers in this folder will be reflected. Make sure the reflectdrivers folder only contains a necessary set of encryption drivers. Having more drivers than necessary can negatively impact upgrade scenarios. See <a href="#">Device drivers</a> to learn more about reflected drivers.

If an update fails or has to be rolled back, a failure.cmd script can be used to perform custom operations or undo actions of previous custom scripts. If the scripts fail, the update will fail.

## Custom action script locations and examples

### Script locations

The following table shows the folder path structure of files for running custom actions:

RUN (SCRIPTS WILL BE MIGRATED)	RUN ONCE (SCRIPTS WON'T BE MIGRATED)
%windir%\System32\update\run\<GUID>\preinstall.cmd	%windir%\System32\update\runonce\<GUID>\preinstall.cmd
%windir%\System32\update\run\<GUID>\precommit.cmd	%windir%\System32\update\runonce\<GUID>\precommit.cmd
%windir%\System32\update\run\<GUID>\failure.cmd	%windir%\System32\update\runonce\<GUID>\failure.cmd
%windir%\System32\update\run\<GUID>\reflectdrivers\example1.inf	%windir%\System32\update\runonce\<GUID>\reflectdrivers\example2.inf
%windir%\System32\update\run\<GUID>\reflectdrivers\example1.sys	%windir%\System32\update\run\<GUID>\reflectdrivers\example2.sys

### Example file paths and names

SCENARIOS	FILE LOCATION EXAMPLES
<b>Script deployment data</b>	%windir%\system32\update\
<b>Migration directories</b>	
Scripts that will be migrated	%windir%\system32\update\run\
Scripts that won't be migrated	%windir%\system32\update\runonce\
<b>Folder path with a GUID</b>	%windir%\system32\update\run\6971CB27-6F59-43CD-A764-969EE9BBCC1C\
<b>Script names with GUID</b>	
	%windir%\system32\update\run\6971CB27-6F59-43CD-A764-969EE9BBCC1C\preinstall.cmd
	%windir%\system32\update\run\6971CB27-6F59-43CD-A764-969EE9BBCC1C\precommit.cmd
	%windir%\system32\update\run\6971CB27-6F59-43CD-A764-969EE9BBCC1C\failure.cmd
	%windir%\system32\update\runonce\6971CB27-6F59-43CD-A764-969EE9BBCC1C\preinstall.cmd
	%windir%\system32\update\runonce\6971CB27-6F59-43CD-A764-969EE9BBCC1C\precommit.cmd
	%windir%\system32\update\runonce\6971CB27-6F59-43CD-A764-969EE9BBCC1C\failure.cmd
<b>Reflected drivers folder</b>	%windir%\system32\update\run\6971CB27-6F59-43CD-A764-969EE9BBCC1C\reflectdrivers\

# Add a Custom Script to Windows Setup

1/18/2019 • 4 minutes to read

**Windows Setup scripts:** **Setupcomplete.cmd** and **ErrorHandler.cmd** are custom scripts that run during or after the Windows Setup process. They can be used to install applications or run other tasks by using **cscript/wscript** scripts.

- **%WINDIR%\Setup\Scripts\SetupComplete.cmd:** This script runs immediately after the user sees the desktop. This setting is disabled when using OEM product keys. It runs with local system permission.
- **%WINDIR%\Setup\Scripts\ErrorHandler.cmd:** This script runs automatically when Setup encounters a fatal error. It runs with local system permission.

**Windows Unattend scripts:** Create an Unattend.xml file with one of these settings to run during the Windows Setup process. This can be used with OEM product keys.

To run services or commands that can start at the same time, use RunAsynchronousCommands. To run commands that need to finish before other commands can start, use RunSynchronousCommands.

**Note** As of Windows 10, [Microsoft-Window-Shell-Setup\LogonCommands\AsynchronousCommand](#) now works like LogonCommands\AsynchronousCommand: all commands using these unattend settings are now started at the same time, and no longer wait for the previous command to finish.

Some of these settings run in the user context, others run in the system context depending on the configuration pass.

- Add [Microsoft-Windows-Setup\RunAsynchronousCommand](#) or [RunSynchronousCommand](#) to run a script as Windows Setup starts. This can be helpful for setting hard disk partitions.
- Add [Microsoft-Windows-Deployment\RunAsynchronousCommand](#) or [RunSynchronousCommand](#) to the **auditUser** configuration pass to run a script that runs when the PC enters audit mode. This can be helpful for tasks like automated app installation or testing.
- Add [Microsoft-Windows-Shell-Setup\LogonCommands\AsynchronousCommand](#) or [FirstLogonCommands\SynchronousCommand](#) to run after the Out of Box Experience (OOBE) but before the user sees the desktop. This can be especially useful to set up language-specific apps or content after the user has already selected their language.

Use these scripts sparingly because long scripts can prevent the user from reaching the Start screen quickly. For retail versions of Windows, additional restrictions apply to these scripts. For info, see the Licensing and Policy guidance on the [OEM Partner Center](#).

**Note** When you add a script using FirstLogonCommands, it will be triggered on the next boot, even if you boot into audit mode using Ctrl+Shift+F3. To boot to audit mode without triggering these scripts, add the setting: [Microsoft-Windows-Deployment\Reseal\Mode](#) = Audit.

## Run a script after setup is complete (SetupComplete.cmd)

### Order of operations

1. After Windows is installed but before the logon screen appears, Windows Setup searches for the **SetupComplete.cmd** file in the **%WINDIR%\Setup\Scripts\** directory.
2. If a **SetupComplete.cmd** file is found, Windows Setup runs the script. Windows Setup logs the action in the **C:\Windows\Panther\UnattendGC\Setupact.log** file.

Setup does not verify any exit codes or error levels in the script after it executes **SetupComplete.cmd**.

**Warning** You cannot reboot the system and resume running **SetupComplete.cmd**. You should not reboot the system by adding a command such as **shutdown -r**. This will put the system in a bad state.

3. If the computer joins a domain during installation, the Group Policy that is defined in the domain is not applied to the computer until **Setupcomplete.cmd** is finished. This is to make sure that the Group Policy configuration activity does not interfere with the script.

## Run a script if Windows Setup encounters a fatal error (ErrorHandler.cmd)

This script is useful when you're installing many systems at the same time. This helps you detect when an error occurs during Windows Setup. When it does, Setup automatically runs a script that can contain custom commands or actions to address the cause of the error.

If Windows Setup encounters a fatal error and is prevented from completing the installation, Windows Setup searches for a command script in the following directory: **%WINDIR%\Setup\Scripts\ErrorHandler.cmd**. One of two actions will occur, depending on whether the script is found.

- If the script is not found, a dialog box is displayed with the error text. A user must dismiss the dialog box before Windows Setup exits.
- If the script is found, the script executes synchronously. No dialog box or error text is displayed. After the **ErrorHandler.cmd** script has finished running, Windows Setup exits.

Depending on the phase of Windows Setup, the computer will return to the environment from which Windows Setup was executed, such as an earlier version of the operating system or Windows Preinstallation Environment (Windows PE), for example.

There may be instances when Windows Setup encounters more than one error and runs the **ErrorHandler.cmd** script more than once. When developing the code for **ErrorHandler.cmd**, make sure that you can run this script multiple times.

**To use ErrorHandler.cmd**, you can do either of the following:

- Mount the image, and add it to the image, in **%WINDIR%\Setup\Scripts\ErrorHandler.cmd**. Unmount the image.  
-or-
- Add **ErrorHandler.cmd** to a temporary file location (for example, C:\Temp\ErrorHandler.cmd), and then run Windows Setup using the **/m** option.

```
Setup /m:C:\Temp
```

To learn more, see [Windows Setup Command-Line Options](#).

## Related topics

[Windows Setup Technical Reference](#)

[Boot from a DVD](#)

[Use a Configuration Set with Windows Setup](#)

[Deploy a Custom Image](#)

[Boot Windows to Audit Mode or OOBE](#)

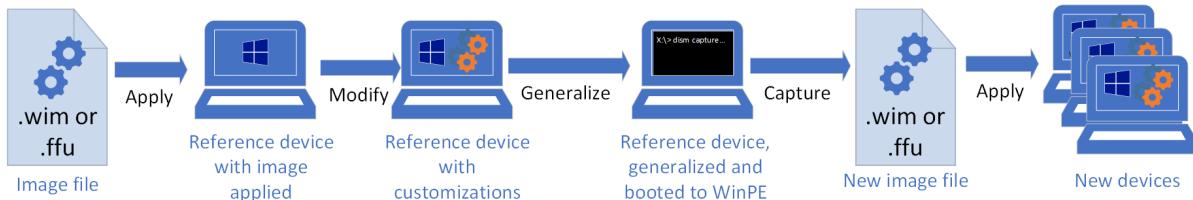
## Add Device Drivers to Windows During Windows Setup

# Modify a Windows image

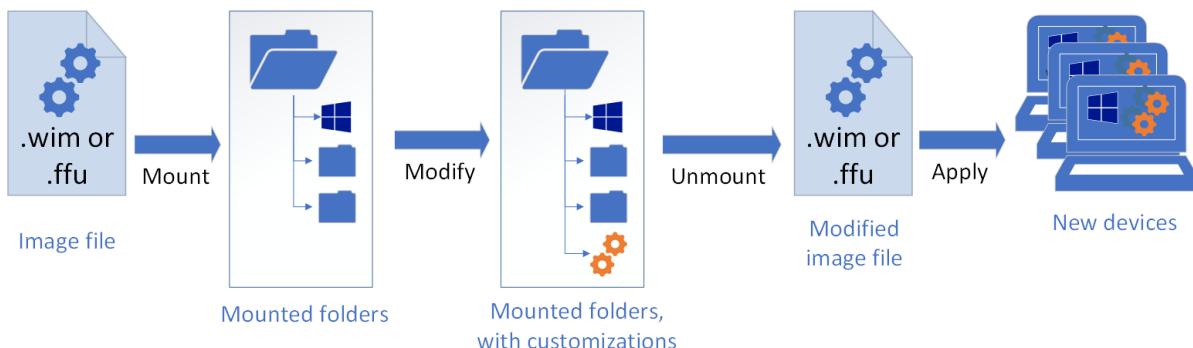
1/24/2019 • 2 minutes to read

You can service, or make changes to, a Windows image in different ways:

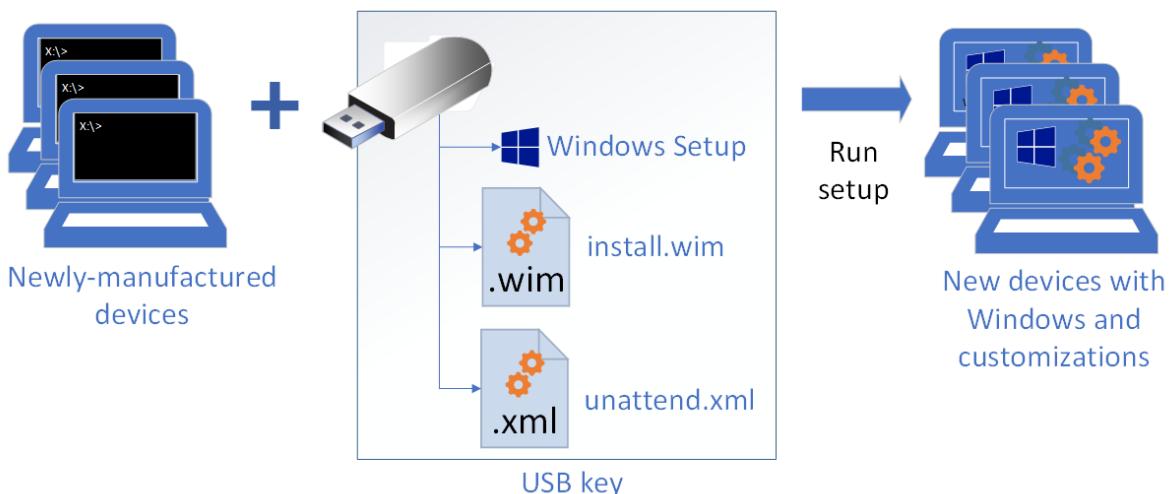
- **Online servicing (audit mode):** Update Windows from a familiar Windows environment. Apply your Windows image to a new reference device, and boot it into the built-in administrator account. From here, you can add drivers, apps, and customizations. When you're done, prepare (generalize) the device and capture a new image file that can be applied to new devices.



- **Offline servicing:** Update Windows faster by using DISM to make your changes without ever booting Windows. Mount your image to a temporary location, install apps, drivers, languages, and more, and then commit the changes so they can be applied to new devices. DISM works from an elevated command-line or from PowerShell, which makes it easier to automate your changes with scripts.



- **Servicing an image by using Windows Setup:** During final deployment, you can use Windows Setup, plus a customized answer file (unattend.xml), to make final modifications.



In this section

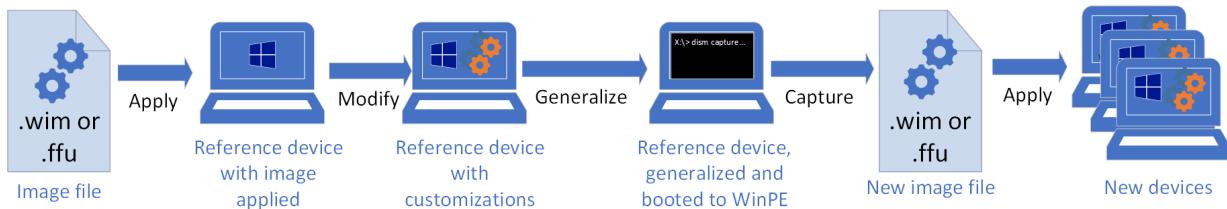
TOPIC	DESCRIPTION
<a href="#">Audit mode</a>	Update Windows from a familiar Windows environment.
<a href="#">Mount and Modify a Windows Image Using DISM</a>	How to use DISM to mount a Windows image and make changes
<a href="#">Repair a Windows Image</a>	How to repair a corrupted Windows image

# Audit Mode Overview

1/18/2019 • 2 minutes to read

When Windows boots, it starts in either Out-Of-Box Experience (OOBE) mode or in audit mode. OOBE is the default out-of-box experience that allows end users to enter their account information, select language, accept the Microsoft Terms of Service, and set up networking.

You can configure Windows to boot to audit mode instead. In audit mode, you can make additional changes to the Windows installation before you send the computer to a customer or capture the image for reuse in your organization. For example, you can install drivers included in a driver package, install applications, or make other updates that require the Windows installation to be running. When you use an answer file, Windows processes settings in the [auditSystem](#) and [auditUser](#) configuration passes.



When you boot to audit mode, you log into the system using the built-in administrator account. After you log on to the system, the built-in administrator account is immediately disabled during the [auditUser](#) configuration pass. The next time that the computer reboots, the built-in administrator account remains disabled. For more information, see [Enable and Disable the Built-in Administrator Account](#).

## Important

- If you are in audit mode and a password-protected screen saver starts, you cannot log back on to the system. The built-in administrator account that was used to log on to audit mode is immediately disabled after logon.

To disable the screen saver, either change the power plan through Control Panel or configure and deploy a custom plan. For more information, see [Create a Custom Power Plan](#).

- Settings in an unattended answer file from the [oobeSystem](#) configuration pass do not appear in audit mode.
- If you're running scripts, installers, and diagnostic tools on Windows 10 S in Audit Mode, you may have to enable manufacturing mode for Windows 10 S. See [Manufacturing mode](#) for details.

## Benefits of using Audit Mode

In audit mode, you can do the following:

- **Bypass OOBE.** You can access the desktop as quickly as possible. You do not have to configure default settings such as a user account, location, and time zone.
- **Install applications, add device drivers, and run scripts.** You can connect to a network and access additional installation files and scripts. You can also install additional language packs and device drivers. For more information, see [Add a Driver Online in Audit Mode](#).
- **Test the validity of a Windows installation.** Before you deploy the system to end users, you can perform tests on the system without creating a user account. Then you can prepare the system to start in OOBE on the next boot.

- **Add more customizations to a reference image.** This reduces the number of images that you have to manage. For example, you can create a single reference image that contains the basic customizations that you want to apply to all Windows images. You can then boot the reference image to audit mode and make additional changes that are specific to the computer. These changes can be customer-requested applications or specific device drivers.

## Boot to Audit Mode

You can boot to audit mode on a new or existing Windows installation. For more information, see [Boot Windows to Audit Mode or OOB](#)E.

## Related topics

[Understanding Servicing Strategies](#)

[Windows Setup Configuration Passes](#)

[How Configuration Passes Work](#)

[Windows Setup Scenarios and Best Practices](#)

[Windows Setup Installation Process](#)

[Windows Setup Automation Overview](#)

[Windows Setup Supported Platforms and Cross-Platform Deployments](#)

[Windows 10 S manufacturing mode](#)

# Run Audit Mode in the Factory

1/18/2019 • 2 minutes to read

In build-to-order scenarios OEMs can boot the destination PCs to audit mode to install customer-specific apps, languages, drivers, and make additional configurations.

After final assembly of the PC you complete integrity testing to ensure the PC is configured correctly.

When ready, boot the PC with Windows PE, or another operating system that allows you to install your custom Windows image to the PC. You can boot the PC by using a USB key, or you can boot the PC from the network using PXE boot and Windows Deployment Services.

Use Windows PE and DISM to boot the PC and apply your custom Windows image.

- [Apply Images Using DISM](#)
- [WinPE for Windows 10](#)
- [Windows Deployment Services Overview](#)

After the image is applied, you boot the PC to audit mode.

- [Audit Mode Overview](#)

While in audit mode, you can install customer requested software, drivers specific to the PC, and additional items. While in audit mode you can also install the latest Windows updates. The following topics go into more detail about how to install drivers, language packs, and Windows updates:

- [Device Drivers and Deployment Overview](#)
- [Language Packs](#)
- [Service a Windows Image Using DISM](#)

Keep in mind that the more items that you install on the factory floor increases the time it takes to assemble, install, and box the PC.

## NOTE

Running scripts, installers, and diagnostic tools in Audit Mode on Windows 10 S may require enabling manufacturing mode for Windows 10 S. See [Manufacturing mode](#) for details on how to enable manufacturing mode.

After you complete your audit mode installations, you must run sysprep /oobe to ensure that the end-user goes through the out-of-box experience and accepts the license terms. You should capture the Windows installation to the recovery partition to help users rest the PC to factory default. By doing this in the factory, you can ensure that the build-to-order customizations that customers make are in the recovery image.

You will need to boot the PC to Windows PE again to capture and apply the Windows installation to the recovery partition.

The following topic describes how to create the recovery image:

- [Deploy Push-Button Reset Features](#)

After the recovery image is captured, you can shut down the PC, box it, and ship it.

Depending on the volume of units you are shipping, you might want to consider pulling one or more PCs off the line to ensure the systems you build meet your quality expectations.

# Boot Windows to Audit Mode or OOBE

1/18/2019 • 5 minutes to read

You can use audit mode to customize your computer, add applications and device drivers, and test your computer in a Windows environment. Booting to audit mode starts the computer in the built-in administrator account. Windows® removes this account automatically during the [generalize](#) configuration pass. After you configure a computer to boot to audit mode, the computer will continue to boot to audit mode by default until you configure the computer to boot to Out-Of-Box Experience (OOBE) when the computer ships to the user.

If a password-protected screen saver starts when you are in audit mode, you cannot log back on to the system. The built-in administrator account that is used to log on to audit mode is immediately disabled after logon. To disable the screen saver, either change the power plan through Windows Control Panel or configure and deploy a custom plan. For more information, see [Create a Custom Power Plan](#).

## Boot to audit mode automatically on a new installation

- To configure Windows to boot to audit mode, add the **Microsoft-Windows-Deployment | Reseal | Mode = audit** answer file setting.

When Windows completes the installation process, the computer boots into audit mode automatically, and the System Preparation (**Sysprep**) Tool appears. For more information about using the Sysprep tool in audit mode, see [Sysprep \(Generalize\) a Windows installation](#).

**Note** Settings in an answer file from the **oobeSystem** configuration pass do not appear in audit mode. For more information about which answer file settings are processed when you boot to audit mode or OOBE, see [How Configuration Passes Work](#).

## Boot to audit mode manually (on a new or existing installation)

- At the OOBE screen, press **CTRL+SHIFT+F3**.

Windows reboots the computer into audit mode, and the System Preparation (**Sysprep**) Tool appears.

**Note** The **CTRL+SHIFT+F3** keyboard shortcut does not bypass all parts of the OOBE process, such as running scripts and applying answer file settings in the **oobeSystem** configuration pass.

## Boot to OOBE automatically on a new installation

- To configure Windows to boot to OOBE, add the **Microsoft-Windows-Deployment | Reseal | Mode = oobe** answer file setting.

If you have configured your Windows image to boot to OOBE, but then you need to make further configurations to your image in audit mode, see [Modify an existing image that is configured to boot to OOBE](#).

## Modify an existing image that is configured to boot to OOBE

- If you have configured your Windows image to boot to OOBE, but then need to make further configurations to your image in audit mode, you can do one of the following:
  - Use the **CTRL+SHIFT+F3** keyboard shortcut. The computer will reboot into audit mode.

This option may trigger any scripts that you have configured to launch in OOBE.

-or-

2. Mount the image, add an answer file with the **audit** setting, and save it as **C:\test\offline\Windows\Panther\Unattend\Unattend.xml**. This may require overwriting an existing answer file at this location.

On the next boot, Windows will boot directly into audit mode.

## Boot to audit mode automatically from an existing image

1. Create a new answer file, and then add the **Microsoft-Windows-Deployment | Reseal | Mode = audit** setting. Save the answer file as **Unattend.xml**.
2. At an elevated command prompt, mount the Windows image. For example:

```
Dism /Mount-Image /ImageFile:C:\test\images\MyImage.wim /index:<image_index>
/MountDir:C:\test\offline
```

where *<image\_index>* is the number of the selected image on the .wim file.

3. Copy the new answer file to the **C:\test\offline\Windows\Panther\Unattend** folder.

4. Commit the changes, and then unmount the image. For example:

```
Dism /Unmount-Image /MountDir:C:\test\offline /commit
```

When the image is applied to the destination computer and Windows is booted, the computer boots into audit mode automatically, and the **Sysprep** tool appears. For sample procedures, see Step 1: Transfer an image to a different computer and Step 2: Prepare the computer for a customer in [Deployment examples](#).

Options for applying an image also include using answer file settings, such as specifying the image to install and the disk configurations to make on the destination computer. For more information, see the [Unattended Windows Setup Reference Guide](#).

## Deployment examples

To transfer an image to a different computer, you must first remove the computer-specific information from the configured computer by generalizing the image with the **Sysprep** tool. To prepare a computer for the customer, you must generalize the computer, and then set it to boot to Oobe when a customer starts the computer for the first time. In the following examples we create and transfer a reference image to a different computer, and then create a model-specific image that ships to a customer.

### Step 1: Transfer an image to a different computer

1. Install Windows on a reference computer.
2. After the installation is complete, boot the computer and install any additional device drivers or applications.
3. After you update the Windows installation, run **Sysprep**:
  - At the command line, run the **Sysprep /generalize /shutdown** command.

-or-

- In the System Preparation Tool window, select the **Generalize** check box under the **System Cleanup Action** box on the **Shutdown Options** box, select **Shutdown**, and then click **OK**.

**Sysprep** removes system-specific data from the Windows installation. System-specific information includes event logs, unique security IDs (SIDs), and other unique information. After **Sysprep** removes the unique system information, the computer shuts down.

4. After the computer shuts down, insert the Windows PE USB flash drive or other bootable media, and reboot into Windows PE.
5. In the Windows PE session, capture the reference image by using the **Dism /capture-image** command.
6. Proceed to the next step to create a model-specific reference image.

## Step 2: Prepare the computer for a customer

1. Install the reference image you created in Step 1 that is destined for your customer.
2. After you update the Windows installation, at the command line run the **Sysprep /audit /generalize /shutdown** command to configure Windows to boot the computer to audit mode. You can then capture the Windows image by booting to another partition or by using Windows PE.
3. Use the new model-specific reference image to install Windows on a new computer. The Windows image is applied to the computer, and Windows boots to audit mode.
4. (Optional) You can install additional applications and other updates based on a customer's order. You can also test the computer to verify that all components are working correctly.
5. After you update the Windows installation, run the **Sysprep /oobe /shutdown command**.

**Note** If you install Windows images by using the **Sysprep /generalize /oobe** command, the user experience will not be ideal. On the next reboot after you run the **Sysprep /generalize /oobe** command, Windows runs the **specialize** configuration pass, Plug and Play, and other Setup tasks before Windows starts OOB. This process can take additional time and can delay a customer's first logon.

6. Package and deliver the computer to your customer.

When the customer starts the computer, OOB runs.

## Related topics

[Windows Setup Technical Reference](#)

[DISM Image Management Command-Line Options](#)

[Audit Mode Overview](#)

[Add a Driver Online in Audit Mode](#)

[Enable and Disable the Built-in Administrator Account](#)

[Boot from a DVD](#)

[Use a Configuration Set with Windows Setup](#)

[Deploy a Custom Image](#)

[Add Device Drivers to Windows During Windows Setup](#)

[Add a Custom Script to Windows Setup](#)

# Enable and Disable the Built-in Administrator Account

1/18/2019 • 3 minutes to read

When manufacturing PCs, you can use the built-in Administrator account to run programs and apps before a user account is created.

**Note** This topic is about manufacturing PCs. For help with the admin account on your own PC, try one of these pages:

- [Log on as an administrator](#)
- [Delete an account called "Administrator"](#)
- [User Account Control](#)

This account is used when you log into the system by using audit mode, or when you add scripts to the `auditUser` configuration pass.

## Enabling the Built-in Administrator Account

**You can use any of the following methods to enable the built-in Administrator account:**

1. [Use an answer file](#)
2. [Log on by using audit mode](#)
3. [Use the Local Users and Groups MMC \(Server versions only\)](#)

### Use an answer file

You can enable the built-in Administrator account during unattended installations by setting the `AutoLogon` setting to **Administrator** in the Microsoft-Windows-Shell-Setup component. This will enable the built-in Administrator account, even if a password is not specified in the `AdministratorPassword` setting.

You can create an answer file by using Windows® System Image Manager (Windows SIM).

The following sample answer file shows how to enable the Administrator account, specify an Administrator password, and automatically log on to the system.

**Note** Both the Microsoft-Windows-Shell-Setup\ `Autologon` section and the Microsoft-Windows-Shell-Setup\ `UserAccounts` \ `AdministratorPassword` section are needed for automatic logon in audit mode to work. The `auditSystem` configuration pass must include both these settings.

The following XML output shows how to set the appropriate values:

```

<component name="Microsoft-Windows-Shell-Setup" processorArchitecture="x86"
publicToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS"
xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
    <AutoLogon>
        <Password>
            <Value>SecurePasswd123</Value>
            <PlainText>true</PlainText>
        </Password>
        <Username>Administrator</Username>
        <Enabled>true</Enabled>
        <LogonCount>5</LogonCount>
    </AutoLogon>
    <UserAccounts>
        <AdministratorPassword>
            <Value>SecurePasswd123</Value>
            <PlainText>true</PlainText>
        </AdministratorPassword>
    </UserAccounts>
</component>

```

To prevent having to enter a password for the built-in Administrator account after you complete the out-of-box experience, set Microsoft-Windows-Shell-Setup\ UserAccounts \ AdministratorPassword in the **oobeSystem** configuration pass.

The following XML output shows how to set the appropriate values:

```

<UserAccounts>
    <AdministratorPassword>
        <Value>SecurePasswd123</Value>
        <PlainText>true</PlainText>
    </AdministratorPassword>
</UserAccounts>

```

For Windows Server® 2012, the built-in Administrator password must be changed at first logon. This prevents the built-in Administrator account from having a blank password by default.

### **Log on by using audit mode**

If the computer has not yet gone through Out-Of-Box Experience (OOBE), you can enter the built-in Administrator account by re-entering audit mode. For more information, see [Boot Windows to Audit Mode or OOBE](#).

### **Use the Local Users and Groups MMC (server versions only)**

Change the properties of the Administrator account by using the Local Users and Groups Microsoft Management Console (MMC).

1. Open MMC, and then select **Local Users and Groups**.
2. Right-click the **Administrator** account, and then select **Properties**.

The **Administrator Properties** window appears.

3. On the **General** tab, clear the **Account is Disabled** check box.
4. Close MMC.

Administrator access is now enabled.

## **Disabling the Built-in Administrator Account**

For new installations, after the end user creates a user account in OOME, the built-in Administrator account is disabled.

For upgrade installations, the built-in Administrator account remains enabled when there is no other active local administrator on the computer, and when the computer is not joined to a domain.

**Use either of the following methods to disable the built-in administrator account:**

**1. Run the sysprep /generalize command**

When you run the **sysprep /generalize** command, the next time that the computer starts, the built-in Administrator account will be disabled.

**2. Use the net user command**

Run the following command to disable the Administrator account:

```
net user administrator /active:no
```

You can run this command after you configure the computer and before you deliver the computer to a customer.

Original equipment manufacturers (OEMs) and system builders are required to disable the built-in administrator account before delivering the computers to customers. To do this, you can use either of the following methods.

## Configuring the Built-in Administrator Password

### Instructions

- When you run the **sysprep /generalize** command on Windows Server 2012 and Windows Server 2008 R2, the Sysprep tool resets the built-in Administrator account password. The Sysprep tool only clears the built-in Administrator account's password for server editions, not for client editions. The next time that the computer starts, Setup displays a prompt for a password.

**Note** In Windows Server 2012, Windows Server 2008 R2, and Windows Server 2008, the default password policy requires a strong password for all user accounts. To configure a weak password, you can use an answer file that includes the Microsoft-Windows-Shell-Setup\ UserAccounts \ AdministratorPassword setting. You cannot configure a weak password, either manually or by using a script such as the **net user** command.

## Related topics

[Windows Deployment Options](#)

[Audit Mode Overview](#)

# Sysprep (System Preparation) Overview

1/24/2019 • 5 minutes to read

**Sysprep** (System Preparation) prepares a Windows installation (Windows client and Windows Server) for imaging, allowing you to capture a customized installation. **Sysprep** removes PC-specific information from a Windows installation, "generalizing" the installation so it can be installed on different PCs. With **Sysprep** you can configure the PC to boot to audit mode, where you can make additional changes or updates to your image. Or, you can configure Windows to boot to the Out-of-Box Experience (OOBE).

Sysprep is part of the Windows image, and is used during audit mode.

## Feature description

Sysprep provides the following features:

- Removes PC-specific information from the Windows image, including the PC's security identifier (SID). This allows you to capture the image and apply it to other PCs. This is known as generalizing the PC.
- Uninstalls PC-specific drivers from the Windows image.
- Prepares the PC for delivery to a customer by setting the PC to boot to OOBE.
- Allows you to add answer file (unattend) settings to an existing installation.

## Practical applications

Sysprep helps you solve business goals such as:

- Helps you manage multiple PCs by creating a generic image that can be used across multiple hardware designs.
- Deploy PCs by capturing and deploying images with unique security identifiers.
- Fine-tune setup of individual PCs by adding apps, languages, or drivers in audit mode. For more information, see [Audit Mode Overview](#).
- Provide more reliable PCs by testing in audit mode before delivering them to customers.

## New and changed functionality

Beginning with Windows 10, version 1607, Sysprep can be used to prepare an image that has been upgraded. For example:

- You can start with a computer that runs Windows 10, version 1511 or Windows 10, version 1507.
- Upgrade the computer to run Windows 10, version 1607.
- Run Sysprep generalize on the upgraded image, re-capture the updated image, and deploy the image to new devices.

This process allows enterprises to efficiently and continuously roll out up-to-date Windows 10 deployment images.

Beginning with Windows 8.1, the Sysprep user interface is deprecated. The Sysprep UI will continue to be supported in this release however it may be removed in a future release. Update your deployment workflow to use Sysprep from the command line. For more information, see [Sysprep Command-Line Options](#).

## Dependencies

- You must run Windows Setup before you use Sysprep.
- You need a tool to capture an image of the installation, such as [DISM - Deployment Image Servicing and Management Technical Reference for Windows](#) or other disk-imaging software.

**Note** When you copy Windows images between PCs, the reference and destination PCs may not have to have compatible hardware abstraction layers (HALs). The /detecthal option in the Boot Configuration Data (BCD) enables a system that has already run Sysprep to install the correct HAL.

## Limitations

Sysprep has the following limitations:

- The security identifier (SID) is only replaced on the operating system volume when you execute Sysprep. If a single PC has multiple operating systems, you must run Sysprep on each image individually.
- In some cases, customized applications that you install before you recapture the Windows image may require a consistent drive letter. Some applications store paths that include the system's drive letter. Uninstallation, servicing, and repair scenarios may not function correctly if the system's drive letter does not match the drive letter that the application specifies.
- The Plug and Play devices on the reference and destination PCs do not have to be from the same manufacturer. These devices include modems, sound cards, network adapters, and video cards. However, the installation must include the drivers for these devices.
- Not all server roles support Sysprep. If you generalize a Windows Server installation that has specific server roles configured, those server roles may not continue to function after the imaging and deployment process. For more information, see [Sysprep Support for Server Roles](#).
- If you run Sysprep on an NTFS file system partition that contains encrypted files or folders, the data in those folders becomes completely unreadable and unrecoverable.
- The Sysprep tool runs only if the PC is a member of a workgroup, not a domain. If the PC is joined to a domain, Sysprep removes the PC from the domain.
- If a PC is joined to a domain, and the Group Policy of that domain assigns a strong account password policy to the PC, all user accounts will require strong passwords. Running Sysprep or OOBE does not remove the strong password policy.

Always use strong passwords for your user accounts. If you do not assign a strong password to a user account before you run Sysprep or OOBE, you may not be able to log on to the PC.

## Unsupported Scenarios

The following scenarios are not supported:

- Moving or copying a Windows image to a different PC without generalizing the PC is not supported.
- Using a different version of the Sysprep tool to configure an image is not supported. You must use only the version of the Sysprep tool that is installed with the Windows image that you intend to configure. Sysprep is installed with every version of Windows. You must always run Sysprep from the %WINDIR%\system32\sysprep directory.
- If you are using a version of Windows earlier than Windows 10, Version 1607, using the Sysprep tool on upgrade installation types, or to reconfigure an existing installation of Windows that has already been deployed is not supported. In this case, Sysprep must be used only to configure new installations of

Windows. You can run Sysprep an unlimited number of times to build and configure your installation of Windows.

- Automating Sysprep by using a Microsoft-Windows-Deployment\RunSynchronous command is not supported. However, you can use the Microsoft-Windows-Deployment\Generalize setting to prepare the PC for imaging after installation.
- Running VM mode outside a virtual machine (VM) is unsupported. You cannot use VM mode to prepare a VHD for deployment to any PC.
- Sysprep cannot be run under the context of a System account. Running Sysprep under the context of System account by using Task Scheduler or PSEexec, for example, is not supported.

## See also

The following table contains links to resources related to this scenario.

CONTENT TYPE	REFERENCES
<b>Product evaluation</b>	<a href="#">Sysprep Process Overview</a>
<b>Operations</b>	<a href="#">Sysprep (Generalize) a Windows installation</a>   <a href="#">Customize the Default User Profile by Using CopyProfile</a>   <a href="#">Use Answer Files with Sysprep</a>
<b>Tools and settings</b>	<a href="#">Sysprep Command-Line Options</a>   <a href="#">Sysprep Support for Server Roles</a>
<b>Related technologies</b>	<a href="#">Windows Setup</a>   <a href="#">Audit Mode Overview</a>   <a href="#">Boot Windows to Audit Mode or OOB</a>

# Sysprep Process Overview

1/18/2019 • 6 minutes to read

The System Preparation (**Sysprep**) tool is used to change Windows® images from a generalized state to a specialized state, and then back to a generalized state. A generalized image can be deployed on any computer. A specialized image is targeted to a specific computer. You must reseal, or generalize, a Windows image before you capture and deploy the image. For example, when you use the **Sysprep** tool to generalize an image, **Sysprep** removes all system-specific information and resets the computer. The next time that the computer restarts, your customers can add user-specific information through Out-Of-Box Experience (OOBE) and accept the Microsoft Software License Terms.

**Sysprep.exe** is located in the **%WINDIR%\system32\sysprep** directory on all Windows installations.

If you transfer a Windows image to a different computer, you must run the **Sysprep** command together with the **/generalize** option, even if the other computer has the same hardware configuration. The **Sysprep /generalize** command removes unique information from your Windows installation so that you can reuse that image on a different computer. For more information, see [Sysprep \(Generalize\) a Windows installation](#).

## Sysprep Executable

**Sysprep.exe** is the main program that calls other executable files that prepare the Windows installation.

**Sysprep.exe** is located in the **%WINDIR%\system32\sysprep** directory on all Windows installations. If you use the command line instead of the **System Preparation Tool** GUI, you must first close the GUI and then run **Sysprep** from the **%WINDIR%\system32\sysprep** directory. You must also run **Sysprep** on the same version of Windows that you used to install **Sysprep**.

**Important** Beginning with Windows 8.1, the Sysprep user interface is deprecated. The Sysprep UI will continue to be supported in this release however it may be removed in a future release. Update your Windows deployment workflow to use the Sysprep command line. For more information about the Sysprep Command line tool, see [Sysprep Command-Line Options](#).

## Sysprep Process Overview

When Sysprep runs, it goes through the following process:

1. **Sysprep verification.** Verifies that Sysprep can run. Only an administrator can run Sysprep. Only one instance of Sysprep can run at a time. Also, Sysprep must run on the version of Windows that you used to install Sysprep.
2. **Logging initialization.** Initializes logging. For more information, see [Sysprep Log Files](#).
3. **Parsing command-line arguments.** Parses command-line arguments. If a user does not provide command-line arguments, a System Preparation Tool window appears and enables users to specify Sysprep actions.
4. **Processing Sysprep actions.** Processes Sysprep actions, calls appropriate .dll files and executable files, and adds actions to the log file.
5. **Verifying Sysprep processing actions.** Verifies that all .dll files have processed all their tasks, and then either shuts down or restarts the system.

## Persisting the Hardware Configuration

If you create an image of this installation for deployment to a different computer, you must run the **Sysprep** command together with the **/generalize** option, even if the other computer has the identical hardware configuration. The **Sysprep /generalize** command removes unique information from a Windows installation so that you can reuse that image on different computers. The next time that you boot the Windows image, the **specialize** configuration pass runs.

If you want to install a Windows image to computers that have the same hardware configuration, you can preserve the device-drivers installation in a Windows image. To do this, in your answer file, specify the **PersistAllDeviceInstalls** setting in the **Microsoft-Windows-PnP\Sysprep** component. The default value is **false**. If you set the setting to **true**, the Plug and Play devices remain on the computer during the **generalize** configuration pass. You do not have to reinstall these devices during the **specialize** configuration pass. For more information, see [Use Answer Files with Sysprep](#) and Unattended Windows Setup Reference Guide.

## Adding Device Drivers

Plug and Play devices include modems, sound cards, network adapters, and video cards. The Plug and Play devices on the reference and destination computers do not have to come from the same manufacturer. However, you must include the drivers for these devices in the installation. For more information, see [Add and Remove Drivers to an Offline Windows Image](#) and [Add Device Drivers to Windows During Windows Setup](#).

## Booting to Audit Mode or OOBE

When Windows boots, the computer can start in one of two modes:

- OOBE

OOBE, also named the out-of-box experience (OOBE), is the first user experience. The OOBE enables end users to customize their Windows installation. End users can create user accounts, read and accept the Microsoft® Software License Terms, and select their language and time zones. By default, all Windows installations boot to OOBE first. The [oobeSystem](#) configuration pass runs immediately before OOBE starts.

If you do not automatically activate Windows by using a product key, OOBE prompts the user for a product key. If the user skips this step during OOBE, Windows reminds the user to enter a valid product key later. To automatically activate Windows by using a product key, specify a valid product key in the **Microsoft-Windows-Shell-Setup\ProductKey** unattend setting during the **specialize** configuration pass. For more information, see [Work with Product Keys and Activation](#).

- Audit Mode

Audit mode enables you to add customizations to Windows images. Audit mode does not require that you apply settings in OOBE. By bypassing OOBE, you can access the desktop more quickly and perform your customizations. You can add more device drivers, install applications, and test the validity of the installation.

You can configure Windows to boot directly to audit mode by using the **Microsoft-Windows-Deployment | Reseal | Mode** setting in an answer file. In audit mode, the computer processes settings in an unattended answer file in the [auditSystem](#) and [auditUser](#) configuration passes.

If you are running a computer in audit mode to configure the installation to boot to OOBE, either use the **Sysprep** GUI or run the **Sysprep /oobe** command. To prepare a computer for an end user, you must configure the computer to boot to OOBE when an end user starts the computer for the first time. In a default Windows installation, OOBE starts after installation is completed, but you can skip OOBE and boot directly to audit mode to customize images.

For more information, see:

- [Audit Mode Overview](#)

- [Boot Windows to Audit Mode or OOB](#)
- [How Configuration Passes Work](#)
- [Enable and Disable the Built-in Administrator Account](#)
- [Add a Driver Online in Audit Mode](#)

## Detecting the State of a Windows Image

You can use Sysprep to identify the state of a Windows image. That is, you can determine whether the image will boot to audit mode or OOB, or if the image is still in the process of installation. For more information, see [Windows Setup Installation Process](#).

## Sysprep Log Files

The **Sysprep** tool logs Windows Setup actions in different directories, depending on the configuration pass. Because the **generalize** configuration pass deletes certain Windows Setup log files, the **Sysprep** tool logs generalize actions outside the standard Windows Setup log files. The following table shows the different log file locations that **Sysprep** uses.

ITEM	LOG PATH
<b>Generalize</b>	%WINDIR%\System32\Sysprep\Panther
<b>Specialize</b>	%WINDIR%\Panther
Unattended Windows Setup actions ( <b>OOBE</b> )	%WINDIR%\Panther\Unattendgc

The **setupact.log** file is the main log file.

For more information, see [Deployment Troubleshooting and Log Files](#).

## Creating and Using Sysprep Providers

Independent software vendors (ISVs) and independent hardware vendors (IHVs) can create **Sysprep** providers that enable their applications to support imaging and deployment scenarios. If an application does not currently support generalize operations by using the **Sysprep** tool, you can create a provider that removes all software-specific and hardware-specific information from the application.

To create a **Sysprep** provider, you must do the following:

1. Determine which configuration pass (**cleanup**, **generalize**, or **specialize**) your Sysprep provider addresses.
2. Create the appropriate entry point for your **Sysprep** provider, based on your choice of configuration pass.
3. Register the **Sysprep** provider for use by the **Sysprep** tool.
4. Test your **Sysprep** provider to validate that the provider functions correctly. Make sure that you review the log files for warnings and errors.

For more information about **Sysprep** providers, see the [System Preparation \(Sysprep\) Tool Provider Developer's Guide](#).

## Related topics

[Sysprep \(System Preparation\) Overview](#)

[Sysprep Command-Line Options](#)

[Sysprep \(Generalize\) a Windows installation](#)

[Sysprep Support for Server Roles](#)

[Use Answer Files with Sysprep](#)

# Sysprep (Generalize) a Windows installation

1/18/2019 • 4 minutes to read

To deploy a Windows image to different PCs, you have to first generalize the image to remove computer-specific information such as installed drivers and the computer security identifier (SID). You can either use **Sysprep** by itself or Sysprep with an [unattend](#) answer file to generalize your image and make it ready for deployment.

## Generalize a Windows installation

When you generalize a Windows image, Windows Setup processes settings in the [generalize](#) configuration pass. Even if you're capturing an image that's going to be deployed to a PC with similar hardware, you still have to generalize the Windows installation to remove unique PC-specific information from a Windows installation, which allows you to safely reuse your image.

When you generalize an image, Windows replaces the computer SID only on the operating system volume where you ran Sysprep. If a single computer has multiple operating systems, you must run **Sysprep** on each image individually.

If you're generalizing a Windows Server installation that has Remote Authentication Dial-In User Service (RADIUS) clients or remote RADIUS server groups defined in the Network Policy Server (NPS) configuration, you should remove this information before you deploy it to a different computer. For more information, see [Prepare a Network Policy Server \(NPS\) for Imaging](#).

### Prevent Sysprep from removing installed devices

When you set up a Windows PC, Windows Setup configures all detected devices. Generalizing a Windows installation uninstalls configured devices, but does not remove device drivers from the PC.

If you're deploying an image to computers that have identical hardware and devices as the original PC, you can keep devices installed on the computer during system generalization by using an unattend file with Microsoft-Windows-PnP\Sysprep | `PersistAllDeviceInstalls` set to **true**. For more information about **Sysprep**-related Windows unattend components, see the [Unattended Windows Setup Reference for Microsoft-Windows-PnpSysprep](#).

### Limits on how many times you can run Sysprep

You can run the **Sysprep** command up to 8 times on a single Windows image. After running Sysprep 8 times, you must recreate your Windows image. In previous versions of Windows, you could use the `SkipRearm` answer file setting to reset the Windows Product Activation clock when running Sysprep. If you are using a volume licensing key or a retail product key, you don't have to use `SkipRearm` because Windows is automatically activated.

### Microsoft Store apps

Installing new Microsoft Store apps or updating your existing Microsoft Store apps before generalizing a Windows image will cause Sysprep to fail. `Sysprep /generalize` requires that all apps are provisioned for all users; however, when you update an app from the Microsoft Store, that app becomes tied to the logged in user account. The following error appears in the Sysprep log files (located at %WINDIR%\System32\Sysprep\Panther):

```
<package name> was installed for a user, but not provisioned for all users. This package will not function properly in the sysprep image.
```

Instead of using the Microsoft Store to update your apps, you should sideload updates to your line-of-business apps, provision offline-licensed Microsoft Store for Business apps for all users, or have end-users update their

apps by using the Microsoft Store on their destination PCs. If Microsoft Store access in a managed environment is disabled by an IT administrator, end-users will not be able to update the Microsoft Store apps.

For more information about sideloading line-of-business Microsoft Store apps, see [Sideload Apps with DISM](#) and [Customize the Start Screen](#).

## Generalize an image

### Generalize from Audit Mode

To generalize an image, you have to first boot into Audit Mode. You can boot into Audit Mode using an unattend file or from the Out-Of-Box Experience (OOBE) screen. You can read about the different ways of booting into Audit Mode at [Boot Windows to Audit Mode or OOBE](#).

1. Boot a PC into Audit Mode. When Windows boots into Audit Mode, **System Preparation Tool** will appear on the desktop. You can choose to either close the **System Preparation Tool** window or allow it to remain open.
2. Customize Windows by adding drivers, changing settings, and installing programs. Do not install any Microsoft Store apps using the Microsoft Store.
3. Run Sysprep.
  - If the **System Preparation Tool** window is still open, click **Generalize**, click **Shutdown**, and then click **OK** to generalize the image and shut down the PC.  
-or-  
• Use Sysprep from Command Prompt. Run `%WINDIR%\system32\sysprep\sysprep.exe` to open the **System Preparation Window**. You can also use the `Sysprep` command together with the `/generalize`, `/shutdown`, and `/oobe` options. See [Sysprep command-line options](#) to see available options.

```
%WINDIR%\system32\sysprep\sysprep.exe /generalize /shutdown /oobe
```

#### NOTE

If you are generalizing a VHD that will be deployed as a VHD on the same virtual machine or hypervisor, use the `/mode:vm` option with the Sysprep command-line.

The computer generalizes the image and shuts down.

4. After the computer shuts down, [capture your image with DISM](#).
5. Deploy this image to a reference computer. When the reference computer boots, it displays the OOBE screen.

### Generalize using unattend

If you use multiple unattend files during your computer deployment, you can add the following settings to your each of your unattend files so Windows Setup will generalize the PC after processing the unattend file.

- To automatically generalize the image and shut down, use the Microsoft-Windows-Deployment | `Generalize` setting. Set `Mode` to **OOBE** or **Audit**, and set `ForceShutdownNow` to **true**.  
-or-  
• To generalize the system, and have it boot into Audit Mode, use the Microsoft-Windows-Deployment |

`Reseal` setting to the `oobeSystem` configuration pass. Set `Mode` to **Audit**.

## Related topics

[Sysprep Process Overview](#)

[Sysprep Command-Line Options](#)

[Sysprep Support for Server Roles](#)

[Work with Product Keys and Activation](#)

# Use Answer Files with Sysprep

1/18/2019 • 4 minutes to read

You can use an answer file together with the System Preparation (**Sysprep**) tool to configure unattended Windows Setup settings. This topic describes some considerations and processes for using answer files together with **Sysprep**. For more information about Windows components and settings that you can add to an answer file, see the [Unattended Windows Setup Reference](#).

## Running Sysprep an Unlimited Number of Times

If you specify a product key, Windows is automatically activated, and you can run the **Sysprep** command an unlimited number of times. To automatically activate Windows by supplying a product key, specify a valid product key in the Microsoft-Windows-Shell-Setup\ `ProductKey` unattend setting during the `specialize` configuration pass. If you don't automatically activate Windows by providing a product key, Windows prompts the end user for a product key.

## Applying Settings in the generalize, auditSystem, and auditUser Configuration Passes

Not all configuration passes run during Windows Setup. The `generalize`, `auditSystem`, and `auditUser` configuration passes are available only when you run **Sysprep**.

If you add settings to your answer file in these configuration passes, you must run **Sysprep** to apply these settings as follows:

- To apply the settings in the `auditSystem` and `auditUser` configuration passes, you must boot in audit mode by using the **Sysprep/audit** command.
- To apply the settings in the `generalize` configuration pass, you must use the **Sysprep/generalize** command. The `generalize` configuration pass removes the system-specific settings so that you can deploy the same image on multiple computers.

For more information, see [How Configuration Passes Work](#).

## Caching Answer Files to the Computer

If you install Windows by using an answer file, that answer file is cached to the system. When later configuration passes run, the computer applies settings in that answer file to the system. Because this answer file is cached, when you run the **Sysprep** command, the system applies settings in the cached answer file. If you use the settings in a different answer file, you can specify a separate Unattend.xml file by using the **Sysprep /unattend:<file\_name>** option. For more information, see [Sysprep Command-Line Options](#). For more information about how to use an implicit answer-file search, see [Windows Setup Automation Overview](#).

## Persisting Plug and Play Device Drivers During the `generalize` Configuration Pass

You can persist device drivers when you run the **Sysprep** command together with the `/generalize` option. To do this, specify the `PersistAllDeviceInstalls` setting in the Microsoft-Windows-PnP\Sysprep component. During the `specialize` configuration pass, Plug and Play scans the computer for devices, and then installs device drivers for the detected devices. By default, the computer removes these device drivers from the system when you generalize the

system. If you set the Microsoft-Windows-PnP\sysprep\ PersistAllDeviceInstalls setting to **true** in an answer file, Sysprep doesn't remove the detected device drivers.

## Displaying RunSynchronous Actions in an Answer File

In audit mode, you can view the status for Microsoft-Windows-Deployment\ RunSynchronous commands that run during the **auditUser** configuration pass. The **AuditUI** window displays the status for commands and provides these:

- Visual progress to indicate that an installation is continuing and not suspended.
- Visual indication of when and where failures occur. This provides a quick diagnosis if the command doesn't create log files.

If the answer file contains Microsoft-Windows-Deployment\ RunSynchronous commands in the **auditUser** configuration pass, a list of the commands appears in the **AuditUI** window. The commands appear in the order that the Microsoft-Windows-Deployment\ RunSynchronous\ RunSynchronousCommand\ Order setting specifies. Each list item in the user interface is the string from one of these:

- Microsoft-Windows-Deployment\ RunSynchronous\ RunSynchronousCommand\ Description (if present)
- Microsoft-Windows-Deployment\ RunSynchronous\ RunSynchronousCommand\ Path

**Sysprep** processes all RunSynchronous commands in order. If the command succeeds, its related list item receives a green check-mark annotation. If the command fails, its related list item receives a red X annotation. If the command requests a reboot, the **AuditUI** window appears after the boot, but only unprocessed list items appear. Previously processed items no longer appear in the **AuditUI** window. If the list of items in the **AuditUI** window exceeds the height of the display, the list is truncated to the display and doesn't scroll. As a result, you may not be able to see some items.

Windows Setup interprets the return codes as status values in the **AuditUI** window. A zero value indicates a success. A nonzero value indicates a failure. The return value of the command might affect the behavior of Windows Setup, depending on the value of the Microsoft-Windows-Deployment\ RunSynchronous\ RunSynchronousCommand\ WillReboot setting.

If the **WillReboot** command is set to **Always**:

- If the command returns 0, its related list item receives a green check-mark annotation. A reboot immediately occurs.
- If the command returns a nonzero number, its related list item receives a red X annotation. A reboot immediately occurs. A nonzero return value isn't treated as a fatal error when **WillReboot** is set to either **Always** or **Never**.

If the **WillReboot** command is set to **Never**:

- If the command returns 0, its related list item receives a green check-mark annotation.
- If the command returns a nonzero number, its related list item receives a red X annotation. A nonzero return value isn't treated as a fatal error when **WillReboot** is set to either **Always** or **Never**.

If the **WillReboot** command is set to **OnRequest**:

- If the command returns 0, its related list item receives a green check-mark annotation.
- If the command returns 1, its related list item receives a green check-mark annotation. A reboot immediately occurs.
- If the command returns 2, its related list item temporarily receives a green check-mark annotation. A reboot

immediately occurs. After the reboot, the related list item appears again in the **AuditUI** window without annotation because the command is still in process.

- If the command returns other values, a fatal error occurs and a blocking dialog box appears. If the Errorhandler.cmd file is present, no dialog box appears. For more information about the Errorhandler.cmd file, see [Add a Custom Script to Windows Setup](#).

## Related topics

[Sysprep \(System Preparation\) Overview](#)

[Sysprep Command-Line Options](#)

[Sysprep Support for Server Roles](#)

[Sysprep Process Overview](#)

[Deployment Troubleshooting and Log Files](#)

# Sysprep Command-Line Options

1/18/2019 • 3 minutes to read

Run **Sysprep** to prepare a Windows installation to be captured. This topic describes the command-line syntax for the System Preparation (Sysprep) tool.

If you intend to create an image of an installation for deployment to a different computer, you must run the **Sysprep** command together with the **/generalize** option, even if the other computer has the same hardware configuration. The **Sysprep /generalize** command removes unique information from your Windows installation so that you can safely reuse that image on a different computer. The next time that you boot the Windows image, the **specialize** configuration pass runs.

## Sysprep Command-Line Options

The following command-line options are available for Sysprep:

**Sysprep.exe [/oobe | /audit]**

**[/generalize]**

**[/mode:vm]**

**[/reboot | /shutdown | /quit]**

**[/quiet]**

**[/unattend:<answerfile>]**

The following table lists Sysprep command-line options:

OPTION	DESCRIPTION
<b>/audit</b>	Restarts the computer into audit mode. Audit mode enables you to add additional drivers or applications to Windows. You can also test an installation of Windows before you send the installation to an end user. For example:  <div style="border: 1px solid #ccc; padding: 5px; text-align: center;"><pre>Sysprep /audit</pre></div> If you specify an answer file, the audit mode of Windows Setup runs the <b>auditSystem</b> and <b>auditUser</b> configuration passes.

OPTION	DESCRIPTION
<b>/generalize</b>	<p>Prepares the Windows installation to be imaged. Sysprep removes all unique system information from the Windows installation. Sysprep resets the security ID (SID), clears any system restore points, and deletes event logs. For example:</p> <pre data-bbox="858 384 1191 413">Sysprep /generalize /shutdown</pre> <p>The next time that the computer starts, the <b>specialize</b> configuration pass runs. The configuration pass creates a new security ID (SID).</p>
<b>/oobe</b>	<p>Restarts the computer into OOBE mode. For example:</p> <pre data-bbox="858 698 1263 727">Sysprep /generalize /shutdown /oobe</pre> <p>OOBE enables end users to customize their Windows operating system, create user accounts, name the computer, and perform other tasks. Sysprep processes any settings in the <b>oobeSystem</b> configuration pass in an answer file before OOBE starts.</p>
<b>/mode:vm</b>	<p>Generalizes a Virtual Hard Disk (VHD) so that you can deploy the VHD as a VHD on the same Virtual Machine (VM) or hypervisor. After the VM restarts, the VM can boot to OOBE. For example:</p> <pre data-bbox="858 1170 1250 1199">Sysprep /generalize /oobe /mode:vm</pre> <p>The only additional switches that apply to VM mode are <b>/reboot</b>, <b>/shutdown</b>, and <b>/quit</b>. You must deploy the VHD on a Virtual Machine (VM) or hypervisor with the same hardware profile. For example, if you created VHD in Microsoft Hyper-V, you can only deploy your VHD to Microsoft Hyper-V VMs with a matching hardware profile. Deploying the VHD to a different VM with a different hardware profile might cause unexpected issues.</p> <div data-bbox="850 1551 958 1581" style="border: 1px solid black; padding: 2px;"><b>Important</b></div> <div data-bbox="850 1590 1287 1619" style="border: 1px solid black; padding: 2px;">You can only run VM mode from inside a VM.</div>
<b>/reboot</b>	<p>Restarts the computer. You can use this option to audit the computer and to verify that the first-run experience operates correctly.</p>
<b>/shutdown</b>	<p>Shuts down the computer after the <b>Sysprep</b> command finishes running.</p>

OPTION	DESCRIPTION
<b>/quiet</b>	Runs the Sysprep tool without displaying on-screen confirmation messages. You can use this option if you automate the Sysprep tool.
<b>/quit</b>	Closes the Sysprep tool without rebooting or shutting down the computer after Sysprep runs the specified commands.
<b>/unattend:&lt;answerfile&gt;</b>	Applies settings in an answer file to Windows during an unattended installation, where <answerfile> specifies the path and file name of the answer file to use. For example:  <div style="border: 1px solid black; padding: 5px; margin-left: 20px;"> <pre>Sysprep /audit /reboot /unattend:F:\Unattend.xml</pre> </div> <p>where F is the drive letter of the portable storage device on which the answer file (Unattend.xml) is located.</p>

### Important

You must use the **Sysprep /generalize** command to generalize a complete Windows installation before you can use the installation for deployment to a new computer, whether you use imaging, hard disk duplication, or another method. Moving or copying a Windows image to a different computer without running the **Sysprep /generalize** command is not supported.

## Related topics

[Sysprep \(System Preparation\) Overview](#)

[Sysprep Process Overview](#)

[Sysprep \(Generalize\) a Windows installation](#)

[Sysprep Support for Server Roles](#)

[Use Answer Files with Sysprep](#)

# Sysprep Support for Server Roles

1/18/2019 • 2 minutes to read

Many common server roles support the System Preparation tool (Sysprep). However, if you run the **Sysprep** command together with the **/generalize** option against an installation of a server, and you are using an unsupported server role, those roles may not function after the imaging and deployment process is completed. Therefore, you must enable and configure any server roles that do not support Sysprep after you have performed the imaging and deployment process.

The following table lists server roles and specifies whether the roles support Sysprep.

SERVER ROLE	SYSPREP SUPPORT IN WINDOWS SERVER 2008	SYSPREP SUPPORT IN WINDOWS SERVER 2008 R2	SYSPREP SUPPORT IN WINDOWS SERVER® 2012
Active Directory Certificate Services (AD CS)	No	No	No
Active Directory Domain Services (AD DS)	No	No	No
Active Directory Federation Services (AD FS)	No	No	No
Active Directory Lightweight Directory Services (AD LDS)	No	No	No
Active Directory Rights Management Services (AD RMS)	No	No	No
Application Server	Yes	Yes	Yes
Dynamic Host Configuration Protocol (DHCP) Server	Yes	No	No
Domain Name System (DNS) Server	Not applicable	Not applicable	Not applicable
Fax Server	No	No	No
File and Storage Services	No	Yes	Yes

SERVER ROLE	SYSprep SUPPORT IN WINDOWS SERVER 2008	SYSprep SUPPORT IN WINDOWS SERVER 2008 R2	SYSprep SUPPORT IN WINDOWS SERVER® 2012
Hyper-V™	Not applicable	Yes  Not supported for a virtual network on Hyper-V™. You must delete any virtual networks before you run the Sysprep tool.	Yes  Not supported for a virtual network on Hyper-V™. You must delete any virtual networks before you run the Sysprep tool.
Network Policy and Access Services (NPAS) <sup>1</sup>	No	No	No
Network Policy Routing and Remote Access Services	Yes	Not applicable	Not applicable
Printing and Document Services (Print Services) <sup>2</sup>	No	Yes	Yes
Remote Desktop Services <sup>3</sup>	Yes	Yes	Yes
Streaming Media Services (available as a download)	Not applicable	Not applicable	Not applicable
UDDI Services <sup>4</sup>	No	Not applicable	Not applicable
Volume Activation Services <sup>5</sup>	Not applicable	Not applicable	Not applicable
Web Server (Internet Information Services)	Yes  Not supported with encrypted credentials in the Applicationhost.config file.	Yes  Not supported with encrypted credentials in the Applicationhost.config file.	Yes  Not supported with encrypted credentials in the Applicationhost.config file.
Windows Deployment Services	No	No	Yes  Not supported if Windows Deployment Services is initialized. <sup>6</sup>
Windows Server Update Services (WSUS)	No	No	No

<sup>(1)</sup> NPAS includes Health Registration Authority (HRA), Network Policy Server (NPS), and Host Credential

Authorization Protocol (HCAP).

<sup>(2)</sup> In Windows Server 2008 R2, Print Services was renamed Printing and Document Services.

<sup>(3)</sup> In Windows Server 2008 R2, Terminal Services was renamed Remote Desktop Services, which is also known as Remote Desktop Session Host.

<sup>(4)</sup> UDDI Services was not included in Windows Server 2008 R2.

<sup>(5)</sup> Volume Activation Services is new for Windows Server 2012.

<sup>(6)</sup> You must uninitialized the server that has the Windows Deployment Services role installed before you run Sysprep. You can uninitialized a server by using the **wdsutil /uninitialize-server** command.

## Related topics

[Sysprep \(System Preparation\) Overview](#)

[Sysprep Process Overview](#)

[Sysprep Command-Line Options](#)

[Sysprep \(Generalize\) a Windows installation](#)

# Modify a Windows image using DISM

1/18/2019 • 5 minutes to read

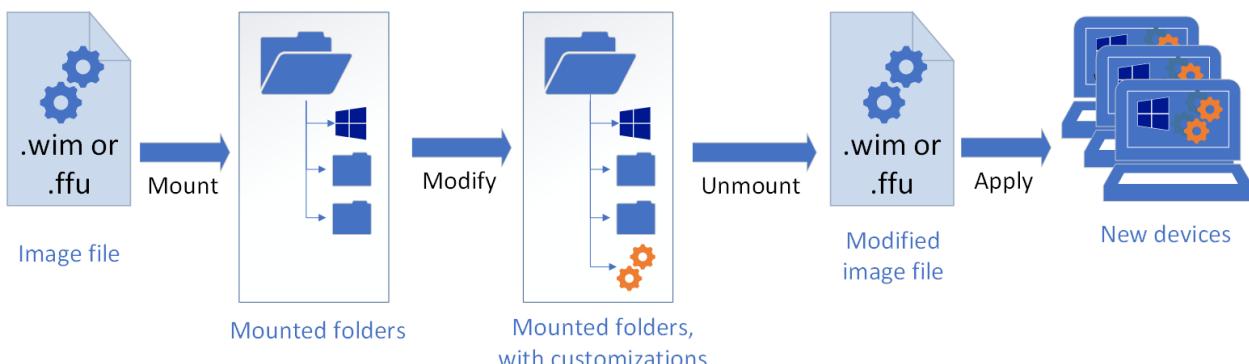
You can make changes to offline mounted or applied Windows images without booting into the operating system you're going to modify.

**Mounted** images are WIM, VHD, or FFU files that have their contents mapped to a folder. Changes to mounted images are made from either a Windows 10 technician PC, or from WinPE. You run DISM commands against a mounted image, as well as run common file operations such as copying, pasting, and renaming on a mounted image. To save changes you make to the image, use the `/commit` option when you use DISM to unmount the image. To make changes to a mounted image, use `DISM /image:`.

**Applied** images are WIM, VHD, or FFU image files that have been applied to a specified partition. Offline changes to an applied image are usually performed from WinPE. To make changes to an applied image, use `DISM /image:`.

When you apply an image that you're going to recapture, apply the image to the root folder of a drive. If you recapture an image that wasn't applied to the root of a drive, the image will inherit the parent folder's security descriptors and might not be the same as what would be captured if the image was applied to the root of a drive. See [Applying an image](#) to learn how to apply an image.

You can mount and modify multiple images on a single computer. For more information, see [Deployment Image Servicing and Management \(DISM\) Best Practices](#).



## Mount an image

You can mount an image using the `/optimize` option to reduce initial mount time. However, when using the `/optimize` option, processes that are ordinarily performed during a mount will instead be completed the first time that you access a directory. As a result, there may be an increase in the time that is required to access a directory for the first time after mounting an image using the `/optimize` option.

1. Open a command prompt with administrator privileges.

### NOTE

If you are using a version of Windows other than Windows 8 or Windows 10, use the Deployment Tools Command Prompt that gets installed with the ADK.

2. Use DISM to mount the image

```
DISM /Mount-image /imagefile:<path_to_Image_file> {/Index:<image_index> | /Name:<image_name>}  
/MountDir:<target_mount_directory> [/readonly] [/optimize]
```

#### NOTE

To mount a Windows image from a VHD or FFU file, you must specify `/index:1`.

For more information about the options available for the **/Mount-Image** option in DISM, see [DISM Image Management Command-Line Options](#).

## Modify an image

After you mount an image, you can use DISM to add and remove drivers, packages, language packs, enumerate drivers and packages, modify configuration settings, and more.

### View and modify an image

You can create, view, and edit files on a mounted image, just as you would any other file on your PC. When you modify the files in a mounted image, those file changes get saved in the image. Note that although you can add application files and folders, you can't install applications directly into a mounted image in the same way that you would on a running PC. If you must add an application or a device, verify that you included all of the required files.

#### Add and remove drivers

See [Add and remove drivers to an offline Windows image](#) to learn how to work with drivers. Note that the DISM driver commands will only run against an offline image.

#### Add and remove packages

See [Add or remove packages offline using DISM](#) to learn how to work with packages.

#### Add or remove languages

See [Add and remove language packs offline using DISM](#) to learn how to work with languages.

#### Upgrade to a higher Windows edition

Any changes you make to a mounted image are also applied to each potential target edition of Windows. Each target edition is staged in the image. The changes will not be lost when you upgrade to a higher edition of Windows.

See [Change the Windows image to a higher edition using DISM](#) to learn how to change editions.

#### Reduce the size of an image

You can use DISM to reduce the footprint of a Windows image by cleaning up superseded components and resetting the base of the superseeded components.

- At an elevated command prompt, run the following command to reduce the size of the image file:

```
Dism /Image:C:\test\offline /cleanup-image /StartComponentCleanup /ResetBase
```

## Commit changes to an image

You can commit changes to an image without unmounting the image.

- At the administrator command prompt, type:

```
Dism /Commit-Image /MountDir:C:\test\offline
```

Use **/CheckIntegrity** to detect and track .wim file corruption when you commit changes to the image. When you apply or mount the image, use **/CheckIntegrity** again to stop the operation if file corruption was detected. **/CheckIntegrity** cannot be used with virtual hard disk (VHD) files.

## Unmounting an image

After you modify a mounted image, you must unmount it. If you mounted your image with the default read/write permissions, you can commit your changes. This makes your modifications a permanent part of the image.

If you modified an applied image, you don't have to do anything else. You'll see any changes you made when you boot the PC.

1. Open a command prompt or the Deployment and Imaging Tools Environment with administrator privileges.
2. Unmount the image.

```
Dism /Unmount-image /MountDir:<target_mount_directory> {/Commit | /Discard}
```

where `C:\test\offline` is the location of the mount directory. If you do not specify the parameters to unmount, this option lists all of the mounted images but does not perform the unmount action.

### IMPORTANT

You must use either the **/commit** or **/discard** argument when you use the **/unmount** option.

## Troubleshooting

See [Repair a Windows Image](#).

### If the DISM commands in this topic fail, try the following:

1. Make sure that you are using the Windows 10 version of DISM that is installed with the Windows ADK.
2. Don't mount images to protected folders, such as your User\Documents folder.
3. If DISM processes are interrupted, consider temporarily disconnecting from the network and disabling virus protection.
4. If DISM processes are interrupted, consider running the commands from the Windows Preinstallation Environment (WinPE) instead.

## Related topics

[DISM Image Management Command-Line Options](#)

[Service a Windows Image Using DISM](#)

# Repair a Windows Image

1/18/2019 • 2 minutes to read

Repair a Windows image using DISM. You can repair offline Windows image in a WIM or VHD file, or an online Windows image. An online Windows image will also attempt to repair itself if it becomes unserviceable. The repair source for this operation is the same source that is used for Features on Demand and is determined by Group Policy settings. For more information, see [Configure a Windows Repair Source](#). When you use the DISM tool to repair an online or offline image, you can use the */Source* argument with the */RestoreHealth* argument to specify additional repair source locations to use to search for the required files.

For a quick check of an online image, you may be able to use the command: `sfc /scannow` to scan and repair files.

For a more extensive check that can repair issues with the store, use `DISM /Cleanup-Image`.

## To check if an image is repairable

1. Scan the image to check for corruption. This operation will take several minutes. For example, at a command prompt, type the following command:

```
Dism /Online /Cleanup-Image /ScanHealth
```

2. Check the image to see whether any corruption has been detected. For example, at a command prompt, type:

```
Dism /Online /Cleanup-Image /CheckHealth
```

When you use the */CheckHealth* sfc argument, the DISM tool will report whether the image is healthy, repairable, or non-repairable. If the image is non-repairable, you should discard the image and start again. If the image is repairable, you can use the */RestoreHealth* argument to repair the image.

## To repair an image

- Use the */RestoreHealth* argument to repair the image. For example, to repair an offline image using a mounted image as a repair source, at a command prompt, type the following command:

```
Dism /Image:C:\offline /Cleanup-Image /RestoreHealth /Source:c:\test\mount\windows
```

Or to repair an online image using some of your own sources instead of Windows Update, type:

```
Dism /Online /Cleanup-Image /RestoreHealth /Source:c:\test\mount\windows /LimitAccess
```

If you do not specify a */Source* for the repair files, the default location for Features on Demand is used. For more information, see [Configure a Windows Repair Source](#). If you specify more than one */Source*, the files are copied from the first location where they are found and the rest of the locations are ignored. You can use */LimitAccess* to prevent the DISM tool from using Windows Update as a repair source or as a backup repair source for online images.

## Repairing images during servicing

In some cases, an image can be corrupted while modifying it with DISM. Use */Cleanup-MountPoints* to repair it. This command will not unmount images that are already mounted, nor will it delete images that can be recovered using the */Remount-Image* command.

```
Dism /Cleanup-Mountpoints
```

## Related topics

[Use the System File Checker tool to repair missing or corrupted system files](#)

[DISM Operating System Package Servicing Command-Line Options](#)

[Configure a Windows Repair Source](#)

# Configure a Windows Repair Source

1/18/2019 • 4 minutes to read

You can use Group Policy to specify a Windows image repair source to use within your network. The repair source can be used to restore Windows features or to repair a corrupted Windows image.

Features on demand enables you to remove an optional feature from a Windows® image and then restore it later. You can disable optional features and remove files associated with those features from a Windows image using the Deployment Image Servicing and Management (DISM) tools. When you use the **/Remove** argument with the DISM **/Disable-Feature** option, the manifest files for the feature or Server role is maintained in the image. However, all other files for the feature are removed. This enables you to store, download, and deploy smaller images without losing features. Once the image has been deployed, users can enable the feature on their computers at any time by using features on demand to retrieve the required files from the repair source. For more information, see [Enable or Disable Windows Features Using DISM](#).

Automatic corruption repair provides files to repair Windows if the operating system has become corrupted. Users can also use a specified repair source on your network or use Windows Update to retrieve the source files that are required to enable a feature or to repair a Windows image. For more information, see [Repair a Windows Image](#).

## Choose a Repair Source

You can use Windows Update to provide the files that are required to restore a Windows feature or repair a corrupted operating system. You can also configure Group Policy to gather the required files from a network location. Multiple source locations can be specified in the Group Policy.

### Use Windows Update to restore optional features and repair Windows images

1. Windows Update will be used by default if it is allowed by the policy settings on the computer.
2. If you want to use Windows Update as a primary or backup source for files that are used to restore optional features or repair Windows images, you should make sure that your firewall is configured to allow access to Windows Update.

### Use a network location to restore optional features and repair Windows images

1. You can use a mounted Windows image from a WIM file as a source to restore optional features and repair a corrupted operating system. For example, c:\test\mount\Windows. For more information about capturing a Windows image as a WIM file, see [Capture Images of Hard Disk Partitions Using DISM](#).
2. You can use a running Windows installation as a source to restore optional features by sharing the c:\Windows folder on your network.
3. You can use a Windows side-by-side folder from a network share or from a removable media, such as the Windows DVD, as the source of the files. For example, z:\sources\SxS.
4. You can use a Windows image (.wim) file on a network share as a source to restore optional features. You must specify the index of the Windows image in the .wim file that you want to use and you must use a **Wim:** prefix in the path to identify this file format. For example, to specify index 3 in a file named contoso.wim, type: Wim:\\network\\images\\contoso.wim:3.

#### **NOTE**

When specifying a source, only use RTM media regardless of whether the source is a WIM or a mounted Windows image. Refresh media has older file versions excluded from the media and the target operating system may need these files. Also, make sure the source is patched to the latest Cumulative Update. If the target OS is patched to a higher level than the source, adding features or repairing Operating Systems may fail because the target OS needs updated files that are not present in the source.

## Set Group Policy

You can use Group Policy to specify when to use Windows Update, or a network location as a repair source for features on demand and automatic corruption repair.

### **Configure Group Policy for Feature on Demand**

1. Open the group policy editor. For example, on a computer that is running Windows 10, from the **Start** screen, type **Edit Group Policy**, and then select **Edit Group Policy** to open the Group Policy Editor.
2. Click **Computer Configuration**, click **Administrative Templates**, click **System**, and then double-click the **Specify settings for optional component installation and component repair** setting.
3. Select the settings that you want to use for Features on Demand.

## Maintaining a Repair Source

If you do not use Windows Update as the repair source for features on demand and automatic corruption repair, you should consider the following guidelines for maintaining a repair source.

### **Servicing updates**

You should keep any repair source current with the latest servicing updates. If you are using an image from a WIM file for features on demand, you can use the DISM tool to service the image. For more information, see [Mount and Modify a Windows Image Using DISM](#). If you are using an online Windows installation shared on your local network as a repair image, you should make sure that the computer has access to Windows Update.

### **Multilingual images**

You must include all of the relevant language packs with your repair source files for the locales that your image supports. If you try to restore a feature without all of the localization components that the Windows installation requires for that feature, the installation will fail.

You can install additional language packs after a feature is restored.

## Related topics

[What is DISM?](#)

[Enable or Disable Windows Features Using DISM](#)

[Repair a Windows Image](#)

[DISM Operating System Package Servicing Command-Line Options](#)

# Capture, apply, and work with Windows images

1/24/2019 • 2 minutes to read

The Windows 10 OPK includes an image file, **install.wim**, which contains the operating system files for your Windows edition.

Starting by deploying this image to a device, add your customizations to it, test, and incorporate those changes into your images, iterating until you're ready to put the image onto the factory floor.

## File formats: WIM, FFU, VHD

To get the Windows image onto a device, you can install Windows [using Windows Setup \(.wim only\)](#) or a [deployment script \(.wim or .ffu\)](#).

Manage multiple variations of your Windows images by [combining them into a single .wim file](#). A single .wim file can take a fraction of the drive space that multiple image files can take.

When you're ready to manufacture devices, [recapture and deploy the image as an .FFU file](#), because FFUs are faster to apply on a factory floor. FFUs can still be modified like .WIM files, but you can only store one variation of the image at a time.

For more details about the file formats, see [WIM vs. VHD vs. FFU: comparing image file formats](#).

## Drives and partitions

Learn about setting up [hard drives and partitions](#).

Notes:

- For WIM-based images, you can usually set up partitions during deployment, just before applying the .WIM image file.
- For VHD and FFU-based images, you may need to do some partition setup before you capture the VHD/FFU image file. If the reference and destination drives are different sizes, you may also need to perform setup after you apply the VHD/FFU image file.

## In this section

TOPIC	DESCRIPTION
<a href="#">Hard Drives and Partitions</a>	Learn how to prepare a hard drive for a Windows installation
<a href="#">WIM vs. VHD vs. FFU: comparing image file formats</a>	Learn the differences of available Windows image formats
<a href="#">Capture and apply WIM images</a>	Capture, deploy, and modify a Windows image using WIM
<a href="#">Capture and apply Full Flash Update (FFU) images</a>	Capture, deploy, and modify a Windows image using FFU
<a href="#">Modify an image</a>	How to make changes to Windows images

# WIM vs. VHD vs. FFU: comparing image file formats

1/18/2019 • 2 minutes to read

Comparing .WIM, .VHD/.VHDX, and .FFU: These file formats are all used to deploy Windows to new devices. Here's how they compare:

	Windows image (.WIM)	Virtual Hard Disk (.VHD/VHDX)	Full Flash Update (.FFU)
Common uses	<p>Fastest for testing and modifying Windows images.</p> <p>Can store multiple image variations in the same .WIM file, often with little additional storage required.</p>	<p>Easiest for deploying Windows to virtual PCs.</p> <p>You can boot a new device directly from a single VHD/VHDX file.</p>	Fastest for capturing and deploying Windows on a factory floor.
Imaging style	File-based	Sector-based	Sector-based
Compression	Supports multiple types of compression	None	Xpress-Huffman is used by default when an FFU is captured with DISM
What does it capture?	A set of files, up to an entire partition.	Captures the full set of drive information, including partitions.	Captures the full set of drive information, including partitions.
When I apply the image, what happens?	<p>Adds the files and folders to the partition.</p> <p>If there are existing files and folders with the same names, they're replaced. Otherwise, the existing files are kept.</p>	Cleans the entire drive.	Cleans the entire drive.
Can I deploy to different sizes of hard drives?	Yes.	Yes, though the new drive must be the same size or larger than the original.	Yes, though the new drive must be the same size or larger than the original.
Can I modify the images?	Yes. With tools like DISM, you can mount, modify, and unmount the image.	Yes, you can mount a VHD/VHDX as if it were removable media, and modify the files.	Yes. With tools like DISM, you can mount, modify, and unmount the image.

Reliability			Includes a catalog and hash table to validate a signature upfront before flashing onto a device. The hash table is generated during capture, and validated when applying the image.
-------------	--	--	---

To learn more, see [/Apply-Image](#) in DISM Image Management Command-Line Options.

## Related topics

[DISM Deployment Image Servicing and Management \(DISM\) Command-Line Options](#)

[VHD/VHDX Boot to VHD \(Native Boot\): Add a Virtual Hard Disk to the Boot Menu](#)

[Deploy Windows on a VHD \(Native Boot\)](#)

[FFU Deploy Windows using Full Flash Update \(FFU\)](#)

[DISM Image Management Command-Line Options](#)

[FFU image format](#)

# Capture and apply Windows Full Flash Update (FFU) images

1/24/2019 • 6 minutes to read

Deploy Windows faster on the factory floor by using the Full Flash Update (FFU) image format. FFU images allow you to apply an image of a physical drive, including Windows, recovery, and system partition information all at once directly to a different drive.

Unlike the file-based WIM format, FFU is a sector-based file container that stores one or more partitions. Sector-based imaging means that FFUs take less time to deploy, but have larger files sizes than WIMs. See [WIM vs. VHD vs. FFU: comparing image file formats](#) for information about the differences between image formats.

Starting with Windows 10, version 1709, DISM has the ability to capture, deploy, and service FFUs, with the following limitations:

- The drive that an FFU is applied to has to be the same or larger than the drive it is captured from
- FFU captures of encrypted disks are not supported
- Captures of disks that have [Volume Shadow Copy Service \(VSS\)](#) enabled are not supported
- Splitting compressed FFUs is not supported

## What you need to work with FFUs in Windows

To capture, deploy, and mount FFU images with DISM, you'll need to work in a Windows 10, version 1709 or later, or WinPE for Windows 10, version 1709 or later environment.

To capture and deploy FFUs using the instructions below, you'll also need:

- A Windows PC that has been [generalized with Sysprep](#). We'll refer to this as the reference PC. For a walkthrough on how to create an image that's ready for deployment, see the [OEM deployment of Windows 10 lab](#).
- A PC to deploy the FFU image to. We'll refer to this as the destination PC. The hard drive on this PC will be overwritten, so make sure you're using a PC that doesn't have any information you want to keep.
- The latest version of the ADK, from [Download the Windows ADK](#)
- Bootable WinPE media for Windows 10, version 1803 or later. See [WinPE: Create USB bootable drive](#) for instructions on how to create WinPE Media.
- Storage
  - USB storage, formatted as NTFS with enough space to save the FFU. 16 GB is enough space to store an FFU of a basic Windows image. You can use the same USB drive for WinPE and storage if you follow the [instructions for creating a multipartiton USB drive](#). For best performance, you want to maximize I/O between where your FFU is stored and the destination PC. For best performance use a USB 3.0 drive to store the image, and an internal SSD for the destination device.

**or**

- Network storage where you can keep your FFU image. For optimal performance, use a 1 Gb or faster network.

## Capture an FFU

1. Boot the reference PC using WinPE bootable media.

- Identify the drive to which you'll be capturing the image from. You can use diskpart, or [add Windows PowerShell support to WinPE](#) and use [Get-Disk](#) for scriptability and more complex setups such as a server with multiple disks.

```
diskpart
list disk
exit
```

The output will list your drives. Make a note of the disk number in the `Disk ###` column. This is the value that you'll use when capturing your image.

```
DISKPART> list disk

Disk ###  Status     Size      Free     Dyn  Gpt
-----  -----
Disk 0    Online    238 GB    0 B      *    
Disk 1    Online    28 GB     0 B

DISKPART>
```

- Use DISM to capture an image of all the partitions on the physical drive. For *disk X*; the string used with `/capturedrive` will look like this: `\.\PhysicalDriveX`, where *X* is the disk number that diskpart provides. For example, to capture Disk 0, you'd use `/CaptureDrive:\.\PhysicalDrive0`.

For more information about PhysicalDrive *X*, see [CreateFile function](#).

To see command line options for capturing FFUs, run `dism /capture-ffu /?` or see [DISM Image Management Command-Line Options](#). Note that you shouldn't have to specify a PlatformID when capturing a desktop image.

The following command captures an FFU image of PhysicalDrive0 called WinOEM.ffu. The `/name` and `/description` arguments allow you to set information about your image. This information is displayed when you use `dism /get-imageinfo`. `/name` is required, `/description` is optional.

```
DISM.exe /capture-ffu /imagefile=e:\WinOEM.ffu /capturedrive=\.\PhysicalDrive0 /name:disk0
/description:"Windows 10 FFU"
```

This command also gives a name and description to the FFU image. Name is a required parameter.

## Deploy Windows from WinPE using an FFU

- Boot your destination PC to WinPE.
- Connect a storage drive or map the network location that has your FFU file and note the drive letter, for example, N.
- Identify the drive to which you'll be applying the image:

```
diskpart
list disk
exit
```

Note the drive number in the `Disk ###` column.

- Apply the image to the cleaned drive. Here, we're applying n:\WinOEM.ffu to Disk 0.

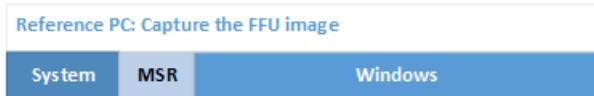
```
DISM /apply-ffu /ImageFile=N:\WinOEM.ffa /ApplyDrive:\\.\PhysicalDrive0
```

To see the commands available with /apply-ffu, run `dism /apply-ffu /?` or see [DISM Image Management Command-Line Options](#).

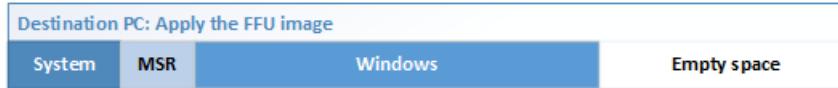
## Capturing and applying FFUs on differently-sized drives

The FFU format captures the partition layout. If the reference PC and destination PC hard drives may be different sizes, use this strategy to make sure your Windows partition is as large as it can be:

1. On the reference PC, install Windows with no recovery partition, and set up the Windows partition as the final partition on the drive. To set this up, use Windows Setup or a [deployment script](#).



2. [Capture the FFU](#).
3. On the destination PC, [deploy the FFU](#). For larger drives, there will be unused space at the end.



4. Using Diskpart, expand the Windows partition to fill the empty space.



Then, shrink the Windows partition to make room for the recovery partition.



Configure your recovery partition.

Example, using [Sample scripts: create recovery partition](#).

```
ApplyRecovery.bat
```

## Mount an FFU for servicing

You can use DISM to mount FFU images for servicing. Like with other image formats, you can mount and modify an FFU before committing changes and unmounting. Mounting an FFU for servicing uses the same `/mount-image` command that you use for mounting other image types. When mounting an FFU, you'll always use `/index:1` when mounting.

Unlike WIM images, FFU images get mounted as virtual hard disks. Files appear in the specified mount folder, but since FFUs can contain more than one partition but only have one index, DISM maps only the Windows partition from the mounted FFU to the mount folder.

To mount an FFU

1. Open a Command Prompt as administrator.

2. Mount the image using `dism /mount-image`. This example mounts D:\WinOEM.ffa to C:\ffumount:

```
dism /mount-image /imagefile:"D:\WinOEM.ffa" /mountdir:"C:\ffumount" /index:1
```

To see available command line options for `/mount-image` run `dism /mount-image /?` or see [DISM image management command line options](#).

3. Service your image. For example, to enable the legacy components feature:

```
dism /image:"C:\ffumount" /enable-feature:legacycomponents
```

To see available options, run `dism /image:<path to mounted image> /?` or

4. Unmount your FFU image and commit or discard changes. If you use `/commit`, your changes will be saved to your FFU file.

To unmount your FFU and commit changes, you'd use `/unmount-image` with the `/commit` option:

```
dism /unmount-image /mountdir:"C:\ffumount" /commit
```

If you decide to not keep the changes you've made to the FFU, you can use `/unmount-image` with the `/discard` option:

```
dism /unmount-image /mountdir:"C:\ffumount" /discard
```

## Related topics

[Download and install the Windows ADK](#)

[FFU image format](#)

[WIM vs. VHD vs. FFU: comparing image file formats](#)

[Planning a Multicast Strategy in Configuration Manager](#)

[Capture and Apply Windows, System, and Recovery Partitions](#)

[DISM Image Management Command-Line Options](#)

[CreateFile function](#)

# Capture and apply a Windows .WIM image

1/24/2019 • 2 minutes to read

This section covers how to capture and apply a Windows WIM image.

## In this section

TOPIC	DESCRIPTION
<a href="#">Capture and apply Windows using a single WIM file</a>	If you're using a standard partition configuration, use this method to capture and apply WIMs.
<a href="#">Capture and apply Windows, System, and Recovery Partitions</a>	If you've got a custom partition configuration, use this method to capture and apply WIMs.
<a href="#">Create a WIM for multiple architecture types</a>	How to create an image that supports more than one architecture
<a href="#">Split a Windows image file (.wim) to span across multiple DVDs</a>	Split a large .wim file into several smaller files that will fit on your selected media.
<a href="#">Append a volume image to an existing .WIM</a>	Save drive space on your deployment server by storing multiple variations of Windows images in a single WIM file.
<a href="#">Create a data WIM Image</a>	How to create an image that contains data instead of a Windows image

# Capture and apply a Windows image using a single .WIM file

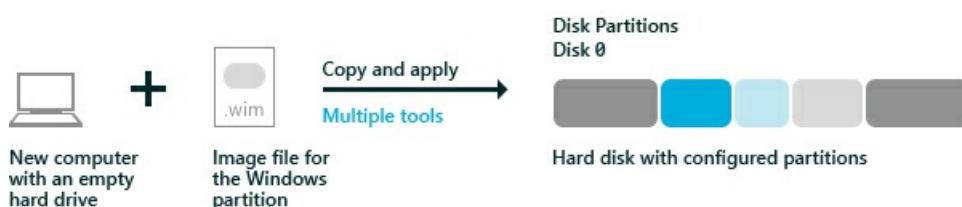
1/25/2019 • 2 minutes to read

Capture a Windows image (.WIM) file and use it to deploy Windows to new devices.

You can start with either the install.wim file from a Windows distribution ISO, or you can generalize and capture a running Windows image into a .WIM file.

WIM files only capture a single partition. You can usually capture just the Windows partition, and then use files from that image to set up the rest of the partitions on the drive. If you've created a custom partition configuration, see [Capture and Apply Windows, System, and Recovery Partitions](#).

**Capture only the Windows image, and configure other partitions during deployment**



## Capture the image\*\*

1. If you've booted into Windows, generalize the image so that it can be deployed to other devices. For more information, see [Sysprep \(Generalize\) a Windows installation](#).
2. Boot the device using [Windows PE](#).
3. Capture the Windows partition. For example:

```
Dism /Capture-Image /ImageFile:"D:\Images\Fabrikam.wim" /CaptureDir:C:\ /Name:Fabrikam
```

Where D: is a USB flash drive or other file storage location.

## Apply the image

1. Boot the device using [Windows PE](#).
2. Wipe the hard drive and set up new hard disk partitions using a [script](#). Use [CreatePartitions-UEFI.txt](#) (or [CreatePartitions-BIOS.txt](#) for older, legacy BIOS devices).

```
diskpart /s CreatePartitions-UEFI.txt
```

3. Apply the images using a [script](#).

```
D:\ApplyImage.bat D:\Images\Fabrikam.wim
```

Sample script:

```
rem == ApplyImage.bat ==

rem == These commands deploy a specified Windows
rem     image file to the Windows partition, and configure
rem     the system partition.

rem   Usage:  ApplyImage WimFileName
rem   Example: ApplyImage E:\Images\ThinImage.wim ==

rem == Set high-performance power scheme to speed deployment ==
call powercfg /s 8c5e7fda-e8bf-4a96-9a85-a6e23a8c635c

rem == Apply the image to the Windows partition ==
dism /Apply-Image /ImageFile:%1 /Index:1 /ApplyDir:W:\

rem == Copy boot files to the System partition ==
W:\Windows\System32\bcdboot W:\Windows /s S:

:rem == Copy the Windows RE image to the
:rem     Windows RE Tools partition ==
md R:\Recovery\WindowsRE
xcopy /h W:\Windows\System32\Recovery\Winre.wim R:\Recovery\WindowsRE\

:rem == Register the location of the recovery tools ==
W:\Windows\System32\Reagentc /Setreimage /Path R:\Recovery\WindowsRE /Target W:\Windows

:rem == Verify the configuration status of the images. ==
W:\Windows\System32\Reagentc /Info /Target W:\Windows
```

## Related topics

[Deploy Windows using Full Flash Update \(FFU\)](#)

[Capture and Apply Windows, System, and Recovery Partitions](#)

[Configure UEFI/GPT-Based Hard Drive Partitions](#)

[Configure BIOS/MBR-Based Hard Drive Partitions](#)

[BCDboot Command-Line Options](#)

[REAgentC Command-Line Options](#)

# Capture and apply Windows, system, and recovery partitions

1/24/2019 • 5 minutes to read

When you're deploying Windows using the WIM format, and using customized partitions, a customized system partition or an additional utility partition, learn how to capture the right partitions to apply to new PCs.

Notes:

- If you don't need custom partitions, you usually can just capture the Windows partition, and then use files from that image to set up the rest of the partitions on the drive. See [Capture and Apply Windows using a single WIM file](#).
- If you're capturing using the FFU format, you'll be fine - FFU captures all of the partitions on the drive.

## Capture the customized partitions

### Step 1: Determine which partitions to capture

This table shows the types of partitions that you must capture and those that are managed automatically.

If you're deploying both UEFI and BIOS systems, you can reuse your primary and logical partitions across UEFI-based and BIOS-based devices, but not the other partition types.

PARTITION TYPE	SHOULD YOU CAPTURE THIS PARTITION?	CAN YOU REUSE THE SAME WIM ON UEFI AND BIOS FIRMWARE?
<b>System partition</b> (EFI System Partition or BIOS system partition)	Optional. If only a simple set of partition files is required, you don't have to capture this partition.	No
<b>Microsoft Reserved partition (MSR)</b>	No	No
<b>Primary partitions</b> (Windows partitions, data / utility partitions that you've added)	Yes	Yes
<b>Recovery partition</b>	Optional. If you haven't customized this partition, you don't have to capture it.	No
<b>Extended partition</b>	No	No
<b>Logical partitions</b> (Windows partitions, data / utility partitions that you've added)	Yes	Yes

### Step 2: Prepare to capture partitions

1. If you've booted the Windows image, generalize it so that it can be deployed to other devices. For more information, see [Sysprep \(Generalize\) a Windows installation](#).
2. Start your reference device by using Windows PE.

3. At the Windows PE command prompt, type `diskpart` to open the DiskPart tool.

```
X:> diskpart  
DISKPART>
```

4. Check to see if the partitions that you want to capture have drive letters assigned.

```
DISKPART> list volume
```

Volume #	Ltr	Label	Fs	Type	Size	Status	Info
Volume 0	C	Windows	NTFS	Partition	475 GB	Healthy	Boot
Volume 1			NTFS	Partition	554 MB	Healthy	
Volume 2		SYSTEM	FAT32	Partition	499 MB	Healthy	System

If any of the partitions you want to capture don't already have a drive letter assigned, continue:

5. List the disks in your PC:

```
DISKPART> list disk
```

Disk #	Status	Size	Free	Dyn	Gpt
Disk 0	Online	127 GB	0 B		*

6. Select the primary hard disk:

```
DISKPART> select disk 0
```

7. View the partitions:

```
DISKPART> list partition
```

Partition #	Type	Size	Offset
Partition 1	System	499 MB	1024 KB
Partition 2	Reserved	128 MB	500 MB
Partition 3	Primary	475 GB	628 MB
Partition 4	Recovery	554 MB	476 GB

8. Select a partition that needs a drive letter:

```
DISKPART> select partition=1
```

9. Assign a letter to the partition with the `assign letter` command. For example,

```
DISKPART> assign letter=S
```

10. Type `exit` to return to the Windows PE command prompt.

```
DISKPART> exit  
X:>
```

For more information, see the DiskPart Help from the command line, or [Diskpart Command line syntax](#).

### Step 3: Capture images for each customized partition.

- At the Windows PE command prompt, capture each customized partition, for example:

```
Dism /Capture-Image /ImageFile:C:\my-windows-partition.wim /CaptureDir:C:\ /Name:"My Windows  
partition"  
Dism /Capture-Image /ImageFile:C:\my-system-partition.wim /CaptureDir:S:\ /Name:"My system partition"
```

For more information about using the DISM tool to capture an image, see [DISM Image Management Command-Line Options](#).

### Step 4: Save images to the network or another safe location.

- Connect an external drive, or connect to a safe network location, for example:

```
net use n: \\Server\Share
```

If prompted, provide your network credentials.

- Copy the partitions to your network share. For example,

```
md N:\Images\  
copy C:\my-windows-partition.wim N:\Images\  
copy C:\my-system-partition.wim N:\Images\
```

## Apply the images

### Step 1: Prepare to apply partitions

- Start your destination device by using Windows PE.
- Connect an external drive, or connect to a safe network location, for example:

```
net use n: \\Server\Share
```

If prompted, provide your network credentials.

- Wipe the hard drive and create new partitions.

To apply to multiple devices, save the Diskpart commands into a script and run them on each new device. For examples, see [Configure UEFI/GPT-Based Hard Drive Partitions](#) or [Configure BIOS/MBR-Based Hard Drive Partitions](#). Example:

```
diskpart /s D:\CreatePartitions-UEFI.txt
```

Where D: is a USB flash drive or other file storage location.

In these **DiskPart** examples, the partitions are assigned the letters: System=S, Windows=W, and Recovery=R.

Change the Windows drive letter to a letter that's near the end of the alphabet, such as W, to avoid drive letter conflicts. Do not use X, because this drive letter is reserved for Windows PE. After the device reboots, the Windows partition is assigned the letter C, and the other partitions don't receive drive letters.

If you reboot, Windows PE reassigns disk letters alphabetically, starting with the letter C, without regard to the configuration in Windows Setup. This configuration can change based on the presence of different drives, such as USB flash drives.

4. Optional: speed up the image capture by setting the power scheme to High performance:

```
powercfg /s 8c5e7fda-e8bf-4a96-9a85-a6e23a8c635c
```

## Step 2: Apply the partitions

1. **Windows and data partitions:** Apply the image(s), example:

```
dism /Apply-Image /ImageFile:N:\Images\my-windows-partition.wim /Index:1 /ApplyDir:W:\
```

where W: is the Windows partition.

2. **System partition:** You can either:

- Configure the system partition by using the BCDBoot tool. This tool copies and configures system partition files by using files from the Windows partition. For example:

```
W:\Windows\System32\bcdboot W:\Windows /s S:
```

or:

- Apply a custom image

```
dism /Apply-Image /ImageFile:N:\Images\my-system-partition.wim /Index:1 /ApplyDir:S:\
```

Where S: is the system partition

3. **Recovery partition:**

- You can either:

- Copy the Windows Recovery Environment (RE) tools into the recovery tools partition.

```
md R:\Recovery\WindowsRE  
copy W:\Windows\System32\Recovery\winre.wim R:\Recovery\WindowsRE\winre.wim
```

Where R: is the recovery partition

or:

- Apply a custom image

```
dism /Apply-Image /ImageFile:N:\Images\my-recovery-partition.wim /Index:1 /ApplyDir:R:\
```

- Register the location of the recovery tools, and hide the recovery partition using Diskpart. You can use our [sample script](#) or perform the steps manually:

```
W:\Windows\System32\reagentc /setreimage /path R:\Recovery\WindowsRE /target W:\Windows
```

Diskpart steps for UEFI:

```
set id="de94bba4-06d1-4d40-a16a-bfd50179d6ac"
gpt attributes=0x8000000000000001
```

Diskpart steps for BIOS:

```
set id=27
```

### Step 3: Verify that it worked

Reboot the device (`exit`). Windows should boot.

#### NOTE

If the device doesn't boot, (for example, if you receive the error message: **Bootmgr not found. Press CTRL+ALT+DEL**) check the steps for setting up the system partition:

- See [BCDBoot command-line options](#) for more info about copying boot files to the system partition.
- Use the DiskPart tool to check to make sure that the system partition is set to Active.

Complete the out of box experience (OOBE) as a new user, and check the recovery partition:

- Check that in File Explorer that the Recovery partition is not visible.
- View the partitions exist, either by right-clicking **Start** and selecting **Disk Management**, or by using diskpart (Open a command prompt as an administrator > `diskpart` > `select disk 0` > `list partition` > `exit`).

## Related topics

[Configure UEFI/GPT-Based Hard Drive Partitions](#)

[Configure BIOS/MBR-Based Hard Drive Partitions](#)

[BCDboot Command-Line Options](#)

[REAgentC Command-Line Options](#)

# Create a WIM for Multiple Architecture Types Using DISM

1/18/2019 • 2 minutes to read

When you plan your deployment scenarios, consider how you will deploy and maintain your images for different architecture types. There are several ways you can manage multiple Windows images for multiple architecture types. Because you can deploy both 32-bit and 64-bit Windows images from a 32-bit preinstallation environment, you can maintain 32-bit and 64-bit Windows images in the same Windows image (.wim) file or separate .wim files.

Because you can store multiple Windows images in a single .wim file, you can create architecture-specific .wim files or a single .wim file that contains images for multiple architecture types.

- 32-bit images only

You can create a .wim file that contains Windows images for a single architecture type. In this scenario, you build a .wim file that contains one or more Windows images for 32-bit systems only. You create separate .wim files for different architecture types.

- 64-bit images only

You can create a .wim file that contains one or more of the 64-bit Windows images that you deploy.

- 32-bit and 64-bit images

You can create a .wim file that contains multiple Windows editions for multiple architecture types. For example, you can create a Windows image that contains two versions of Windows, one for 32-bit architectures, and one for 64-bit architectures.

## To Create a Windows Image for Multiple Architecture Types

You can create a single .wim file that includes both 32-bit and 64-bit Windows images. You must have both a 32-bit Windows distribution and a 64-bit Install.wim file. (A Windows distribution is the collection of files on the Windows installation media that includes not only the Install.wim file, but the additional files and directories that are required for Setup.) Cross-platform deployment is supported only from 32-bit Windows Setup.

1. Copy the entire 32-bit Windows distribution to a temporary directory on the local computer.
2. Copy the 64-bit Install.wim file to a separate temporary directory on the local computer.
3. At a command prompt, use the **Dism** command to export the 64-bit Windows images to the Install.wim file in the Windows distribution.
4. Repeat the **Dism /Export-Image** command for each 64-bit Windows image that you want to add to the Windows distribution.

For example, if you copy the distribution to C:\WindowsDistribution and the 64-bit Install.wim file to C:\Windows64-bit, you would use the following at a command prompt.

```
Dism /Export-Image /SourceImageFile:c:\windows64-bit\install.wim /SourceIndex:1  
/DestinationImageFile:c:\windowsdistribution\sources\install.wim /DestinationName:"Fabrikam 64-bit Image"
```

**Note** It is important to add the name of the Windows image to indicate that it is for 64-bit computers only.

The 64-bit Windows image and all accompanying metadata are copied to the Install.wim file to a new index during the export process. When you have added all Windows images to the Install.wim file, your Windows distribution is ready to be used in your environment.

During attended installations, users will be prompted to select which architecture-specific Windows image to install (x86 or x64 images).

In unattended installations, if you store multiple Windows editions for multiple architecture types in a single .wim file, you must explicitly specify which image to install during Windows Setup with the `MetaData` setting.

## Related topics

[DISM Image Management Command-Line Options](#)

[Windows Setup Supported Platforms and Cross-Platform Deployments](#)

# Split a Windows image file (.wim) to span across multiple DVDs

1/18/2019 • 2 minutes to read

Split a Windows image (.wim) file into a set of smaller (.swm) files.

Use this procedure when you're installing Windows from media that can't handle the Windows image file size, for example:

- **DVDs** (A standard single-sided DVD stores 4.7GB).
- **USB keys formatted as FAT32.** FAT32 is required to boot many modern (UEFI-based) PCs, but has a maximum file size of 4GB. (Workaround: [Create a USB key with multiple partitions](#).)

## Limitations:

- You can't modify a set of split image (.swm) files.
- Applying split image (.swm) files is only supported when all of the .swm files are in the same folder. This means for DVD deployment, you'll need to copy the files over to the destination PC before you can use Windows Setup or DISM /Apply-Image, as shown in this topic.

## Split the file

1. Mount your Windows distribution ISO.

2. Split the Windows image:

```
Dism /Split-Image /ImageFile:C:\sources\install.wim /SWMFile:C:\sources\install.swm /FileSize:4700
```

where:

- `C:\sources\install.wim` is the name and the location of the image file that you want to split.
- `C:\sources\install.swm` is the destination name and the location for the split .swm files. The first split .swm file is named `install.swm`. The file names for the next files include numbers, for example, `install2.swm`, `install3.swm`, and so on.
- `4700` is the maximum size in MB for each of the split .swm files to be created.

## USB deployment

Store all of the .swm files in the same folder on the USB key.

For Windows Setup instructions, see the Troubleshooting section from [Install Windows from a USB flash drive](#).

## DVD deployment

1. Copy the files to individual DVDs. For example, insert the first DVD and type:

```
copy C:\images\install.swm D:\*
```

Then insert the second DVD and type:

```
copy C:\images\install2.swm D:\*
```

And so on until all .swm files are copied to DVDs.

## 2. Boot your destination PC to Windows PE from DVD.

From Windows PE, you can either deploy using Windows Setup or a script.

### 3. Configure and format your hard drive partitions, as shown in [Capture and Apply Windows, System, and Recovery Partitions](#).

### 4. Copy the files to a single temporary folder. For example, insert the first DVD and type:

```
md C:\TempInstallFolder  
copy d:\install.swm c:\TempInstallFolder\*
```

Then insert the second DVD and type:

```
copy d:\install2.swm c:\TempInstallFolder\*
```

And so on until all .swm files are copied.

### 5. Install using Windows Setup or a script.

- **Windows Setup:** Insert a DVD with Windows Setup, and use it to install the split Windows image files.

```
D:\Setup.exe /InstallFrom:"C:\TempInstallFolder\install.swm"
```

- **Use a script**

- Apply your image using the DISM /Apply-Image /SWMFile option:

```
Dism /Apply-Image /ImageFile:C:\TempInstallFolder\install.swm  
/SWMFile:c:\TempInstallFolder\install*.swm /Index:1 /ApplyDir:D:\
```

- Set up your system and recovery partitions, as shown in [Deploy Windows using a Script](#).

### 6. Clean up: remove the temporary folder

```
rd c:\TempInstallFolder /s /q
```

## Related topics

[Capture and Apply Windows, System, and Recovery Partitions](#)

[WinPE: Use a single USB key for WinPE and a WIM file \(.wim\)](#)

[Install Windows from a USB flash drive](#)

[DISM Image Management Command-Line Options](#)

# Append, apply, and export volume images with a Windows Image (.wim) file

1/18/2019 • 2 minutes to read

Manage multiple Windows images by combining them into a single .wim file. A single .wim file can take a fraction of the drive space that multiple .wim files can take.

When you combine two or more Windows image files into a single .wim, any files that are duplicated between the images are only stored once.

## Apply a volume image



Run these commands using DISM from a command prompt with administrator privileges.

## Multiple Windows Images in a .wim file

### Combine images: append a volume image to an existing image

Example: append an image of the D drive to an existing image called install.wim. Each new image receives a new index number, starting from 1.

```
Dism /Append-Image /ImageFile:"C:\images\install.wim" /CaptureDir:D:\ /Name:"Home + drivers"
```

### See a list of the volume images contained in a .WIM file

```
Dism /Get-ImageInfo /ImageFile:"C:\images\install.wim"
```

### Apply a volume image from the .WIM file

You can refer to an image either by image name or image index number. Examples:

```
Dism /Apply-Image /ImageFile:"C:\images\install.wim" /Index:2 /ApplyDir:D:\
```

```
Dism /Apply-Image /ImageFile:"C:\images\install.wim" /Name:"Home + drivers" /ApplyDir:D:\
```

### Extract an image from the .WIM file

Create a new .WIM file that includes only the files you need from a single volume image, for example, when [creating recovery media](#). The destination .WIM file starts with a new index number: 1.

Examples:

```
Dism /Export-Image /SourceImageFile:"C:\images\install.wim" /SourceIndex:2  
/DestinationImageFile:"C:\resetmedia_amd64\media\sources\install.wim"  
  
Dism /Export-Image /SourceImageFile:"C:\images\install.wim" /SourceName:"Home + drivers"  
/DestinationImageFile:"C:\resetmedia_amd64\media\sources\install.wim"
```

For more information, see [DISM Image Management Command-Line Options](#).

## Related topics

[Capture Images of Hard Disk Partitions Using DISM](#)

[DISM Image Management Command-Line Options](#)

# Create a Data Image Using DISM

1/18/2019 • 2 minutes to read

To add applications, files, and other resources to Windows during an installation, you can create a data image. By using the Deployment Image Servicing and Management (DISM) tool, you can create additional Windows image (.wim) files that contain only files and applications that you intend to copy to the Windows installation.

Data images enable you to add:

- Applications, files, scripts, and other resources to Windows during an installation.
- Files, resources, and other data to a partition other than the operating system partition.

**Note** Data images must be used only to add new files to a Windows installation. Do not use data images to replace existing Windows files. Overwriting operating system data is unsupported.

Previous methods of transferring data to a Windows installation required the use of `$OEM$` folders. These folder structures are still supported, but data images provide an easier and more efficient means of transferring additional data to Windows.

In unattended installations, the Windows image to install is specified by the `osImage` setting in the Microsoft-Windows-Setup component. You can add one or more `DataImage` settings in the Microsoft-Windows-Setup component that represent additional data images that you add to the system. For more information, see the [Windows Unattended Setup Reference](#).

## To create a data image

1. Locate the data that you will create a data image for.
2. Open a command prompt as an administrator, or boot the computer to Windows PE to open the Windows PE command prompt.
3. Use DISM to compress your data files to a .wim file. For example:

```
Dism /Capture-Image /ImageFile:c:\data\myData.wim /CaptureDir:C:\data\dataFiles /Name:MyData
```

In this example, everything under the C:\Data\DataFiles directory is added to the .wim file and the .wim file is given the label "MyData". All files and folders under C:\Data\DataFiles are extracted to the root of the drive specified in the answer file.

For more information about how to use DISM, see [DISM Image Management Command-Line Options](#).

4. Copy the data image to an available location such as another partition or a network share during Windows Setup.

## To add a data image path to an answer file

1. Use Windows System Image Manager (Windows SIM) to create an answer file that contains the path to the data image to install and the location for the installation.
2. Add the Microsoft-Windows-Setup\ `DataImage` settings to the appropriate configuration pass for your environment. For example: `windowsPE`.
3. Save the answer file and close Windows SIM.

The answer file must resemble the following example:

```
<settings pass="windowsPE">
  <component name="Microsoft-Windows-Setup" processorArchitecture="x86"
  publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS"
  xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ImageInstall>
      <DataImage wcm:action="add">
        <InstallTo>
          <DiskID>0</DiskID>
          <PartitionID>1</PartitionID>
        </InstallTo>
        <InstallFrom>
          <Credentials>
            <Domain>Fabrikam</Domain>
            <Username>MyUsername</Username>
            <Password>MyPassword</Password>
          </Credentials>
          <Path>\networkshare\share\MyData.wim</Path>
        </InstallFrom>
        <Order>1</Order>
      </DataImage>
    </ImageInstall>
  </component>
</settings>
```

#### NOTE

If you're specifying a local folder in `path`, see [Path](#) in the [Unattended Windows Setup Reference](#) to learn about using relative or absolute paths.

4. Run Setup.exe, specifying the location of the answer file. For example:

```
setup /unattend:C:\unattend.xml
```

All the files and folders specified in the data image are extracted to the root of the drive during installation. Executable files and scripts are not run when the data image is applied; they are only copied to the drive. You can use `FirstLogonCommands` to specify commands to run the first time a user logs on to the computer. For more information about `FirstLogonCommands`, see the [Windows Unattended Setup Reference](#).

# Boot to a virtual hard disk: Add a VHDX or VHD to the boot menu

1/24/2019 • 4 minutes to read

Native Boot allows you to create a virtual hard disk (VHDX), install Windows to it, and then boot it up, either on your PC side-by-side with your existing installation, or on a new device.

A native-boot VHDX can be used as the running operating system on designated hardware without any other parent operating system. This differs from a scenario where a VHDX is connected to a virtual machine on a computer that has a parent operating system.

Native boot for Windows 10 requires the **.vhdx** format, not the .vhd format.

VHDXs can be applied to PCs or devices that have no other installations of Windows, without a virtual machine or hypervisor. (A hypervisor is a layer of software under the operating system that runs virtual computers.) This enables greater flexibility in workload distribution because a single set of tools can be used to manage images for virtual machines and designated hardware.

You can also deploy the VHDX to a PC that already has Windows installed on it, and use a boot menu to select between the existing version of Windows, or the version on the VHD.

To learn more about using VHDXs in an enterprise environment, see [Understanding Virtual Hard Disks with Native Boot](#).

## Prerequisites

- A technician PC with the Windows Assessment and Deployment Kit (Windows ADK) tools installed on it.
- A generalized Windows image (.WIM file). To learn more, see [Sysprep \(Generalize\) a Windows installation](#).
- A bootable Windows PE drive. To learn more, see [WinPE: Create USB Bootable drive](#).
- A destination PC or device on which to install the VHDX. This device requires 30 gigabytes (GB) or more of free disk space. You can install the VHDX to a device already running other operating system installations, or as the only operating system on a device.

## Step 1: Create a VHDX from diskpart

On the technician PC:

1. From the Command Prompt, open Diskpart.

```
diskpart
```

2. Create and prepare a new VHDX. In this example, we create a 25 GB fixed-type VHDX.

```
create vdisk file=C:\windows.vhdx maximum=25600 type=fixed
```

3. Attach the VHDX. This adds the VHDX as a disk to the storage controller on the host.

```
attach vdisk
```

4. Create a partition for the Windows files, format it, and assign it a drive letter. This drive letter will appear in File Explorer.

```
create partition primary  
format quick label=vhdx  
assign letter=v
```

5. Exit Diskpart

```
exit
```

## Step 2: Apply a Windows image to the VHD

On your technician PC, apply a generalized Windows image to the primary partition of the VHDX that you created and attached in [Step 1](#).

```
Dism /Apply-Image /ImageFile:install.wim /index:1 /ApplyDir:V:\
```

## Step 3: Detach the VHD, copy it to a new device, and attach it (optional)

You can deploy the VHDX to a device that already has a copy of Windows installed on it, or you can clean and prepare the destination PC's hard drive to use the VHD.

### **Detach the VHDX and save it to a network share or storage drive**

1. Use diskpart to detach the virtual disk from your technician PC.

```
diskpart  
select vdisk file=C:\windows.vhdx  
detach vdisk  
exit
```

2. Copy the VHDX to a network share or removable storage drive. The following maps a drive letter to a network share, creates a directory for the VHD, and then copies the VHD.

```
net use n: \\server\share\  
md N:\VHDs  
copy C:\windows.VHDX n:\VHDs\
```

### **Clean and prepare a new device for native boot**

On your destination PC:

1. Use your bootable WinPE key to [boot the destination PC to WinPE](#).
2. Clean and prepare the destination PC's hard drive. Create a system partition (S), and a main partition (M) where the VHDX will be stored.

BIOS:

```
diskpart
select disk 0
clean
rem == 1. System partition =====
create partition primary size=100
format quick fs=ntfs label="System"
assign letter="S"
active
rem == 2. Main partition =====
create partition primary
format quick fs=ntfs label="Main"
assign letter="M"
exit
```

UEFI:

```
diskpart
select disk 0
clean
convert gpt
rem == 1. System partition =====
create partition efi size=100
format quick fs=fat32 label="System"
assign letter="S"
rem == 2. Microsoft Reserved (MSR) partition =====
create partition msr size=128
rem == 3. Main partition =====
create partition primary
format quick fs=ntfs label="Main"
assign letter="M"
exit
```

3. Connect to the network drive or storage location where you copied the VHDX in [step 3.2](#).

```
net use N: \\server\share
```

4. Copy the VHDX from the network drive or storage location to the destination PC's main partition.

```
copy N:\VHDs\Windows.vhdx M:
```

## Attach the VHDX

1. While still booted into WinPE, attach your VHDX to the destination PC.

```
diskpart
select vdisk file=M:\windows.vhdx
attach vdisk
```

2. Identify the attached VHDX's volume letter. (Optional: Change it to another letter that makes more sense, for example V, and leave the diskpart command line open for the next step).

```
list volume
select volume 3
assign letter=v
```

## Step 4: Add a boot entry

1. From your destination PC, open Diskpart (if necessary) and identify the drive letters of the VHDX and the system partition, for example, V and S.

```
diskpart
list volume
exit
```

2. Add a boot entry to the device. You can add multiple VHDX files using this method.

BIOS:

```
V:
cd v:\windows\system32
bcdboot v:\windows /s S: /f BIOS
```

UEFI:

```
V:\ 
cd v:\windows\system32
bcdboot v:\windows /s S: /f UEFI
```

3. Remove the Windows PE USB key.

4. Restart the destination PC.

If there's only one boot entry, the device immediately boots to Windows. If there's more than one boot entry, you'll see a boot menu where you can choose between the available versions of Windows on the device.

## Related topics

[Understanding Virtual Hard Disks with Native Boot](#)

[BCDboot Command-Line Options](#)

# Hard drives and partitions

1/18/2019 • 7 minutes to read

Learn methods of deploying Windows to different drives, including hard drives, solid-state drives (SSDs), or virtual hard drives (VHDs), and with different partition layouts, including with data and utility partitions.

## What's new in Windows 10

- Use Compact OS and single-sourcing to save more space on the hard drive: [Compact OS, single-sourcing, and image optimization](#).
- Use the FFU image format to apply images faster to your devices: [Deploy Windows using Full Flash Update \(FFU\)](#)
- In Windows 10 for desktop editions (Home, Pro, Enterprise, and Education), we've changed the partition layout. While we still use a separate recovery tools image, Windows no longer needs a separate full-system recovery image to use push-button reset features. This can save several GB of drive space.

We now recommend that you place the Windows recovery tools partition immediately after the Windows partition. This allows Windows to modify and recreate the partition later if future updates require a larger recovery image.

If you use scripts to deploy Windows, check out the sample scripts we've created for different device firmware types (the newer UEFI-based BIOS, or the legacy BIOS). To learn more, see [UEFI/GPT-based hard drive partitions](#) and [BIOS/MBR-based hard drive partitions](#).

- It's no longer necessary to run the Windows System Assessment Tests (WinSAT) on SSD drives. Windows detects SSD drives and tunes itself accordingly.
- On [UEFI/GPT-based drives](#), we've reduced the recommended size of the MSR partition from 128MB to 16MB.

## Drive types

You can install Windows to a hard drive, such as a hard disk drive or a solid-state drive. For additional security, you can use hard drives that the factory has pre-encrypted. A single computer may contain multiple drives.

### Solid-state drives

A solid-state drive (SSD) is a hard drive that uses solid-state memory to store persistent data. An SSD must have a minimum of 16 gigabytes (GB) of space to install Windows. For more information about drive space and RAM considerations, see [Compact OS, single-sourcing, and image optimization](#).

**Note** It's no longer necessary to run the Windows System Assessment Tests (WinSAT) on SSD drives. Windows now detects SSD drives and will tune itself accordingly.

### Advanced format drives

You can use some Advanced Format Drives to provide additional drive space.

Advanced Format 512 emulation (512e) drives are supported on either BIOS-based or UEFI-based computers.

Advanced Format 4K Native (4Kn) drives are supported on UEFI-based computers only.

**Warning** For Advanced Format 4K Native drives (4-KB-per-sector) drives, the minimum partition size is 260 MB, due to a limitation of the FAT32 file format. The minimum partition size of FAT32 drives is calculated as sector

size (4KB) x 65527 = 256 MB. For more information, see [Configure UEFI/GPT-Based hard drive partitions](#).

## Factory-encrypted hard drives

To help protect your deployment environment, you can use a factory pre-encrypted hard drive to prevent unauthorized access before you install Windows or any other software. For more information, see [Factory Encrypted Drives](#).

## Multiple hard drives

If you install Windows on a device that has multiple hard drives, you can use the disk location path to make sure that your images are applied to the intended drives.

To do this, use the `diskpart SELECT DISK=<disk location path>` command to select each drive. For example:

```
SELECT DISK=PCIROOT(0)#PCI(0100)#ATA(C00T00L00)
```

**Note** The system drive might not appear as disk 0 in the DiskPart tool. The system might assign different numbers to drives when you reboot. Different computers that have the same drive configuration can have different disk numbers.

To learn more, see [Configure multiple hard drives](#) and [Hard disk location path format](#).

## Partitions

You can divide your hard drive into multiple partitions. You can create separate system, recovery, Windows, or data partitions.

To enhance the security of the Windows partition or a data partition, you can use BitLocker to encrypt the partition. For more information, see [BitLocker Drive Encryption](#).

The partition types must match the firmware of the computer. You can install Windows on hard drives that are based on any of the following types of firmware:

- **Basic Input/Output System (BIOS)**. Uses the Master Boot Record (MBR) partition structure.
- **Extensible Firmware Interface (EFI) (Class 1)**: Uses the GUID Partition Table (GPT) partition structure.
- **Unified Extensible Firmware Interface (UEFI) Class 2**: Uses the GPT partition structure. Also includes a compatibility support module (CSM) that enables you to use BIOS functions, including the MBR partition structure. This module can be enabled or disabled in the firmware.
- **Unified Extensible Firmware Interface (UEFI) Class 3**: Uses the GPT partition structure.

To determine your system type, consult your hardware manufacturer.

### System and utility partitions

A *system partition* is a partition that contains the hardware-specific files that are needed to load Windows.

By default, during Windows Setup, Windows stores these hardware-specific files in a separate partition. This enables the computer to use the following:

- **Security tools**. Some security tools, such as BitLocker, require a separate system partition.
- **Recovery tools**. Some recovery tools, such as Windows Recovery Environment (Windows RE), require a separate system partition.
- **Multiple operating systems**. If a computer has multiple operating systems, such as Windows 10 for desktop editions and Windows 7, the computer displays a list of operating systems. The user can then select which operating system to boot. When the system boot files are on a separate partition, it is easier to remove a Windows partition or replace the partition with a new copy of Windows.

We recommend adding system utility partitions before the Windows partition, because in the event that a full-system recovery is needed, this partition order helps to prevent the recovery tools from overwriting the system and utility partitions.

For information about how to configure system partitions while you apply images, see [Capture and Apply Windows, System, and Recovery Partitions](#).

### **Microsoft reserved partition (MSR)**

The MSR is used on UEFI/GPT systems, to support software components that formerly used hidden sectors.

For more information about configuring MSR partitions, see [Configure UEFI/GPT-Based Hard Drive Partitions](#).

For more information about MSR partitions, see [Windows and GPT FAQ](#)

### **Recovery partitions**

Add a separate partition for the Windows Recovery Environment (Windows RE) at the end of the hard drive, just after the Windows partition. With this partition order, if future updates require adding to or replacing the Windows RE tools partition, Windows will be able to manage the partition size automatically.

For BIOS/MBR-based systems, it's still possible to combine the Windows RE tools partition with the system partition. To save drive space, consider creating logical partitions to get around the four-partition limit. For more info, see [Configure more than four partitions on a BIOS/MBR-based hard disk](#).

For Windows 10 for desktop editions, it's no longer necessary to create and maintain a separate full-system recovery image. Windows can perform a push-button reset using built-in tools.

### **Data partitions**

You can include a separate data partition to enable easier maintenance for situations where either the primary operating system is likely to be replaced, or when multiple operating systems exist on the same device, such as Windows 10 and Windows 7. When a device has multiple hard drives, a data partition may be stored on another drive.

**Warning** For typical single-drive configurations, we do not recommend that you use a separate data partition. There are two main reasons:

- The partition may not automatically protect data that is stored outside the user profile folders. For example, a guest user might have access to files in an unprotected data partition.
- If you change the default location of the user profile folders to any volume other than the system volume, you cannot service your image. The computer may not apply updates, fixes, or service packs to the installation. For a list of known issues related to changing the default folder locations, see [Description of known issues with the FolderLocation settings](#).

## See also

CONTENT TYPE	REFERENCES
<b>Deployment</b>	<a href="#">Configure UEFI/GPT-Based Hard Drive Partitions</a>   <a href="#">Configure BIOS/MBR-Based Hard Drive Partitions</a>   <a href="#">Configure More than Four Partitions on a BIOS/MBR-Based Hard Disk</a>
<b>Multiple drives</b>	<a href="#">Configure Multiple Hard Drives</a>   <a href="#">Hard Disk Location Path Format</a>   <a href="#">Internal and External SATA Port Configuration</a>   <a href="#">Configuring Disk Mirroring</a>

CONTENT TYPE	REFERENCES
<b>Using smaller drives</b>	<a href="#">Compact OS, single-sourcing, and image optimization</a>
<b>Operations</b>	<a href="#">Capture and Apply Windows, System, and Recovery Partitions</a>   <a href="#">Deploy Windows using Full Flash Update (FFU)</a>   <a href="#">Deploy Windows on a VHD (Native Boot)</a>   <a href="#">Factory Encrypted Drives</a>   <a href="#">BitLocker Drive Encryption</a>
<b>Troubleshooting</b>	<a href="#">Repair the boot menu on a dual-boot PC</a>
<b>Tools and settings</b>	<a href="#">UEFI Firmware</a>   <a href="#">The Windows and GPT FAQ</a>   <a href="#">BCDboot Command-Line Options</a>   <a href="#">DiskPart Command line syntax</a>   <a href="#">WIM vs. VHD vs. FFU: comparing image file formats</a>

# UEFI/GPT-based hard drive partitions

1/18/2019 • 7 minutes to read

Create custom partition layouts for your hard disk drives (HDDs), solid-state drives (SSDs), and other drives when deploying Windows to Unified Extensible Firmware Interface (UEFI)-based devices.

**Note** If you use a custom partition layout on Windows 10 for desktop editions (Home, Pro, Enterprise, and Education), update the push-button recovery script so the recovery tools can recreate the custom partition layout when needed.

## Partition Requirements

When you deploy Windows to a UEFI-based device, you must format the hard drive that includes the Windows partition by using a GUID partition table (GPT) file system. Additional drives may use either the GPT or the master boot record (MBR) file format.

A GPT drive may have up to 128 partitions.

Each partition can have a maximum of 18 exabytes (~18.8 million terabytes) of space.

### System partition

The device must contain a system partition. On GPT drives, this is known as the EFI System Partition, or the ESP. This partition is usually stored on the primary hard drive. The device boots to this partition.

The minimum size of this partition is 100 MB, and must be formatted using the FAT32 file format.

This partition is managed by the operating system, and should not contain any other files, including Windows RE tools.

#### NOTE

For Advanced Format 4K Native drives (4-KB-per-sector) drives, the minimum size is 260 MB, due to a limitation of the FAT32 file format. The minimum partition size of FAT32 drives is calculated as sector size (4KB) x 65527 = 256 MB.

Advanced Format 512e drives are not affected by this limitation, because their emulated sector size is 512 bytes. 512 bytes x 65527 = 32 MB, which is less than the 100 MB minimum size for this partition.

### Microsoft reserved partition (MSR)

Beginning in Windows 10, the size of the MSR is 16 MB.

Add an MSR to each GPT drive to help with partition management. The MSR is a reserved partition that does not receive a partition ID. It cannot store user data.

### Other utility partitions

Any other utility partitions not managed by Windows must be located before the Windows, data, and recovery image partitions. This allows end users to perform actions such as resizing the Windows partition without affecting system utilities.

Protect end users from accidentally modifying utility partitions by identifying them using a GPT attribute. This prevents these partitions from appearing in File Explorer.

### To set partitions as utility partitions

- When you are deploying Windows by using the \*\*DiskPart\*\* tool, use the \*\*attributes volume set GPT\\_ATTRIBUTE\\_PLATFORM\\_REQUIRED\*\* command after you create the partition to identify the partition as a utility partition. For more information, see the MSDN topic: [PARTITION\\_INFORMATION\\_GPT structure] (<https://go.microsoft.com/fwlink/?linkid=240300>).

## To verify that system and utility partitions exist

1. Click \*\*Start\*\*, right-click \*\*This PC\*\*, and then click \*\*Manage\*\*. The \*\*Computer Management\*\* window opens.
2. Click \*\*Disk Management\*\*. The list of available drives and partitions appears.
3. In the list of drives and partitions, confirm that the system and utility partitions are present and are not assigned a drive letter.

## Windows partition

- The partition must have at least 20 gigabytes (GB) of drive space for 64-bit versions, or 16 GB for 32-bit versions.
- The Windows partition must be formatted using the NTFS file format.
- The Windows partition must have 16 GB of free space after the user has completed the Out Of Box Experience (OOBE) and Automatic Maintenance has completed.

## Recovery tools partition\*\*

This partition must be at least 300 MB.

The Windows Recovery Environment (Windows RE) tools require additional free space:

- If the partition is less than 500 MB, it must have at least 50 MB of free space.
- If the partition is 500 MB or larger, it must have at least 320 MB of free space.
- If the partition is larger than 1 GB, we recommend that it should have at least 1 GB free.

When calculating free space, note:

- The recovery image, winre.wim, is typically between 250-300MB, depending on what drivers, languages, and customizations you add.
- The file system itself can take up additional space. For example, NTFS may reserve 5-15MB or more on a 750MB partition.

This partition must use the Type ID: DE94BBA4-06D1-4D40-A16A-BFD50179D6AC.

The recovery tools should be in a separate partition than the Windows partition to support automatic failover and to support booting partitions encrypted with Windows BitLocker Drive Encryption.

We recommend that you place this partition immediately after the Windows partition. This allows Windows to modify and recreate the partition later if future updates require a larger recovery image.

## Data partitions

The recommended partition layout for Windows 10 does not include data partitions. However, if data partitions are required, they should be placed after the Windows RE partition. This allows future updates to Windows RE to grow the Windows RE partition by shrinking the Windows partition.

This layout makes it more difficult for end users to remove the data partition and merge the space with the Windows partition. To do so, the Windows RE partition must be moved to the end of the unused space reclaimed from the data partition, so that the Windows partition can be extended.

Windows 10 does not include functionality or utility to facilitate this process. However, manufacturers can develop and provide such a utility if PCs are shipped with data partitions.

## Partition layout

The default partition layout for UEFI-based PCs is: a system partition, an MSR, a Windows partition, and a recovery tools partition.

### Disk 0 default partition layout (UEFI-based PCs)



This layout lets you use Windows BitLocker Drive Encryption through both Windows and through the Windows Recovery Environment.

## Sample files: configure drive partitions by using Windows PE and DiskPart scripts

For image-based deployment, boot the PC to [Windows PE](#), and then use the **DiskPart** tool to create the partition structures on your destination PCs.

**Note** In these **DiskPart** examples, the partitions are assigned the letters: System=S, Windows=W, and Recovery=R. The MSR partition does not receive a drive letter.

Change the Windows drive letter to a letter that's near the end of the alphabet, such as W, to avoid drive letter conflicts. Do not use X, because this drive letter is reserved for Windows PE. After the device reboots, the Windows partition is assigned the letter C, and the other partitions don't receive drive letters.

If you reboot, Windows PE reassigns disk letters alphabetically, starting with the letter C, without regard to the configuration in Windows Setup. This configuration can change based on the presence of different drives, such as USB flash drives.

The following steps describe how to partition your hard drives and prepare to apply images. You can use the code in the sections that follow to complete these steps.

### To partition hard drives and prepare to apply images

1. Save the following code in the as a text file (CreatePartitions-UEFI.txt) on a USB flash drive.

```

rem == CreatePartitions-UEFI.txt ==
rem == These commands are used with DiskPart to
rem     create four partitions
rem     for a UEFI/GPT-based PC.
rem     Adjust the partition sizes to fill the drive
rem     as necessary. ==
select disk 0
clean
convert gpt
rem == 1. System partition =====
create partition efi size=100
rem     ** NOTE: For Advanced Format 4Kn drives,
rem             change this value to size = 260 **
format quick fs=fat32 label="System"
assign letter="S"
rem == 2. Microsoft Reserved (MSR) partition =====
create partition msr size=16
rem == 3. Windows partition =====
rem ==     a. Create the Windows partition =====
create partition primary
rem ==     b. Create space for the recovery tools ===
rem         ** Update this size to match the size of
rem             the recovery tools (winre.wim)
rem             plus some free space.
shrink minimum=500
rem ==     c. Prepare the Windows partition =====
format quick fs=ntfs label="Windows"
assign letter="W"
rem == 4. Recovery tools partition =====
create partition primary
format quick fs=ntfs label="Recovery tools"
assign letter="R"
set id="de94bba4-06d1-4d40-a16a-bfd50179d6ac"
gpt attributes=0x8000000000000001
list volume
exit

```

2. Use Windows PE to boot the destination PC.
3. Clean and partition the drive. In this example, *F* is the letter of the USB flash drive.

```
DiskPart /s F:\CreatePartitions-UEFI.txt
```

4. If you use a custom partition layout on Windows 10 for desktop editions, update the push-button recovery script so the recovery tools can recreate the custom partition layout when needed.

## Next steps

Use a deployment script to apply the Windows images on the newly created partitions. For more information, see [Capture and Apply Windows, System, and Recovery Partitions](#).

## Related topics

### Sample scripts

[Configure BIOS/MBR-Based Hard Drive Partitions](#)

[BitLocker Drive Encryption](#)

[WinPE: Install on a Hard Drive \(Flat Boot or Non-RAM\)](#)

[Configuring Disk Mirroring](#)



# BIOS/MBR-based hard drive partitions

1/18/2019 • 6 minutes to read

Create custom partition layouts for your hard disk drives (HDDs), solid-state drives (SSDs), and other drives when deploying Windows to BIOS-based devices.

**Note** If you use a custom partition layout on Windows 10 for desktop editions (Home, Pro, Enterprise, and Education), update the push-button recovery script so the recovery tools can recreate the custom partition layout when needed.

## Partition Requirements

When you deploy Windows to a BIOS-based device, you must format hard drives by using an MBR file system. Windows does not support the GUID partition table (GPT) file system on BIOS-based computers.

An MBR drive can have up to four standard partitions. Typically, these standard partitions are designated as *primary partitions*. For information about how to create additional partitions beyond this limit, see [Configure More than Four Partitions on a BIOS/MBR-Based Hard Disk](#).

### System partition

Each bootable drive must contain a system partition. The system partition must be configured as the active partition.

The minimum size of this partition is 100 MB.

### Windows partition

- This partition must have at least 20 gigabytes (GB) of drive space for 64-bit versions, or 16 GB for 32-bit versions.
- This partition must be formatted using the NTFS file format.
- This partition must have enough 10 GB of free space after the user has completed the Out Of Box Experience (OOBE).
- This partition can have a maximum of 2 terabytes (TB) of space. Software tools to extend the visible partition space beyond 2 TB are not supported on BIOS because they can interfere with software solutions for application compatibility and recovery.

### Recovery tools partition

Create a separate recovery partition to support automatic failover and to support booting Windows BitLocker Drive Encryption-encrypted partitions.

We recommend that you place this partition in a separate partition, immediately after the Windows partition. This allows Windows to modify and recreate the partition later if future updates require a larger recovery image.

The Windows Recovery Environment (Windows RE) tools require additional free space:

- If the partition is less than 500 MB, it must have at least 50 MB of free space.
- If the partition is 500 MB or larger, it must have at least 320 MB of free space.
- If the partition is larger than 1 GB, we recommend that it should have at least 1 GB free.

When calculating free space, note:

- The recovery image, `winre.wim`, is typically between 250-300MB, depending on what drivers, languages, and customizations you add.

- The file system itself can take up additional space. For example, NTFS may reserve 5-15MB or more on a 750MB partition.

## Data partitions

The recommended partition layout for Windows 10 does not include utility or data partitions.

However, if utility or data partitions are required, they should be placed either before the Windows partition or after the Windows RE partition. By keeping the Windows and recovery partitions together, then when future updates of Windows RE area available, Windows will be able to grow the Windows RE partition by shrinking the Windows partition.

This layout makes it more difficult for end users to remove the data partition and merge the space with the Windows partition. For example, the Windows RE partition may need to be moved to the end of the unused space reclaimed from the data partition, so that the Windows partition can be extended. Windows 10 does not include functionality or utility to facilitate this process. However, manufacturers can develop and provide such a utility if PCs are shipped with data partitions.

Each partition can have a maximum of 2 terabytes (TB) of space.

If you're going to be adding more than four total partitions to the disk, see [Configure More than Four Partitions on a BIOS/MBR-Based Hard Disk](#) for more info.

## Partition layout

If you install Windows using a bootable USB key made by Windows Imaging and Configuration Designer (ICD), it creates the following layout by default: a system partition, a Windows partition, and a recovery tools partition.

### Disk 0 default partition layout (BIOS-based PCs)



## System and utility partitions

By default, system partitions do not appear in File Explorer. This helps protect end users from accidentally modifying a partition.

To keep system and utility partitions from being reset, use type 0x27. Do not use any of the following types: 0x7, 0x0c, 0x0b, 0x0e, 0x06, and 0x42.

### To set partitions as utility partitions

1. When you are deploying Windows by using Windows ICD, the partition type will be set automatically.
2. When you are deploying Windows by using the **DiskPart** tool, use the **set id=27** command after you create the partition.

### To verify that system and utility partitions exist

1. Click **Start**, right-click **This PC**, and then click **Manage**. The **Computer Management** window opens.
2. Click **Disk Management**. The list of available drives and partitions appears.
3. In the list of drives and partitions, confirm that the system and utility partitions are present and are not assigned a drive letter.

## Sample files: configuring disk layout by using Windows PE and DiskPart scripts

For image-based deployment, boot the PC to [Windows PE](#), and then use the **DiskPart** tool to create the

partition structures on your destination PCs.

**Note** In these **DiskPart** examples, the partitions are assigned the letters: System=S, Windows=W, and Recovery=R.

Change the Windows drive letter to a letter that's near the end of the alphabet, such as W, to avoid drive letter conflicts. Do not use X, because this drive letter is reserved for Windows PE. After the device reboots, the Windows partition is assigned the letter C, and the other partitions don't receive drive letters.

If you reboot, Windows PE reassigns disk letters alphabetically, starting with the letter C, without regard to the configuration in Windows Setup. This configuration can change based on the presence of different drives, such as USB flash drives.

The following steps describe how to partition your hard drives and prepare to apply images. You can use the code in the sections that follow to complete these steps.

### To partition hard drives and prepare to apply images

1. Save the following code as a text file (CreatePartitions-BIOS.txt) on a USB flash drive.

```
rem == CreatePartitions-BIOS.txt ==
rem == These commands are used with DiskPart to
rem     create three partitions
rem     for a BIOS/MBR-based computer.
rem     Adjust the partition sizes to fill the drive
rem     as necessary. ==
select disk 0
clean
rem == 1. System partition =====
create partition primary size=100
format quick fs=ntfs label="System"
assign letter="S"
active
rem == 2. Windows partition =====
rem ==   a. Create the Windows partition =====
create partition primary
rem ==     b. Create space for the recovery tools
rem         ** Update this size to match the size of
rem             the recovery tools (winre.wim)
rem             plus some free space.
shrink minimum=500
rem ==   c. Prepare the Windows partition =====
format quick fs=ntfs label="Windows"
assign letter="W"
rem == 3. Recovery tools partition =====
create partition primary
format quick fs=ntfs label="Recovery"
assign letter="R"
set id=27
list volume
exit
```

2. Use Windows PE to boot the destination computer.
3. Clean and partition the drive. In this example, F is the letter of the USB flash drive.

```
DiskPart /s F:\CreatePartitions-BIOS.txt
```

4. If you use a custom partition layout on Windows 10 for desktop editions, update the push-button recovery script so the recovery tools can recreate the custom partition layout when needed.

## Next steps

Use a deployment script to apply the Windows images on the newly created partitions. For more information, see [Capture and Apply Windows, System, and Recovery Partitions](#).

## Related topics

[Sample scripts](#)

[Configure More than Four Partitions on a BIOS/MBR-Based Hard Disk](#)

[Configure UEFI/GPT-Based Hard Drive Partitions](#)

[BitLocker Drive Encryption](#)

[Configuring Disk Mirroring](#)

# Configure More than Four Partitions on a BIOS/MBR-Based Hard Disk

1/18/2019 • 3 minutes to read

This topic describes how to configure more than four disk partitions when you deploy Windows on BIOS and master boot record (MBR)-based devices.

## Disk partition rules

- On BIOS-based systems, you can designate one of the four standard partitions as an *extended partition*.

An extended partition is a special partition that can be divided into additional partitions that are called *logical partitions*. An extended partition cannot store files. An extended partition does not receive a partition ID.

- You can include as many logical partitions as your disk can hold.

Logical partitions can store files. You can use a logical partition as the Windows partition.

For additional disk partition rules for BIOS-based systems, see [Configure BIOS/MBR-Based Hard Drive Partitions](#).

### Recommendations

- Add system and utility partitions before you add the Windows partition.
- Add the recovery tools partition immediately after the Windows partition. When you use this partition order, then when future updates to the recovery tools are needed, the partition can be resized automatically.

### Sample partition layout:

Sample partition layout for BIOS-based PCs with more than 4 partitions



## Configuring disk partitions by using a DiskPart script in Windows PE

For image-based deployment, boot the device by using Windows PE, and then use the DiskPart tool to create the partition structures on your destination devices. For more information, see [Apply Images Using DISM](#).

**Note** Windows PE reassigns disk letters alphabetically, beginning with the letter "C", without regard to the configuration in Windows Setup. This configuration can change based on the presence of different drives, including USB flash drives.

In these DiskPart examples, the partitions are assigned the letters "S", "U", "V", "W", and "R" to avoid drive-letter conflicts. After the device reboots, Windows PE automatically assigns the letter "C" to the Windows partition. The Utility1, Utility2, system, and recovery image partitions do not receive drive letters.

The following steps describe how to partition your hard drives and prepare to apply images. You can use the code in the sections that follow to complete these steps.

## To partition hard drives and prepare to apply images

1. Save the code in the following sections as a text file (PrepareMyPartitions.txt) on a USB flash drive.
2. Use Windows PE to boot the destination device.
3. Use the `DiskPart /s F:\PrepareMyPartitions.txt` command, where *F*: is the letter of the USB flash drive, to partition the drives.

## Sample code

Save the following code as "PrepareMyPartitions.txt", and then run the script by using the DiskPart tool to automate the configuration of the Utility1, Utility2, system, extended, Windows, and recovery tools partitions:

```
select disk 0
clean
rem == 1. System partition =====
create partition primary size=100
format quick fs=ntfs label="System"
assign letter="S"
active
rem == 2. Utility partition =====
create partition primary size=100
format quick fs=ntfs label="Utility1"
assign letter="U"
set id=27
rem == 3. Utility partition =====
create partition primary size=200
format quick fs=ntfs label="Utility2"
assign letter="V"
set id=27
rem == 4. Extended partition =====
create partition extended
rem == 4a. Windows partition =====
rem ==     a. Create the Windows partition =====
create partition logical
rem ==     b. Create space for the recovery tools
rem          ** Update this size to match the size of
rem          the recovery tools (winre.wim)
rem          plus some free space.
shrink minimum=500
rem ==     c. Prepare the Windows partition =====
format quick fs=ntfs label="Windows"
assign letter="W"
rem == 4b. Recovery tools partition =====
create partition logical
format quick fs=ntfs label="Recovery"
assign letter="R"
set id=27
list volume
exit
```

## Next Steps

After you create the partitions, you can use a deployment script to apply the Windows images on the newly created partitions. For more information, see [Capture and Apply Windows, System, and Recovery Partitions](#).

## Related topics

[Configure BIOS/MBR-Based Hard Drive Partitions Sample scripts](#)

# Configure multiple hard drives

7/1/2019 • 4 minutes to read

If you are deploying Windows to a computer that has multiple hard drives, you can verify that the image is applied to a specific hard drive by using hardware-specific identifiers such as the location path or the hardware interrupt value.

The location path is a string that specifies the physical location that each drive is connected to the computer, for example: `PCIROOT(0)#PCI(0100)#ATA(C00T00L00)`. When manufacturing a computer, use a consistent physical location when connecting your drives, and then use the location path string to identify each hard drive.

For BIOS-based computers or a computer that is running Virtual Disk Service (VDS), you can use the **SELECT DISK=SYSTEM** and **SELECT DISK=NEXT** commands to select the appropriate hard drive.

## Identifying a drive location path

- Use the DiskPart commands: **list disk** and **select disk <disk number>** (Example: **select disk 1**) to navigate between the drives on your computer.

To show the location path of a selected drive, use the DiskPart command `detail disk`.

In the following example, the location path of the selected drive is  
`PCIROOT(0)#PCI(0100)#ATA(C00T00L00)`.

```
DISKPART> detail disk

HITACHI HTS722016K9SA00
Disk ID: 5E27161A
Type    : ATA
Bus     : 0
Target  : 0
LUN ID  : 0
Location Path : PCIROOT(0)#PCI(0100)#ATA(C00T00L00)
Read-only  : No
Boot Disk  : Yes
PagefileDisk : Yes
Hibernation File Disk : No
CrashdumpDisk : Yes
Clustered Disk  : No

Volume ### Ltr Label      Fs     Type        Size     Status     Info
-----  --  --  -----  -----  -----  -----
Volume 1   C               NTFS   Partition  149 GB  Healthy   System

DISKPART>
```

## Selecting Drives

### Selecting the system drive

1. **BIOS-based computers:** Use the command **SELECT DISK=SYSTEM** to select the default system drive.

This command selects the drive that has an interrupt 13h value of 80h. If the value 80h is assigned to a USB flash drive, this command selects a hard drive that has a value of 81h.

2. **UEFI-based computers:** To select a drive, use the DiskPart command **SELECT DISK=<location path>**.

#### Note

Do not use the **SELECT DISK=SYSTEM** command or the GetSystemDiskNTPath API on Unified Extensible Firmware Interface (UEFI)-based computers to select the system drive. The **SELECT DISK=SYSTEM** command and the GetSystemDiskNTPath API identify the drive that the operating system was booted from as the system drive. If you boot from Windows® PE, this command selects the Windows PE drive as the system drive. If you boot from a system that has multiple drives that include an EFI system partition (ESP), this command may select the wrong drive.

#### Selecting a non-system drive

1. **Select the drive by location path.** To select a drive, use the DiskPart command **SELECT DISK=<location path>**, where *<location path>* is the location path of your drive. This command helps specify a drive by location.

Example:

```
SELECT DISK=PCIROOT(0)#PCI(0100)#ATA(C00T00L00)
```

2. **Select the drive by using the "NEXT" drive.** Use the DiskPart command **SELECT DISK=NEXT**. This command helps specify any remaining hard drives, regardless of location. To select more drives, repeat the **SELECT DISK=NEXT** command to select each drive in order. If there are no more drives to select, DiskPart returns an error.

#### Note

The computer maintains the context for the **SELECT DISK=NEXT** command as long as DiskPart continues running. If DISKPART exits, the computer loses this context.

Example (Two UEFI/GPT drives):

```

SELECT DISK=PCIROOT(0)#PCI(0100)#ATA(C00T00L00)
clean
convert gpt
rem == 1. System partition =====
create partition efi size=100
rem    ** NOTE: For Advanced Format 4Kn drives,
rem           change this value to size = 260 ***
format quick fs=fat32 label="System"
assign letter="S"
rem == 2. Microsoft Reserved (MSR) partition =====
create partition msr size=16
rem == 3. Windows partition =====
rem ==   a. Create the Windows partition =====
create partition primary
rem ==   b. Create space for the recovery tools ===
shrink minimum=500
rem      ** Update this size to match the size of
rem          the recovery tools (winre.wim)
rem          plus some free space.
rem ==   c. Prepare the Windows partition =====
format quick fs=ntfs label="Windows"
assign letter="W"
rem == 4. Recovery tools partition =====
create partition primary
format quick fs=ntfs label="Recovery tools"
assign letter="R"
set id="de94bba4-06d1-4d40-a16a-bfd50179d6ac"
gpt attributes=0x8000000000000001
rem NON-SYSTEM DRIVE =====
SELECT DISK=NEXT
clean
convert gpt
rem == 1. Microsoft Reserved (MSR) partition =====
create partition msr size=16
rem == 2. Data partition =====
create partition primary
format quick fs=ntfs label="Data"
assign letter=z

```

## Identifying the system drive after a reboot

After you reboot, drive lettering may change. You can use the following example script to select the system drive and then reassign letters to the ESP, recovery, and Windows partitions.

```

SELECT DISK=PCIROOT(0)#PCI(0100)#ATA(C01T01L00)
select partition=1
assign letter=s
select partition=2
assign letter=t
select partition=3
assign letter=w

```

## Formatting non-system drives

This example script selects the system drive and then skips past the drive without modifying the contents of the drive. The script then selects two non-system drives and creates a single, formatted, empty partition on each drive. The partitions do not receive an image, so it is not necessary to specifically identify them.

Example (Two UEFI/GPT drives):

```
SELECT DISK=PCIROOT(0)#PCI(0100)#ATA(C01T01L00)
SELECT DISK=NEXT
clean
convert gpt
create partition msr size=16
create partition primary
format quick fs=ntfs label="DataDrive1"
SELECT DISK=NEXT
clean
convert gpt
create partition primary
format quick fs=ntfs label="DataDrive2"
```

## Related topics

[Hard Disk Location Path Format](#)

[DiskPart Command line syntax](#)

# BitLocker Drive Encryption

1/18/2019 • 2 minutes to read

This topic highlights the requirements for deploying a Windows BitLocker Drive Encryption solution. For more information about BitLocker, see [BitLocker Drive Encryption](#) on the TechNet website.

## What Is BitLocker Drive Encryption?

BitLocker provides offline-data and operating-system protection for your computer. BitLocker helps ensure that data that is stored on a computer that is running Windows® is not revealed if the computer is tampered with when the installed operating system is offline. BitLocker uses a microchip that is called a Trusted Platform Module (TPM) to provide enhanced protection for your data and to preserve early boot-component integrity. The TPM can help protect your data from theft or unauthorized viewing by encrypting the entire Windows volume.

BitLocker is designed to offer the most seamless end-user experience with computers that have a compatible TPM microchip and BIOS. A compatible TPM is defined as a version 1.2 TPM that has the BIOS modifications that are required to support the Static Root of Trust Measurement, as defined by the Trusted Computing Group. The TPM interacts with BitLocker to help provide seamless protection when the computer restarts.

The path to the TPM driver file is %WINDIR%\Inf\Tpm.inf. For information about how to add the TPM driver to Windows Preinstallation Environment (Windows PE), see [WinPE: Mount and Customize](#).

## BitLocker Drive Encryption Partitioning Requirements

BitLocker must use a system partition that is separate from the Windows partition. The system partition:

- Must be configured as the active partition.
- Must not be encrypted or used to store user files.
- Must have at least 100 megabytes (MB) of space.
- Must have at least 50 MB of free space.
- May be shared with a recovery partition.

For more information about BitLocker partitioning requirements, see [Hard Drives and Partitions Overview](#).

## Related topics

[Hard Drives and Partitions Overview](#)

# Hard Disk Location Path Format

1/18/2019 • 2 minutes to read

This topic describes the hard disk location-path format. This format is used to identify each disk in the DiskPart tool by using the location path. The location-path format is based on the physical connection to the computer.

For instructions that describe how to configure Windows® to identify a drive based on the location-path format, see [Configure Multiple Hard Drives](#).

## Location-Path Format

The basic syntax for the location path for disks that have a Small Computer System Interface (SCSI), Serial Attached SCSI (SAS), or Redundant Array of Independent Disks (RAID) bus type is as follows:

*<PnP location path of the adapter>#<Bus Type>(P<Path ID>T<Target ID>L<LUN ID>)*

The basic syntax for the location path for disks that have an Advanced Technology Attachment (ATA) or Serial ATA (SATA) bus type is as follows:

*<PnP location path of the adapter>#<Bus Type>(C<Channel ID>T<Target ID>L<LUN ID>)*

The following table defines the elements in the location path.

ELEMENT	DESCRIPTION
<i>&lt;PnP location path of the adapter&gt;</i>	Path of the adapter. Retrieve the path by calling the SetupDiGetDeviceProperty with the DEVPKEY_Device_LocationPaths property.
<i>#&lt;Bus Type&gt;</i>	One of the following types: ATA, SCSI, SAS, or RAID.
<i>P&lt;Path ID&gt;</i>	PathId field of SCSI_ADDRESS. Retrieve the PathID by calling IOCTL_SCSI_GET_ADDRESS.
<i>C&lt;Channel ID&gt;</i>	PathId field of SCSI_ADDRESS. Retrieve the PathID by calling IOCTL_SCSI_GET_ADDRESS. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"><b>Note</b><p>For disks that use the ATA/SATA bus type, the Channel ID refers to the same field as PathID. The prefix C is still used.</p></div>
<i>T&lt;Target ID&gt;</i>	TargetId field of SCSI_ADDRESS. Retrieve the TargetId by calling IOCTL_SCSI_GET_ADDRESS.
<i>L&lt;LUN ID&gt;</i>	Logical Unit Number (LUN) field of SCSI_ADDRESS. Retrieve the LUN by calling IOCTL_SCSI_GET_ADDRESS.

## Examples

The following table gives an example of a location path for each bus or disk type:

BUS OR DISK TYPE	LOCATION PATH
Integrated Development Environment (IDE), ATA, Parallel ATA (PATA), or SATA	PCIROOT(0)#PCI(0100)#ATA(C01T03L00)
SCSI	PCIROOT(0)#PCI(1C00)#PCI(0000)#SCSI(P00T01L01)
SAS	PCIROOT(1)#PCI(0300)#SAS(P00T03L00)
Peripheral Component Interconnect (PCI) RAID	PCIROOT(0)#PCI(0200)#PCI(0003)#PCI(0100)#RAID(P02T00L00)

## Related topics

[Configure Multiple Hard Drives](#)

[DiskPart Command-Line Syntax](#)

# Windows and GPT FAQ

1/24/2019 • 21 minutes to read

Answers to frequently asked questions about the GUID Partition Table (GPT).

This version of the Windows and GPT FAQ applies to Windows 10 and Windows Server 2016. For a previous version of this FAQ, see [Windows and GPT FAQ on MSDN](#).

Since the introduction of the personal computer, the data storage area on a hard disk has been divided into smaller areas called sectors. These sectors are grouped into partitions creating separate volumes, or 'drives' on a disk. The partitions were organized using a scheme called the Master Boot Record (MBR). The MBR is a table of disk locations, or addresses, along with a certain length, of each of the partitions present on the disk. The MBR itself occupies a small amount of the disk and is read during the boot phase to determine where to locate the operating system to boot into. The MBR information is also used by the operating system as a map of the volumes present on the disk.

Eventually, data density for disks became too large for the MBR scheme to account for all the available data locations. Also, the layout, or format, of the MBR was designed for early computers and not flexible enough to accommodate newer disk configurations. A new partitioning method was needed so the GUID Partition Table (GPT) partitioning scheme was created.

## GPT

### What is a GPT disk?

The GUID Partition Table (GPT) was introduced as part of the Unified Extensible Firmware Interface (UEFI) initiative. GPT provides a more flexible mechanism for partitioning disks than the older Master Boot Record (MBR) partitioning scheme that was common to PCs.

A partition is a contiguous space of storage on a physical or logical disk that functions as if it were a physically separate disk. Partitions are visible to the system firmware and the installed operating systems. Access to a partition is controlled by the system firmware before the system boots the operating system, and then by the operating system after it is started.

### What is wrong with MBR partitioning?

MBR disks support only four partition table entries. For more than four partitions, a secondary structure known as an extended partition is necessary. Extended partitions can then be subdivided into one or more logical disks.

Windows creates MBR disk partitions and logical drives on cylinder boundaries based on the reported geometry, although this information no longer has any relationship to the physical characteristics of the hardware (disk driver or RAID controller). Starting with Windows Vista and Windows Server 2008, more logical boundaries are selected when the hardware provides better hints at the true cache or physical alignment. Because this partition information is stored on the drive itself, the operating system is not dependent on the alignment.

MBR partitioning rules are complex and poorly specified. For example, does cylinder alignment mean that each partition must be at least one cylinder in length? An MBR partition is identified by a two-byte field, and coordination is necessary to avoid collision. IBM originally provided that coordination, but today there is no single authoritative list of partition identifiers.

Another common practice is using partitioned or "hidden" sectors to hold specific information by using undocumented processes and results in problems that are difficult to debug. In the past, vendor-specific implementations and tools were released to the public, which made support difficult.

## **Why do we need GPT?**

GPT disks allow for growth. The number of partitions on a GPT disk isn't constrained by temporary schemes such as container partitions as defined by the MBR Extended Boot Record (EBR). The GPT disk partition format is well defined and fully self-identifying. Data critical to platform operation is located in partitions and not in unpartitioned or "hidden" sectors. GPT disks use primary and backup partition tables for redundancy and CRC32 fields for improved partition data structure integrity. The GPT partition format uses version number and size fields for future expansion.

Each GPT partition has a unique identification GUID and a partition content type, so no coordination is necessary to prevent partition identifier collision. Each GPT partition has a 36-character Unicode name. This means that any software can present a human-readable name for the partition without any additional understanding of the partition.

## **Where can I find the specification for GPT disk partitioning?**

Chapter 5 of the Unified Extensible Firmware Interface (UEFI) specification (version 2.3) defines the GPT format. This specification is available at <https://www.uefi.org/specifications>.

## **What is the GPT format for basic disks?**

Basic disks are the most commonly used storage types with Windows. "Basic disk" refers to a disk that contains partitions, such as primary partitions and logical drives, usually formatted with a file system to become a volume for file storage.

The protective MBR area on a GPT partition table exists for backward compatibility with disk management utilities that operate on MBR. The GPT header defines the range of logical block addresses that are usable by partition entries. The GPT header also defines its location on the disk, its GUID, and a 32-bit cyclic redundancy check (CRC32) checksum that is used to verify the integrity of the GPT header. Each entry in the GUID partition table begins with a partition type GUID. The 16-byte partition type GUID, which is similar to a System ID in the partition table of an MBR disk, identifies the type of data that the partition contains and identifies how the partition is used, for example, whether it is a basic disk or a dynamic disk. Note that each GUID partition entry has a backup copy.

For more information about basic disks, see [Basic and Dynamic Disks](#).

## **What is the GPT format for dynamic disks?**

Dynamic disks were first introduced with Windows 2000 and provide features that basic disks don't, such as the ability to create volumes that span multiple disks (spanned and striped volumes) and the ability to create fault-tolerant volumes (mirrored and RAID-5 volumes). Dynamic disks can use the MBR or GPT partition styles on systems that support both. For more information about dynamic disks, see [Basic and Dynamic Disks](#).

## **Is UEFI required for a GPT disk?**

No. GPT disks are self-identifying. All the information needed to interpret the partitioning scheme of a GPT disk is completely contained in structures in specified locations on the physical media.

## **How big can a GPT disk be?**

In theory, a GPT disk can be up to  $2^{64}$  logical blocks in length. Logical blocks are commonly 512 bytes in size.

The maximum partition (and disk) size depends on the operating system version. Windows XP and the original release of Windows Server 2003 have a limit of 2TB per physical disk, including all partitions. For Windows Server 2003 SP1, Windows XP x64 edition, and later versions, the maximum raw partition of 18 exabytes can be supported. (Windows file systems currently are limited to 256 terabytes each.)

## **How many partitions can a GPT disk have?**

The specification allows an almost unlimited number of partitions. However, the Windows implementation restricts this to 128 partitions. The number of partitions is limited by the amount of space reserved for partition entries in the GPT.

## **Can a disk be both GPT and MBR?**

No. However, all GPT disks contain a Protective MBR.

## **What is a Protective MBR?**

The Protective MBR, beginning in sector 0, precedes the GPT partition table on the disk. The MBR contains one type 0xEE partition that spans the disk.

## **Why does the GPT have a Protective MBR?**

The Protective MBR protects GPT disks from previously released MBR disk tools such as Microsoft MS-DOS FDISK or Microsoft Windows NT Disk Administrator. These tools are not aware of GPT and don't know how to properly access a GPT disk. Legacy software that does not know about GPT interprets only the Protected MBR when it accesses a GPT disk. These tools will view a GPT disk as having a single encompassing (possibly unrecognized) partition by interpreting the Protected MBR, rather than mistaking the disk for one that is unpartitioned.

## **Why would a GPT-partitioned disk appear to have an MBR on it?**

This occurs when you use an MBR-only-aware disk tool to access the GPT disk. For more information, see the following questions:

- Can a disk be both GPT and MBR?
- What is a Protective MBR?
- Why does the GPT have a Protective MBR?

# **Windows disk support**

## **Can Windows XP x64 read, write, and boot from GPT disks?**

Windows XP x64 Edition can use GPT disks for data only.

## **Can the 32-bit version of Windows XP read, write, and boot from GPT disks?**

No. The 32-bit version will see only the Protective MBR. The EE partition will not be mounted or otherwise exposed to application software.

## **Can the 32- and 64-bit versions of Windows Server 2003 read, write, and boot from GPT disks?**

Starting with Windows Server 2003 Service Pack 1, all versions of Windows Server can use GPT partitioned disks for data. Booting is only supported for 64-bit editions on Itanium-based systems.

## **Can Windows Vista, Windows Server 2008, and later read, write, and boot from GPT disks?**

Yes, all versions can use GPT partitioned disks for data. Booting is only supported for 64-bit editions on UEFI-based systems.

## **Can Windows 2000, Windows NT 4, or Windows 95/98 read, write, and boot from GPT?**

No. Again, legacy software will see only the Protective MBR.

## **Is it possible to move a GPT disk to another computer?**

You can move, or migrate, data-only GPT disks to other systems that are running Windows XP (64-bit edition only) or later versions of the operating system (32- or 64-bit editions). You can migrate data-only GPT disks after the system has been shutdown or after the safe removal of the disk.

## **What about mixing and matching GPT and MBR disks on the same system?**

GPT and MBR disks can be mixed on systems that support GPT, as described earlier. However, you must be aware of the following restrictions:

- Systems that support UEFI require that boot partition must reside on a GPT disk. Other hard disks can be either MBR or GPT.

- Both MBR and GPT disks can be present in a single dynamic disk group. Volume sets can span both MBR and GPT disks.

### **What about removable media?**

Removable media must be MBR, GPT, or "superfloppy."

### **What is a superfloppy?**

Removable media without either GPT or MBR formatting is considered a "superfloppy". The entire media is treated as a single partition.

The media manufacturer performs any MBR partitioning of removable media. If the media has an MBR, only one partition is supported. There is little user-discriminable difference between MBR-partitioned media and superfloppies.

Examples of removable media include floppy disk drives, JAZ disk cartridges, magneto-optical media, DVD-ROM, and CD-ROM. Hard disk drives on external buses such as SCSI or IEEE 1394 are not considered removable.

What is the default behavior of Windows XP 64-Bit Edition Version 2003 when partitioning media?

For Windows XP 64-Bit Edition Version 2003 only (for Itanium-based systems), fixed disks are partitioned by using GPT partitioning. GPT disks can be converted to MBR disks only if all existing partitioning is first deleted, with associated loss of data.

### **What is the default behavior of the 32-bit version of Windows XP, Windows Server 2003 and Windows XP x64 when partitioning media?**

Only MBR disks can be used.

### **How can a drive letter in the operating system be mapped to a partition in UEFI firmware?**

There is no inherent mapping between drive letter and partition that can be used to determine one from the other. A basic data partition must be identified by its partition GUID.

### **How can an ESP partition be created?**

ESP partitions can be created by using the UEFI firmware utility Diskpart.efi or the Windows command line utility Diskpart.exe.

### **What can be changed on a partition?**

You shouldn't directly change any partition header entry. Don't use disk tools or utilities to make alterations or changes.

### **What partitioning does Windows support on detachable disks?**

Detachable disks are typically expected to migrate between computers or simply to be unavailable to the operating system at times. Examples of detachable disks are USB disks, which can be easily disconnected by the end-user.

Windows XP supports only MBR partitioning on detachable disks. Later versions of Windows support GPT partitions on detachable disks.

For more about removable media, see the following questions:

- What about removable media?
- What is a superfloppy?

## **Windows GPT required partitions: EFI System Partition**

### **What is the Extensible Firmware Interface System Partition (ESP)?**

The ESP contains the NTDR, HAL, Boot.txt, and other files that are needed to boot the system, such as drivers. The Partition GUID defines the ESP:

```
DEFINE_GUID (PARTITION_SYSTEM_GUID, 0xC12A7328L, 0xF81F, 0x11D2, 0xBA, 0x4B, 0x00, 0xA0, 0xC9, 0x3E, 0xC9,  
0x3B)
```

## **Do only GPT Disks have ESPs?**

No, MBR disks can also have ESPs. UEFI specifies booting from either GPT or MBR. The ESP on an MBR disk is identified by partition type 0xEF. However, Windows does not support booting UEFI from MBR disks or 0xEF partitions.

## **How big is the ESP?**

The ESP is approximately 100MBs.

## **Can there be two ESPs on a single disk?**

Such a configuration shouldn't be created, and is not supported in Windows.

## **What about two ESPs on two different disks?**

ESP partitions can be replicated for high-availability configurations. Replication must be done manually and the contents must be synchronized manually when using software volumes. Hardware vendors may provide additional solutions for high availability. ESP partitions cannot be mirrored.

## **What does Microsoft place in the ESP?**

Microsoft places the HAL, loader, and other files that are needed to boot the operating system in the ESP.

## **Where should the ESP be placed on the disk?**

The ESP should be first on the disk. The primary benefit to placing the ESP first, is that it is impossible to span volumes when the ESP is logically between the two data partitions that you are attempting to span.

## **What should a system or device manufacturer place in the ESP?**

The ESP should only include files that are required for booting an operating system, platform tools that run before operating system boot, or files that must be accessed before operating system boot. For example, files that are required for performing pre-boot system maintenance must be placed in the ESP.

Other value-add files or diagnostics used while the operating system is running should not be placed in the ESP. It is important to note that the space in the ESP is a limited system resource; its primary purpose is to provide storage for the files that are needed to boot the operating system.

## **Where should a system manufacturer place files such as platform diagnostics or other value-added files?**

The preferred option is for system manufacturers to place value-add contents in an OEM-specific partition. Just like MBR OEM partitions, the contents of GPT OEM (or other unrecognized) partitions are not exposed (given drive letters or returned in volume lists). Users are warned that deleting the partition can cause the system to fail to operate. An OEM-specific partition should be placed before the MSR and after any ESP on the disk. Although not architectural, this placement has the same benefits as placing the ESP first. For example, it is also impossible to span volumes when an OEM-specific partition is logically between the two data partitions that you are attempting to span.

Placement in the ESP is an option for applications or files that execute in the pre-operating system boot environment. However, the ESP is architecturally shared space and represents a limited resource. Consuming space in the ESP should be considered carefully. Files that are not relevant to the pre-operating system boot environment should not be placed in the ESP.

## **What is a Microsoft Reserved Partition (MSR)?**

The Microsoft Reserved Partition (MSR) reserves space on each disk drive for subsequent use by operating system software. GPT disks do not allow hidden sectors. Software components that formerly used hidden sectors now allocate portions of the MSR for component-specific partitions. For example, converting a basic disk to a dynamic disk causes the MSR on that disk to be reduced in size and a newly created partition holds the dynamic disk database. The MSR has the Partition GUID:

```
DEFINE_GUID (PARTITION_MSFT_RESERVED_GUID, 0xE3C9E316L, 0x0B5C, 0x4DB8, 0x81, 0x7D, 0xF9, 0x2D, 0xF0, 0x02,  
0x15, 0xAE)
```

## What disks require an MSR?

Every GPT disk must contain an MSR. The order of partitions on the disk should be ESP (if any), OEM (if any) and MSR followed by primary data partition(s). It is particularly important that the MSR be created before other primary data partitions.

## Who creates the MSR?

The MSR must be created when disk-partitioning information is first written to the drive. If the manufacturer partitions the disk, the manufacturer must create the MSR at the same time. If Windows partitions the disk during setup, Windows creates the MSR.

## Why must the MSR be created when the disk is first partitioned?

After the disk is partitioned, there will be no free space left to create an MSR.

## How big is the MSR?

When initially created, the size of the MSR depends on the size of the disk drive:

- On drives less than 16GB in size, the MSR is 32MB.
- On drives greater than or equal to 16GB, the MSR is 128 MB.

As the MSR is divided into other partitions, it becomes smaller.

# Windows GPT ESP implementation

## What partitions are required by Windows?

For UEFI systems, the boot drive must contain an ESP, an MSR, and at least one basic data partition that contains the operating system. Only one ESP should exist on a system even if multiple operating systems are installed on that system. In a mirrored boot configuration there may actually be two drives with an ESP but they are considered to be a redundant copy of the same ESP. Each data drive must contain at least an MSR and one basic data partition.

All basic data partitions on the drive should be contiguous. As noted above, placing an OEM-specific or other unrecognized partition between data partitions imposes limitations on later volume spanning.

## What is a basic data partition?

Basic data partitions correspond to primary MBR partitions 0x6 (FAT), 0x7 (NTFS), or 0xB (FAT32). Each basic partition can be mounted using a drive letter or mount point, other volume device object, or both. Each basic data partition is represented in Windows as a volume device object, and optionally as a mount point or a drive letter.

## How is a basic data partition identified?

It has the following partition type GUID:

```
DEFINE_GUID (PARTITION_BASIC_DATA_GUID, 0xEB0A0A2L, 0xB9E5, 0x4433, 0x87, 0xC0, 0x68, 0xB6, 0xB7,  
0x26, 0x99, 0xC7);
```

## Will end-users see the ESP partition?

The ESP partition isn't hidden, but also doesn't have an assigned drive letter. It will not appear in Explorer unless a drive letter gets assigned to it, but some tools will be able to list it.

## Will end-users see the MSR and OEM-specific partitions?

Users will not see these partitions exposed in Windows Explorer, nor is any recognized file system exposed to legacy programs such as Context Indexing. The OEM-specific and other unrecognized partitions will be visible only in the Disk Management MMC snap-in since they will not have a recognizable file system.

## **What partitions are mounted by default by Windows?**

Windows exposes only basic data partitions. Other partitions with FAT file systems may be mounted, but not exposed only programmatically. Only basic data partitions are assigned drive letters or mount points.

The ESP FAT file system is mounted, but not exposed. This allows programs running under Windows to update the contents of the ESP. Assigning a drive letter to the ESP using `mountvol /s` will allow access to the partition.

Access to the ESP requires admin privilege. Although the MSR, and any partitions created from the MSR, could have recognizable file systems, none are exposed.

Any OEM-specific partitions or partitions associated with other operating systems are not recognized by Windows. Unrecognized partitions with recognizable file systems are treated like the ESP. They will be mounted, but not exposed. Unlike MBR disks, there is no practical difference between OEM-specific partitions and other operating system partitions; all are "unrecognized."

## **How can the user see the ESP, OEM, and other unrecognized partitions?**

The user can use disk management tools such as the Disk Management utility or the `diskpart.exe` Windows command line. The MSR and any partitions created from the MSR are only visible from the command line.

## **What about dynamic disks?**

Dynamic disks use two different GPT partitions?

- A data container partition that corresponds to the MBR partition 0x42, with the following GUID:

```
DEFINE_GUID (PARTITION_LDM_DATA_GUID, 0xAF9B60A0L, 0x1431, 0x4F62, 0xBC, 0x68, 0x33, 0x11, 0x71, 0x4A,  
0x69, 0xAD)
```

```
;
```

- A partition to contain the dynamic configuration database, with the following GUID:

```
DEFINE_GUID(PARTITION_LDM_METADATA_GUID, 0x5808C8AAL, 0x7E8F, 0x42E0, 0x85, 0xD2, 0xE1, 0xE9, 0x04,  
0x34, 0xCF, 0xB3
```

```
);
```

Volumes are created in the data container and mounted by default. Again, this is exactly the same as the contents of 0x42 MBR partitions.

## **What happens when a basic disk is converted to dynamic?**

For a drive to be eligible for conversion to dynamic, all basic data partitions on the drive must be contiguous. If other unrecognized partitions separate basic data partitions, the disk can't be converted. This is one of the reasons that the MSR must be created before any basic data partitions. The first step in conversion is to separate a portion of the MSR to create the configuration database partition. All non-bootable basic partitions are then combined into a single data container partition. Boot partitions are retained as separate data container partitions. This is analogous to conversion of primary partitions.

Windows XP and later versions of Windows differ from Windows 2000 in that basic and extended partitions are preferentially converted to a single 0x42 partition, rather than being retained as multiple distinct 0x42 partitions as on Windows 2000.

## **Can a system contain a mix of GPT and MBR dynamic disks?**

Yes. For more information, see What about mixing and matching GPT and MBR disks on the same system?

## **How can a specific partition be mounted?**

You can access the GPT disk partitions of different types using the tools that are listed in the following table.

TOOL	WINDOWS	FIRMWARE
Diskpart.efi Disk Partition Tool		ESP MSR Data

TOOL	WINDOWS	FIRMWARE
Diskpart.exe Disk Partition Tool	ESP MSR Data	
Diskmgmt.msc Logical Disk Manager	ESP Data	
Explorer.exe File Explorer	Data	

By using the Microsoft Platform SDK APIs, you can also develop your own tools to access the GPT disk partitions at their primitive levels.

### How are GPT disks managed in Windows?

GPT and MBR disks are managed the same way. Disks can be formatted as GPT or MBR by using the Diskpart.exe command prompt utility or by using the Disk Administrator snap-in. Volumes can be created on both GPT and MBR disks, and both kinds of disks can be mixed in the same dynamic disk group.

### What about FTdisk sets?

Starting with Windows XP, there is no FTdisk set support on Windows for MBR or GPT disks. The only support for logical volumes is through dynamic disks.

### Can a disk be converted from GPT to MBR, and vice versa?

Yes, Microsoft offers [MBR2GPT.exe](#) which converts disks from MBR to GPT.

### What file systems are supported on GPT disks?

NTFS is recommended on all basic data partitions and all dynamic volumes. Windows Setup and the Disk Management snap-in offer only NTFS. To circumvent that, the partition or volume must be formatted explicitly via the Format command-line tool.

## Manipulating GPT disks and their contents

### How do I create a GPT disk?

You can create a GPT disk only on an empty, unpartitioned disk (raw disk or empty MBR disk). For more information about creating GPT disks, see [Using GPT Drives](#).

### How do I convert an MBR or GPT disk?

You can convert an existing partition format to another format. For more information, see the following TechNet articles:

- [Change a Master Boot Record Disk into a GUID Partition Table Disk](#)
- [Change a GUID Partition Table Disk into a Master Boot Record Disk](#)

### Is it possible to make a sector-by-sector copy of a GPT disk?

No. The Disk and Partition GUIDs will no longer be unique. This must never happen. You can make a sector-by-sector copy of the contents of ESP or basic data partitions.

### Is there any way to copy a whole GPT disk using the OPK imaging tools?

Yes. However, there are some key caveats. The OEM Preinstallation Kit (OPK) initializes the Disk and Partition GUIDs to zero. On first boot of Windows, the operating system generates unique GUIDs. The OPK only supports generation of ESP, MSR, and basic data partitions.

If an application has recorded any Disk or Partition GUIDs it may break. Any applications, drivers, utilities, or firmware implementations supplied by system manufacturers or application vendors that rely on GUIDs should be capable of handling GUIDs that change from the OPK initialization values to those generated by the operating system.

### **What is the Diskpart.efi MAKE command?**

It is a way for OEMs to simplify operating system preinstallation and system recovery. This command can easily be extended to create a "default" disk configuration for the platform. For example, the system manufacturer could extend the MAKE command to automatically partition the boot drive with an ESP, MSR, an OEM-specific partition, and one basic data partition.

For example, consider a possible disk configuration called BOOT\_DISK. In the event of business failure recovery, MAKE BOOT\_DISK would allow the customer to completely repartition a boot disk to the original factory defaults.

### **What happens if a duplicate Disk or Partition GUID is detected?**

Windows will generate new GUIDs for any duplicate Disk GUID, MSR Partition GUID, or MSR basic data GUID upon detection. This is similar to the duplicate MBR signature handling in Windows 2000. Duplicate GUIDs on a dynamic container or database partition cause unpredictable results.

# Customize

1/18/2019 • 2 minutes to read

This section covers how to customize a Windows installation.

## In this section

TOPIC	DESCRIPTION
<a href="#">Understanding Servicing Strategies</a>	Learn about strategies when servicing (modifying) a Windows image
<a href="#">Audit Mode</a>	Learn how to make changes to a running PC prior to shipping a device
<a href="#">Apps</a>	Learn how to preload Microsoft Store apps on a Windows image
<a href="#">Drivers</a>	Learn how to work with drivers
<a href="#">Configuration and settings</a>	Various configuration options when deploying Windows
<a href="#">Features</a>	Learn how to add and remove Features on Demand and legacy Windows features
<a href="#">OOBE</a>	Configure OOBE to show a custom experience the first time a user boots their PC

# Apps

1/18/2019 • 2 minutes to read

This section covers how to preinstall apps in a Windows image

## In this section

TOPIC	DESCRIPTION
<a href="#">Siloed provisioning packages (SPPs)</a>	Learn how to capture apps and/or settings in a siloed provisioning package that you can deploy after applying an image
<a href="#">Create a provisioning package with Windows desktop applications</a>	Create a provisioning package that can be added to a Windows image
<a href="#">Sideload Apps with DISM</a>	How to sideload apps into a Windows image
<a href="#">Preinstall Apps Using DISM</a>	Learn how to preinstall apps in a Windows image
<a href="#">Export or Import Default Application Associations</a>	Learn how to migrate default app associations
<a href="#">Microsoft .NET Framework 3.5 Deployment Considerations</a>	How to deploy .Net 3.5 to support applications

# Siloed provisioning packages

1/18/2019 • 21 minutes to read

Siloed provisioning packages are a new type of provisioning package that is available for Windows 10, version 1607. Where traditional provisioning packages can capture all classic Windows applications and settings that are installed with a Windows image, a siloed provisioning package can capture classic Windows applications individually, drivers plus applications, settings, or capture add-ons for provisioning packages that were captured previously. This provides more flexibility for the manufacturing process and helps reduce the time required to build Windows-based computers in the factory.

## Performance comparison

The following table shows a comparison between using the Office installer vs using siloed provisioning packages in a typical factory floor process. When using the siloed provisioning packages to install Office, the base Office en-us package, along with the add-on Office fr-fr and Office de-de packages are captured using the User State Migration Tool (USMT) ScanState.exe utility as a one-time process in the imaging lab. The data in the following table was derived from a sample run on a VM with Windows 10, version 1607 desktop image. The actual time savings at the factory floor will vary based on the number and size of applications being installed and the hardware spec of physical devices. The time savings can be calculated by:

(time to Sysprep & boot to Audit mode + time to install applications + time to capture applications in a PPKG + <optional> time to single-instance the PPKG) – (time to apply SPPs + time to Sysprep & boot to Audit mode)

PHASE/TASK	FACTORY PROCESS USING OFFICE INSTALLER	FACTORY PROCESS USING SILEOED PROVISIONING PACKAGES
Apply image to the device	4 min.	4 min.
Install language packs – fr-fr & de-de	20 min.	20 min.
Run BCDBoot.exe	negligible	negligible
Run DISM to apply siloed Office en-us base, Office fr-fr, and Office de-de packages	N/A	3 min.
Run Sysprep & boot to Audit mode	10 min.	19 min.
Install Office 2016 en-us, fr-fr, & de-de	12 min.	N/A
Run ScanState to capture Office into provisioning package (for PBR)	10 min.	N/A
(Optional – for low disk space) Single-instancing Office files captured in the provisioning package	7 min.	N/A
<b>Total</b>	<b>56-63 min.</b>	<b>46 min.</b>
<b>Overall application install time</b>		<b>45-65% faster</b>

PHASE/TASK	FACTORY PROCESS USING OFFICE INSTALLER	FACTORY PROCESS USING SILEOED PROVISIONING PACKAGES
<b>Overall E2E deployment time</b>		<b>18-30% faster</b>

## Work with siloed provisioning packages

To create and deploy siloed provisioning packages, you'll need to copy binaries from various folders in the ADK install location that enable DISM and ScanState to work with SPPs. To facilitate the copy process, a script (CopyDandI.cmd) is included in the Windows ADK when 'Deployment Tools' option is selected at install. Run the script to copy all necessary files to an output folder, for example D:\ADKTools:

```
<%Windows ADK install root%>\Deployment Tools\CopyDandI.cmd amd64 D:\ADKTools
```

Before you use DISM, you'll need to copy the ADK tools again to a non-removable drive on the destination device. Copying the file to a non-removable location avoids an error associated with installing DISM from removable drives.

```
xcopy D:\ADKTools\ W:\ADKTools\ /s
```

You'll then have to install the tools:

```
W:\ADKTools\amd64\WimMountAdkSetupAmd64.exe /Install /q
```

And then run DISM from that location:

```
W:\ADKTools\amd64\DISM.exe /Apply-SiloedPackage /ImagePath:C:\ /PackagePath:e:\repository\SPP_base.spp
/PackagePath:e:\repository\SPP_AddOn1.spp /PackagePath:e:\repository\SPP_AddOn2.spp
```

You'll use ScanState to capture siloed provisioning packages from a booted Windows installation, and DISM to apply SPPs to an applied Windows image from WinPE.

For the full walkthrough, see [Lab 10: Add desktop applications and settings with siloed provisioning packages](#).

## Create siloed provisioning packages

This section covers how to use ScanState.exe from the Windows desktop to create siloed provisioning packages that contain applications, system settings, and drivers.

### Recommendations:

- Use a clean Windows installation. This prevents any potentially unwanted settings from being included in the package.
- Use virtual machines with checkpoints to quickly capture SPPs and then revert to a clean Windows installation.

### Configuration files

ScanState for Windows 10, version 1607 can now capture individual Windows desktop applications, and by default only captures components from the Windows namespace. You can choose what gets captured by using /apps by using configuration files. The configuration files that are installed with the Windows Assessment and Deployment Toolkit, along with the User State Migration Tool (USMT), is here:

<%Windows ADK install root%>\User State Migration Tool\<arch>\

Customize these files to capture more or fewer components in the siloed provisioning package:

CONFIGURATION FILE	USAGE
Config_AppsOnly.xml	Captures Windows desktop applications and application settings, without other non-relevant settings. This is typically used to create an inventory of applications that can be deployed in the final factory floor process.
Config_SettingsOnly.xml	Captures only system settings. This is typically used to create an inventory of applications that can be deployed in the final factory floor process.
Config_AppsAndSettingsOnly.xml	Captures both desktop applications and system settings. This can be used either while creating an inventory of applications or as the last step in the factory floor process.

### Capture Windows desktop applications

The following example uses a configuration file to create a siloed provisioning package that contains Windows desktop applications installed on a reference device:

```
ScanState.exe /apps:-sysdrive /o /v:13 /config:Config_AppsOnly.xml /ppkg e:\repository\SPP_base.spp  
/l:C:\Scanstate.log
```

Here are what the parameters for the above command mean:

PARAMETER	DESCRIPTION
/apps	Tells ScanState to capture desktop applications.
-sysdrive (or +sysdrive)	Tells ScanState to ignore all the folders outside the Windows namespace. For example, if there is a folder c:\Folder, that folder will be captured when running with /apps (or /apps:+sysdrive) but it will not be captured when running with /apps:-sysdrive.  Typically you use +sysdrive if you want to capture the entire state of the machine into a single siloed provisioning package; use -sysdrive if you want to capture a single application (or a small group of applications).  The Windows namespace is the set of folders created by a Windows installation, typically: <ul style="list-style-type: none"><li>• %systemdrive%\Users</li><li>• %systemdrive%\ProgramData</li><li>• %systemdrive%\Program Files</li><li>• %systemdrive%\Program Files (x86)</li><li>• %systemdrive%\Windows</li><li>• %systemdrive%\Inetpub</li></ul>
/o	Overwrites any existing data in the store. If not specified, ScanState will fail if the store already contains data.

PARAMETER	DESCRIPTION
/v:13	Produces a MigLog.xml file that indicates what gets captured.
/diff	Used with the /apps command option to capture application add-on components relative to parent applications already captured in siloed provisioning packages.
/l:ScanState.log	Tells ScanState where to save log files. When used in combination with <code>/v:13</code> , ScanState will save MigLog.xml to the same folder as ScanState.log

### Capture add-on components

You can use /diff to create a siloed provisioning package that captures components relative to parent applications already captured in a siloed provisioning package. To use /diff, you must specify a SPP to compare the capture against:

```
ScanState.exe /apps:-sysdrive /o /v:13 /config:Config_AppsOnly.xml /diff:e:\repository\SPP_base.spp /ppkg e:\repository\SPP_AddOn1.spp
```

### Capture system settings

In a scenario where all deployment tasks for a device are complete and no Windows desktop applications have been installed, you can use ScanState to capture system settings that have not been captured in any other SPP by using Config\_SettingsOnly.xml. The SPP can be placed directly into the recovery folder during capture for use during Push-Button Reset.

The following example creates a siloed provisioning package of only system settings on a device, and places it into the recovery folder:

```
ScanState.exe /apps:-appfiles /o /v:13 /config:Config_SettingsOnly.xml /ppkg %systemdrive%\Recovery\Customizations\systemsettings.spp
```

### Capture system settings and Windows desktop applications in the same package

Config\_AppsAndSettingsOnly.xml is intended to capture Windows desktop applications and system settings that are installed last-minute, so they can be placed in the recovery folder for use during PBR. For example, after a device is booted into Audit mode on the factory floor, additional Win32 apps are installed and need to be captured. In this case, you have two options:

- Capture the additional apps and their relevant settings in one .spp using the /diff switch and Config\_AppsOnly.xml. Then capture the system settings in a separate .spp using the Config\_SettingsOnly.xml.
- Capture the additional apps and the system settings into one SPP using the /diff switch and Config\_AppsAndSettings.xml.

Config\_AppsAndSettingsOnly.xml can also be used when you want to capture all apps and settings into one .spp file, for use in an imaging lab or on the factory floor.

### Capture drivers

This section covers how to capture different types of drivers with ScanState.

#### Types of drivers

ScanState for Windows 10, version 1607 captures 3rd-party drivers when you use the /drivers switch. By default, ScanState.exe captures all 3rd party drivers, but can also capture a subset of drivers based on .inf name, manufacturer, or class. Some driver types, like filter drivers, may not be captured when using /drivers. In this case,

run Scanstate.exe with using /apps. The /drivers switch can also be used in combination with /apps in situations where you want to capture drivers and their associated management software, such as for printers or video cards.

## Hardware drivers

To capture drivers that are installed using an .inf file, use the /drivers switch. It's not necessary to use the /apps switch.

To capture drivers that are installed using another method (for example, a setup.exe file), use both /drivers and /apps. This ensures that both the driver package and all Windows desktop applications and settings created by setup for that driver are captured at the same time. To filter out other driver packages, use the arguments that are in combination with /drivers.

## Other drivers

Drivers such as filter drivers are not captured using the /drivers switch. To capture these types of drivers, use only the /apps switch.

### Capture drivers by using patterns

Because ScanState.exe captures all 3rd-party drivers by default when using `scanState.exe /drivers`, if you want to capture only certain drivers, you have to use patterns to narrow the number of captured drivers. ScanState processes commands left to right, so the last pattern specified in a command will take precedence over previous patterns in the same command. For example, if you want to only capture a specific set of drivers, you would first exclude all drivers from capture, and then include specific drivers. Since the arguments are processed in order, the drivers specified after the exclusion of all drivers will be captured.

Here are the patterns you can use to select which drivers will be captured:

PATTERN	DESCRIPTION
+n	Selects drivers to be included based on inf name
-n	Selects drivers to be excluded based on inf name
+p	Selects drivers to be included based on publisher name
-p	Selects drivers to be excluded based on publisher name
+c	Selects drivers to be included based on class name or class GUID
-c	Selects drivers to be excluded based on class name or class GUID

The following example uses a pattern to create a siloed provisioning package that contains drivers of a specific class.

```
ScanState.exe /drivers:-n:* /drivers:+c:{4d36e96f-e325-11ce-bfc1-08002be10318} /ppkg e:\repository\drivers.spp
```

Here are what the parameters for the above command mean:

PARAMETER	DESCRIPTION
/drivers	Tells ScanState to capture 3rd party drivers.

PARAMETER	DESCRIPTION
-n:*	Removes all drivers from capture.
+c:{4d36e96f-e325-11ce-bfc1-08002be10318}	Adds drivers of a particular class back into the capture.
/ppkg	Specifies that the output will be a ppkg. This is required for use with /drivers.

## Capture Applications and Drivers in the same SPP

You can use the /apps and /drivers switches in the same command to create SPPs that contain both applications and drivers.

The following is an example of capturing a siloed provisioning package that contains Windows desktop applications and drivers with only with a specific class GUID.

```
ScanState.exe /drivers:-n:* /drivers:+c:{4d36e96f-e325-11ce-bfc1-08002be10318} /apps:-sysdrive /o /v:13
/config:Config_AppsOnly.xml /ppkg e:\repository\apps_and_drivers.spp
```

## Apply siloed provisioning packages

This section covers how to use DISM from WinPE to apply siloed provisioning packages.

DISM supports applying siloed provisioning packages to a Windows image through a new DISM provider, which is only available through the Windows ADK. You can get this version of DISM by using CopyDandl.cmd.

### Limitations

The functionality for applying siloed provisioning packages using DISM is limited to support the following scenarios:

- The DISM SiloedPackageProvider is not included in the Windows image, nor is it included in Windows PE, version 1607. The Windows ADK version of DISM must be installed on the servicing host, and then launch DISM.exe from the Windows ADK installed location. On a host that is not supported by Windows ADK installer, such as Windows PE, the required binaries can be copied to the host using the CopyDandl.cmd script in <%Windows ADK install root%>\Deployment Tools.
- DISM only supports applying siloed provisioning packages to a Windows image that has been applied at the root of a disk volume on a device, e.g. "C:\". It does not support applying siloed provisioning packages to a Windows image that is mounted for offline servicing. The typical scenario is booting the device to Windows PE, and running the Windows ADK version of DISM in Windows PE to apply siloed provisioning packages after the Windows image has been applied to the device.
- The DISM command to apply siloed provisioning packages to a Windows image (DISM Apply-SiloedPackage) can be run only once. All of the siloed provisioning packages to be applied to the Windows image must be specified in the right order in a single command operation. The order of the installation will be preserved, so the packages can be restored in the same order during PBR.
- If additional siloed provisioning packages need to be applied to a Windows desktop image that has already gone through the entire deployment process with using DISM to apply a set of siloed provisioning packages, the image can be Sysprep generalized and captured as a new model image. DISM can then be run again to apply more siloed provisioning packages when this new model image is deployed onto other devices.
- Siloed provisioning packages must be applied to the same operating system architecture that they were captured on. For example, capturing an app on an x86 operating system in an .spp and applying it to an x64 operating system is not supported.
- Siloed provisioning packages can be applied to other editions of Windows. For example, an application

captured on Windows 10 Enterprise can be applied to Windows 10 Pro.

- Windows 10, Version 1607, does not support applying siloed provisioning packages on a generalized image that is set to boot into Audit mode. If booting into Audit mode is required, use Unattend.xml to reseal to Audit mode.

## Use DISM to apply siloed provisioning packages

The following example uses DISM that is created by CopyDandl.cmd to apply a base SPP, as well as two add-on SPPs:

```
DISM.exe /Apply-SiloedPackage /ImagePath:C:\ /PackagePath:e:\repository\BaseSPP.spp  
/PackagePath:e:\repository\SPP_AddOn1.spp /PackagePath:e:\repository\SPP_AddOn2.spp
```

For syntax, see [DISM Image Management Command-Line Options](#), or run `DISM.exe /Apply-SiloedPackage /?` from the target location of CopyDandl.cmd.

All of the siloed provisioning packages applied by DISM will be placed in `%systemdrive%\Recovery\Customizations` folder.

### Saving drive space: single-instancing is automatic on compact OS

When DISM applies siloed provisioning packages to the OS image that has been applied as Compact OS on a device, by default the packages will be applied with application files single-instanced (using WIMBoot v1 style) on the device.

To single-instance your siloed provisioning packages on devices without Compact OS image, use DISM /Apply-CustomDataImage while the device is booted into Windows PE.

```
DISM.exe /ImagePath:C:\ /Apply-CustomDataImage /CustomDataImage:C:\Recovery\Customizations\myApp.spp  
/SingleInstance
```

The /Apply-SiloedPackage command works with both traditional provisioning packages and siloed provisioning packages (.spp). -If you create provisioning packages in audit mode, you can choose to single-instance contents by using the DISM /Apply-CustomDataImage /SingleInstance command. To learn more see [Lab 1g: Make changes from Windows \(audit mode\)](#).

### Push-button reset

When using ScanState to capture traditional provisioning packages, only one package with all the applications and system settings can be placed in `%systemdrive%\Recovery\Customizations` folder. During push-button reset (PBR), the single provisioning package is processed to restore the applications and system settings.

Beginning with Windows 10, version 1607, applications can be captured in multiple siloed provisioning packages and system settings can also be captured in a separate siloed provisioning package. As a result, PBR is enhanced to allow multiple siloed provisioning packages to be applied, in the preserved order in which they were applied using Dism /Apply-Siloed Package. The packages can then be queued and processed in the right order during PBR to restore the applications and system settings captured in these packages. If the packages were applied using single-instancing, it will be honored when PBR restores them to the device.

Single-instancing can occur automatically if Compact OS is used, or manually.

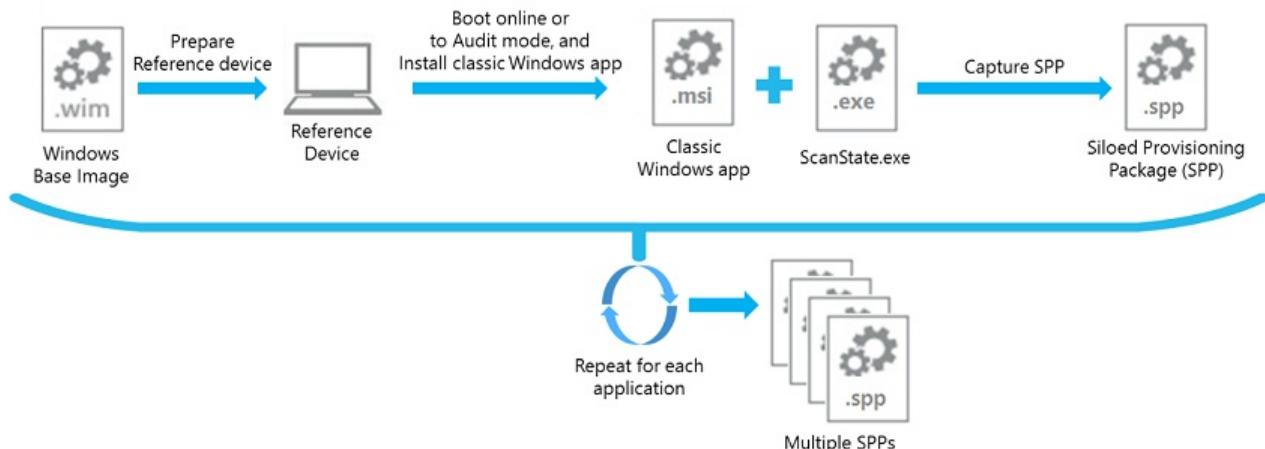
- If you use WinPE, then applying an image as Compact OS, then apply SPPs to it, Windows automatically single-instances the contents of the package. To learn more, see [Lab 10: Add desktop applications and settings with siloed provisioning packages \(SPPs\)](#)
- If you create provisioning packages in audit mode, you can choose to single-instance the contents by using the DISM /Apply-CustomDataImage /SingleInstance command. To learn more, see [Lab 9: Make changes from Windows \(audit mode\)](#).

# Scenarios for using siloed provisioning packages

This section covers scenarios using siloed provisioning packages.

## Capturing and applying independent applications

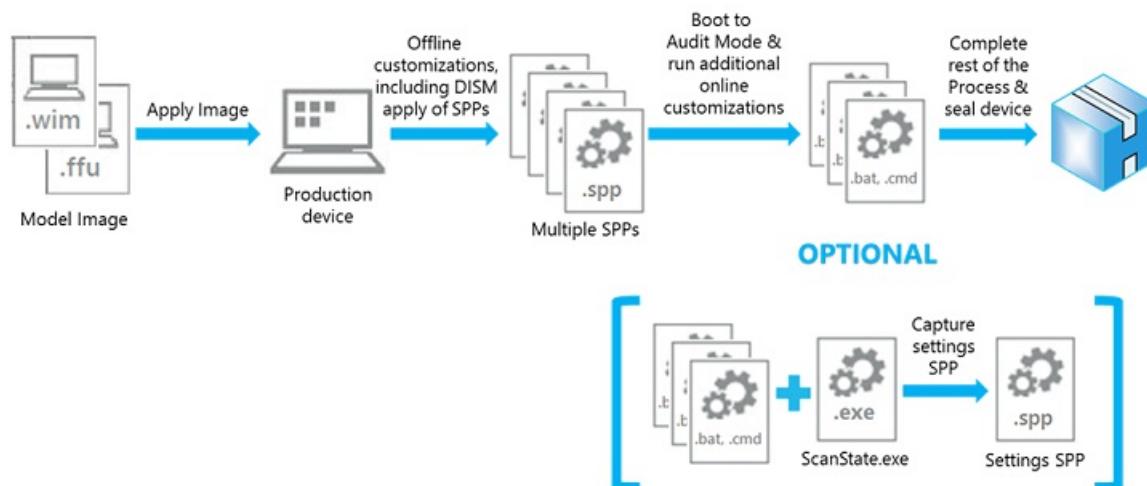
A Microsoft partner can capture siloed provisioning packages of individual classic Windows applications while in the imaging lab, and then install any combination of siloed provisioning packages in a customized order at factory floor. For example, a partner could capture siloed provisioning packages for a PDF reader application and an antivirus program, and then install those program packages on a specific device model at factory floor.



1. Clean install Windows 10, version 1607 on a reference device.
2. At the desktop, install antivirus software.
3. Run ScanState.exe to capture an antivirus software siloed provisioning package.
4. Wipe and clean install the reference device
5. Repeat steps 2-4 for the PDF reader application.

Alternatively, the siloed provisioning packages can be captured using a VM instead of a physical device:

1. Create a VM and boot it online using a Windows 10, version 1607 VHD/VHDX image.
2. Create a checkpoint of the clean OS installation on the VM.
3. At the desktop, install the antivirus software.
4. Run ScanState.exe to capture an antivirus software siloed provisioning package.
5. Revert the VM to the checkpoint.
6. At the desktop, install the PDF reader application.
7. Run ScanState.exe to capture the PDF reader application siloed provisioning package.



1. On the target device, boot to Windows PE, and apply the Windows 10, version 1607, desktop image.

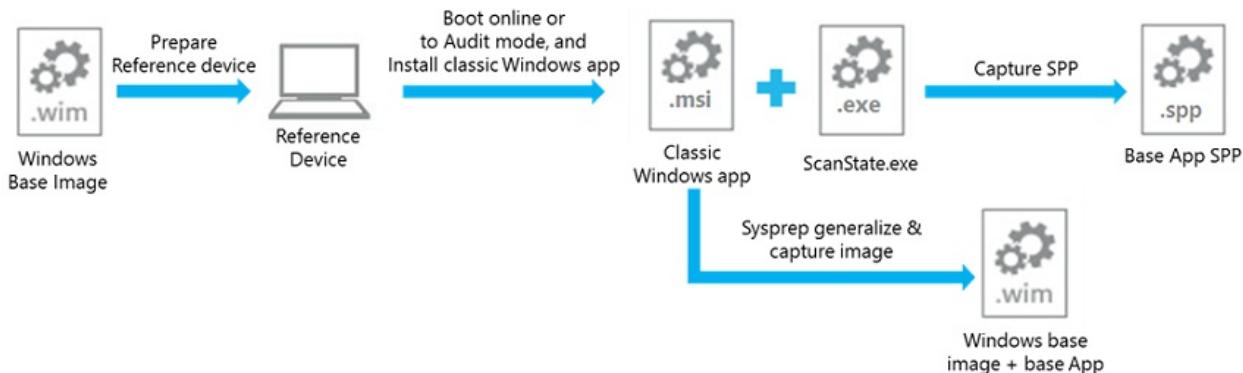
2. While in Windows PE, run DISM /Apply-SiloedPackage command with the PDF reader and antivirus program packages to apply the application files in the packages onto the applied desktop image.
3. Complete the rest of the offline customization tasks.
4. Go through first boot and run through specialize to get to Audit mode.
5. Complete the online customization/configuration tasks.
6. (Optional) While in Audit mode, run ScanState to capture only the system settings into siloed provisioning package and place it in the recovery folder.
7. Complete the rest of the factory floor tasks and shutdown/seal the product.

### Capturing and applying applications with dependencies

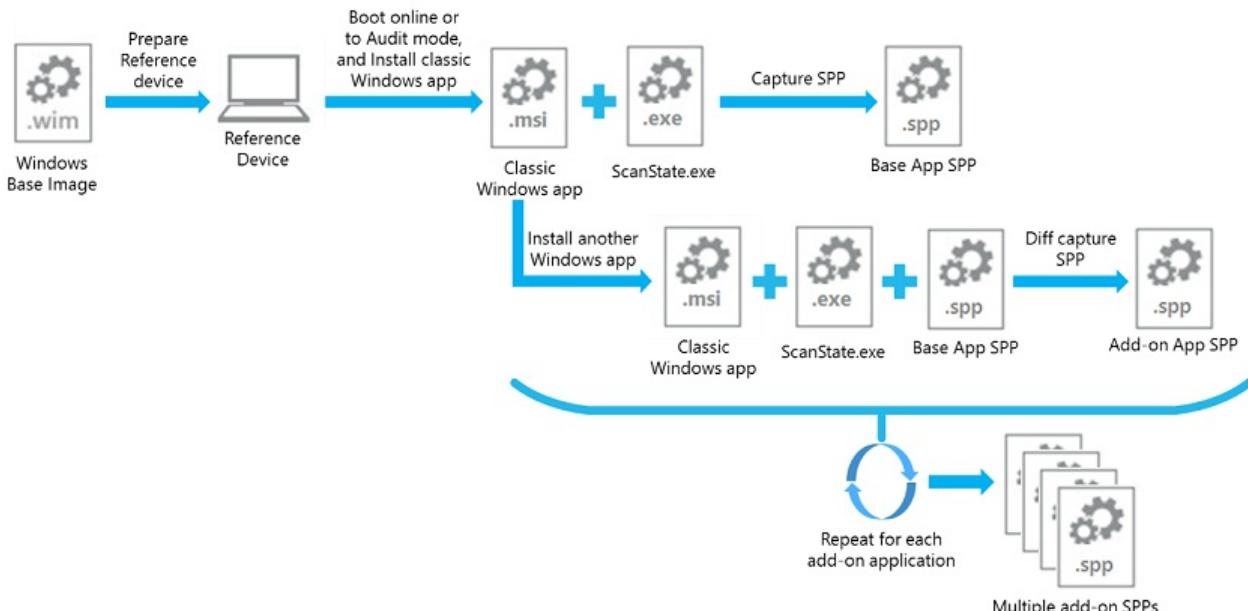
A Microsoft partner can use diff capture support to generate supplemental (or add-on) siloed provisioning packages that are relevant to a previously captured parent siloed provisioning package. The siloed provisioning packages can then be installed on devices at factory floor, with parent package first followed by combinations of supplemental packages in the customized order.

For example, you could capture the antivirus program base siloed provisioning package, and then diff capture an antivirus program patch (MSP) siloed provisioning packages using the base package as the parent. At the factory floor, the antivirus program base package and a selection of patch packages, specified in the desired order, can then be installed on a specific model device.

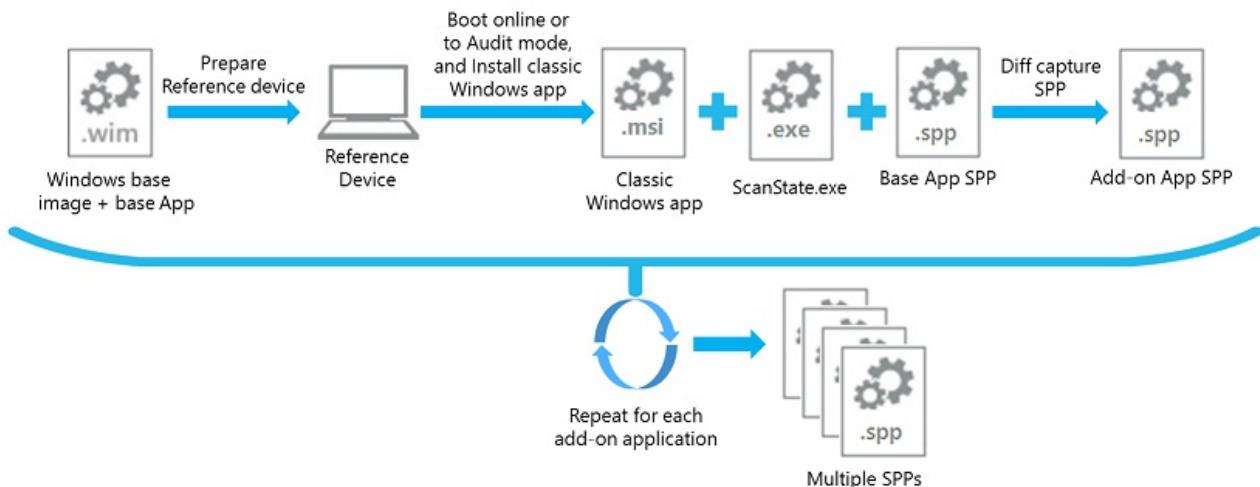
### Capturing Base App SPP



### Capturing Add-on App SPPs – Option 1



## Capturing Add-on App SPPs – Option 2



1. Clean install Windows 10, version 1607 on a reference device.
2. At the desktop, install the antivirus application.
3. Sysprep generalize and capture the OS image from the reference device.
4. Run ScanState.exe to capture your antivirus base siloed provisioning package.
5. Install antivirus software program patches.
6. Run ScanState.exe to diff capture the antivirus software program patches in a siloed provisioning package using the antivirus base package.
7. Either continue using the diff switch with the already captured base and program patches packages to capture another antivirus program patch siloed provisioning package:
  - a. Install additional antivirus software program patches.
  - b. Run ScanState.exe to diff capture the additional program patches siloed provisioning package using antivirus base package and the first program patches SPPs.
8. Or wipe and start clean again on the reference device to diff capture another antivirus software siloed provisioning package:
  - a. Wipe and clean install the reference device using the OS image captured in step 3.
  - b. At the desktop, install antivirus software.
  - c. Run ScanState.exe to diff capture antivirus software program patches siloed provisioning package using the antivirus base package captured in step 4.
9. Repeat either step 7 or 8 to capture any additional antivirus program patch siloed provisioning packages.

Alternatively, the siloed provisioning packages can be captured using VM instead of a physical device. When using a VM:

1. Create a VM and boot it online using a Windows 10, version 1607 VHD/VHDX image.
2. At the desktop, install the antivirus application.
3. Create a checkpoint of the OS installation with the antivirus software on the VM.
4. Run ScanState.exe to capture your antivirus base siloed provisioning package.
5. Install antivirus software program patches.
6. Run ScanState.exe to diff capture the antivirus software program patches in a siloed provisioning package using the antivirus base package.
7. Either continue using diff switch with the already captured base and language packages to capture another Office 2016 language siloed provisioning package:
  - a. Install additional antivirus software program patches.
  - b. Run ScanState.exe to diff capture the additional program patches siloed provisioning package using antivirus base package and the first program patches SPPs.

8. Or restart the VM to diff capture another antivirus software program patch siloed provisioning package:
  - a. Revert the VM to the checkpoint generated in step 3.
  - b. At the desktop, install antivirus software.
  - c. Run ScanState.exe to diff capture antivirus software program patches siloed provisioning package using the antivirus base package captured in step 4.
9. Repeat either step 7 or 8 to capture any additional antivirus program patch siloed provisioning packages.

Siloed provisioning packages can also capture applications with dependencies. For example, to capture multiple apps that depend on .NET Framework:

1. Create a VM and boot it online using a Window 10, version 1607 VHD/VHDX image.
2. Install .NET Framework.
3. Create a checkpoint of the OS installation with .NET Framework.
4. Capture a base .spp, for example, DotNet.spp.
5. Install App1, capture it as App1.spp, using /diff:DotNet.spp.
6. Revert the VM to the checkpoint created in Step 3.
7. Install App2, capture it as App2.spp, using /diff:DotNet.spp.

To preserve the dependency, apply the packages in this order:

- DotNet.spp, App1.spp, App2.spp
- or-
- DotNet.spp, App2.spp, App1.spp

The important point is DotNet.spp must be applied first.

### **Capturing an Application with associated device drivers**

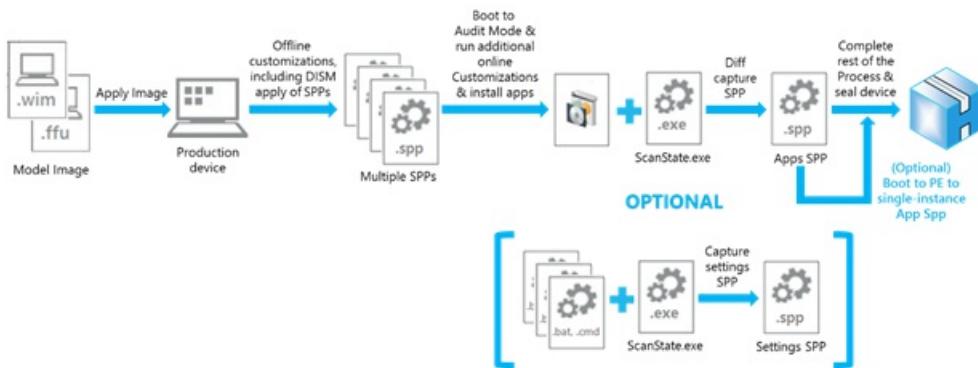
A Microsoft partner can capture a siloed provisioning package of individual classic Windows applications that have hardware drivers associated with them while in the imaging lab, and then install any combination of siloed provisioning packages in a customized order at factory floor. For example, a partner could capture a siloed provisioning package for Microsoft Mouse and Keyboard Center that contains both application and driver files.

1. Clean install Windows 10, version 1607 on a reference device.
2. At the desktop, install Microsoft Mouse and Keyboard Center.
3. Run ScanState.exe to capture Mouse and Keyboard Center siloed provisioning package, using both the /apps and /drivers switches.
4. Wipe and clean install the reference device
5. On the target device, boot to Windows PE, and apply the Windows 10, version 1607, desktop image.
6. While in Windows PE, run DISM /Apply-SiloedPackage command with the Microsoft Mouse and Keyboard Center package to apply the application and driver files in the packages onto the applied desktop image.
7. Complete the rest of the offline customization tasks.
8. Go through first boot and run through specialize to get to Audit mode.
9. Complete the online customization/configuration tasks.
10. (Optional) While in Audit mode, run ScanState to capture only the system settings into siloed provisioning package and place it in the recovery folder.
11. Complete the rest of the factory floor tasks and shutdown/reseal the product.

### **Capturing and applying applications for BTO model**

In the BTO model, last minute customizations on the factory floor could include installing additional classic Windows applications on to a customized image. If any classic Windows applications were not captured in siloed provisioning packages in the imaging lab, the factory floor process will then include the tasks shown in the

following diagram:



1. On the target device, boot to Windows PE, and apply Windows 10, version 1607 desktop image.
2. While in Windows PE, run DISM /Apply-SiloedPackage command specifying all the siloed provisioning packages to apply the application files in the packages onto the applied desktop image.
3. Complete the rest of the offline customization tasks.
4. Go through first boot and run through specialize to get to Audit mode.
5. Online install of classic Windows applications in Audit mode.
6. Complete the online customization/configuration tasks.
7. Run ScanState.exe to diff capture the applications installed in step 5 into a single siloed provisioning package, using the siloed provisioning packages for the applications already installed in the base model image as a reference.
8. (Optional) Run ScanState to capture only the system settings into siloed provisioning package and place it in the recovery folder.
9. (Optional) Boot the device to Windows PE, and run the DISM command to single-instance the application files in the siloed provisioning package captured in step 7.
10. Complete the rest of the factory floor tasks and shutdown/seal the product.

**Preferred process guidelines for BTO model:** As described in the preceding steps, the diff capture support provides flexibility to allow installing a classic Windows applications at the factory floor as last minute customizations. However, the diff capture operation may take some time to complete, depending on the number and the size of the siloed provisioning packages it needs to diff against. There is also overhead cost for the other steps in the process. Therefore, the preferred guideline for installing a classic Windows application in the BTO model is to incur the onetime cost of capturing the siloed provisioning packages for these applications in the imaging lab. They can then be applied on the factory floor as needed for last-minute customizations.

## Related topics

[WinPE: Create USB bootable drive](#)

[Lab 9: Make changes from Windows \(audit mode\)](#)

[Lab 10: Add desktop applications and settings with siloed provisioning packages \(SPPs\)](#)

# Create a provisioning package with Windows desktop applications

1/18/2019 • 2 minutes to read

Here's how to add Windows desktop applications and other data by using audit mode. You'll recapture these Windows desktop applications and data into a provisioning package by using the ScanState tool. As new builds of Windows are released, and as you prepare for different markets, you can mix and match the Windows images and provisioning packages, rather than rebuilding and customizing the images each time.

Once you've captured the provisioning package, you can add it to your image by using Windows ICD.

The recovery tools also use this provisioning package. When your users refresh the device (often used in case of device failure) or reset the device (often used to clean a device for a new user), the device keeps their installed Windows updates, plus the updates in this provisioning package.

## Step 1: Prepare a copy of ScanState

1. On your technician PC, plug in another USB key or drive.
2. In File Explorer, create a new folder on the USB key, for example: **D:\ScanState x64**.
3. Copy the files from "**C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\User State Migration Tool\amd64**" into **D:\ScanState x64**. You don't need to copy the subfolders.
4. Copy the files from "**C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Setup\amd64\Sources**" into **D:\ScanState x64**. There will be duplicate files, it's OK to skip copying these files. You don't need to copy the subfolders.

## Step 2: Install a Windows desktop application in audit mode

Use this method to install Windows desktop applications and any drivers that require installation (as opposed to .inf-style drivers.)

1. On the reference device, install the image that was created in Lab 1a. If the image is already installed, start the reference device. Either the **Languages** or the **Hi there** screen appears.
2. Press **Ctrl+Shift+F3** to enter Audit mode. The device reboots to the desktop, and the System Preparation Tool (Sysprep) appears. You can close Sysprep.
3. Ensure that your customizations from [Lab 1a](#) are available. To do this, in **Settings** under **System > About**, you should see the technical support info that you entered earlier appear (company name, support phone number, and support website).
4. Install a Windows desktop application application. For example, to install Office 2013, put in a USB key with the Office installation program, open File Explorer and navigate to [oemsetup.en-us.com](#). To learn more, download the Office OPK Update image from the Office OPK Connect site.

## Step 3: Save your updates to a provisioning package

### Capture your updates into a provisioning package

First, plug the USB key with ScanState into the reference device.

- **If you'd like to keep a copy of this provisioning package and deploy it to other devices**, save the file

to a USB drive.

Capture the changes into the provisioning package, and save it on the USB key.

```
D:\ScanState_x64\scanstate.exe /apps /ppkg D:\Provisioning\ClassicApps.ppkg /o /c /v:13  
/l:D:\ScanState.log
```

where *D* is the letter of the drive with ScanState.

- **For build-to-order devices**, you can wrap up these changes and prepare the device for immediate delivery. Capture the changes to provisioning package, and save it as C:\Recovery\Customizations\usmt.ppkg:

```
D:\ScanState_x64\scanstate.exe /apps /ppkg C:\Recovery\Customizations\usmt.ppkg /o /c /v:13  
/l:D:\ScanState.log
```

## Step 4: Prepare the device for an end user

- **For build-to-order devices**, prepare the device for the end user: Right-click **Start**, select **Command Prompt (Admin)**, and run the following command:

```
C:\Windows\System32\Sysprep\sysprep /oobe /shutdown
```

The Sysprep tool reseals the device. This process can take several minutes. After the process completes, the device shuts down automatically. You can now send the device to the customer.

# Sideload Apps with DISM

1/18/2019 • 15 minutes to read

You can sideload line-of-business (LOB) Windows apps to a Windows 10 by using PowerShell or Deployment Image Servicing and Management (DISM). Windows apps include:

- Universal Windows apps devices: Windows apps built upon the Universal Windows app platform, targeting the universal device family.
- Universal Windows 8 apps: Windows apps that target Windows 8.x.

Typically, Windows apps are available only through the Microsoft Store. You can submit LOB Windows apps to the Microsoft Store and make them available outside of your enterprise. However, you can also develop Windows apps for use only within your enterprise and add them to Windows devices you manage through a process we call *sideloading*. Sideloaded apps do not have to be certified by or installed through the Microsoft Store.

Here's what you'll need to know in order to sideload apps:

HOW TO?	DESCRIPTION
<a href="#">Understand Sideload Concepts</a>	Introduces some basic concepts you'll need to know about sideloading apps.
<a href="#">Configure PCs for Sideload Requirements</a>	Shows the requirements to be met in order to sideload apps on devices running different Windows Editions. Includes how to use Group Policy to configure your enterprise PCs for sideloading apps.
<a href="#">Configure PCs for Developing Microsoft Store Apps</a>	Shows you how to configure your PC to have a developer license that does not expire. The PC can be used to develop Microsoft Store apps or enterprise apps that will be added to your enterprise devices.
<a href="#">Add Apps</a>	Shows you how to sideload apps that you develop.
<a href="#">Add Multiple Languages for Apps</a>	Shows you how to prepare a multi-lingual image, sign-in to the image, install any desired app resource packs (including language) and then use Copy Profile to capture the image.
<a href="#">Inventory Apps</a>	Shows you how to list the LOB apps installed on the devices in your enterprise or in an offline Windows image.
<a href="#">Remove Apps</a>	Shows you how to remove individual instances of an app or remove the provisioning setting of an app.

## Understand Sideload Concepts

Windows apps differ from Windows desktop applications in their design and in the way users can interact with

them. To learn more about Windows apps , see [what is a Microsoft Store App?](#).

You cannot sideload an app that has been downloaded from the Microsoft Store. To install Windows apps that are not part of your business line, you must use the Microsoft Store. To learn more, see [Managing Client Access to the Microsoft Store](#).

LOB Windows apps that are not signed by the Microsoft Store can be sideloaded or added to a PC in the enterprise through scripts at runtime on a per-user basis. They can also be provisioned in an image by the enterprise so that the app is registered to each new user profile that's created on the PC. The requirements to sideload the app per-user or in the image are the same, but the Windows PowerShell cmdlets you use to add, get, and remove the apps are different. This topic provides steps for both methods.

Before you can sideload LOB Windows apps that are not signed by the Microsoft Store, you will need to configure the PC, see [Configure PCs for Sideload Requirements](#).

### **When You're Developing LOB Windows apps for Your Enterprise**

LOB Windows apps that are not signed by the Microsoft Store must be cryptographically signed. The apps can only be installed on a computer that trusts the signing certificate.

For more information about how to sign an app and using certificates, see [App Packaging Tools](#).

However, you can use a developer license to add apps that are in development to your PC. For more information about testing apps that are in development, see [Get a Developer License](#).

You can use Group Policy to configure your domain-joined PCs to have a developer license that does not expire to support app development. Once the PCs are configured, you won't need to connect to the Internet to obtain or renew a license. See [Configure PCs for Developing Microsoft Store Apps](#) for more information.

## **Configure PCs for Sideload Requirements**

Until the device meets all of the sideloading requirements, app tiles on the Start menu will show an "X" in the bottom-right corner to indicate that a problem is preventing the app from running.

In some cases, part of those requirements includes using a sideloading product key. This key provides use rights needed to deploy Windows 8, or Windows 8.1 apps directly to devices without having to install them through the public Microsoft Store.

Before you can add and run sideloaded LOB Windows apps that are not signed by the Microsoft Store you must configure your device based on the following conditions:

1. For those devices that are joined to a workgroup, you must:

- Activate the sideloading product key on the device.
- And enable the **Allow all trusted applications to install** Group Policy setting. See [Use Group Policy to configure your enterprise PCs for sideloading apps](#).

This applies to:

- Windows 10 Enterprise
- Windows 8.1 Enterprise
- Windows 8 Enterprise
- Windows Embedded 8.1 Industry Enterprise
- Windows 8.1 Pro Update

2. For those devices that will be joined to an Active Directory domain, you must:

- Join the device to an Active Directory domain.

- And enable the **Allow all trusted applications to install** Group Policy setting. See [Use Group Policy to configure your enterprise PCs for sideloading apps](#).

This applies to:

- Windows 10 Enterprise
- Windows 8.1 Enterprise
- Windows 8 Enterprise
- Windows Embedded 8.1 Industry Enterprise
- Windows 8.1 Pro Update
- Windows Server 2016 Technical Preview
- Windows Server 2012 R2 Update
- Windows Server 2012

3. For those devices that will require a sideloading product key, whether the device is domain-joined or a member of a workgroup, you must:

- Activate the sideloading product key on the device.
- And enable the **Allow all trusted applications to install** Group Policy setting. See [Use Group Policy to configure your enterprise PCs for sideloading apps](#).

This applies to:

- Windows 10 Pro
- Windows RT 8.1
- Windows 8.1 Pro
- Windows RT
- Windows 8 Pro
- Windows Embedded 8.1 Industry Pro

4. For certain Windows Embedded 8 Industry devices, you no longer need a sideloading product key whether the device is domain-joined or a member of a workgroup. In this case, you must:

- Enable the **Allow all trusted applications to install** Group Policy setting on the device.

For more information about sideloading on Windows Embedded 8 Industry, see [Enterprise guide to installing Universal Windows 8 apps on Windows Embedded 8 Industry](#).

This applies to:

- Windows Embedded 8.1 Industry Pro Update
- Windows Embedded 8.1 Industry Enterprise Update

## Use Group Policy to configure your enterprise PCs for sideloading apps

1. Open the Group Policy Management Editor for a domain—based Group Policy Object (GPO) to which you will be applying the group policy setting, as specified below, to your selected PCs.

**Note** The steps provided in this procedure assume you understand the basics of Group Policy design and operations. To administer domain—based Group Policy on a Windows 8.1 PC, you will need to install the Group Policy Management Console which is installed with the [Remote Server Administration Tools for Windows 8.1](#). For more information about Group Policy, see [Group Policy for Beginners](#) and the [Group Policy Techcenter](#).

2. Click to expand **Computer Configuration**, **Administrative Templates**, **Windows Components**, and then **App Package Deployment**.
3. Double-click the **Allow all trusted apps to install** setting.

4. In the **Allow all trusted apps to install** window, click **Enabled** and then click **OK**.

Setting the Group Policy to allow trusted applications updates the following registry setting:

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Appx\AllowAllTrustedApps = 1**

#### To activate a sideloading product key

1. Open a command prompt with administrator privileges and type the following to add the sideloading product key:

```
slmgr /ipk <sideloading product key>
```

Where *<sideloading product key>* is the 25 digit key to enable sideloading on the computer.

2. Activate the sideloading key by typing:

```
slmgr /ato ec67814b-30e6-4a50-bf7b-d55daf729d1e
```

**Note** The activation GUID is not the same as the sideloading product key. The activation GUID will always be ec67814b-30e6-4a50-bf7b-d55daf729d1e.

For more information about sideloading product keys, see the [Windows 8 Licensing Guide](#).

## Configure PCs for Developing Windows apps

You can configure your PCs to have a developer license that does not expire. Once the PCs are configured, you won't need to connect to the Internet to obtain or renew a license. Your computer must be a member of a domain and be running either of the following operating systems:

- Windows 10 Enterprise
- Windows 8.1 Enterprise
- Windows 8 Pro

**Note** To enable sideloading on Windows 8 Pro device, you must use a sideloading product activation key. For more information see, [Configure PCs for Sideload Requirements](#)

#### To configure your enterprise PCs with a developer license

1. Open the Group Policy Management Editor for a domain—based Group Policy Object (GPO) to which you will be applying the group policy settings, as specified below, to your selected PCs.

**Note** The steps provided in this procedure assume you understand the basics of Group Policy design and operations. To administer domain—based Group Policy on a Windows 8.1 PC, you will need to install the Group Policy Management Console which is installed with the [Remote Server Administration Tools for Windows 8.1](#). For more information about Group Policy, see [Group Policy for Beginners](#) and the [Group Policy Techcenter](#).

2. Click to expand **Computer Configuration**, **Administrative Templates**, **Windows Components**, and then **App Package Deployment**.
3. Double-click the **Allow development of Windows apps without installing a developer license** setting.
4. In the **Allow development of Windows apps without installing a developer license** window, click **Enabled** and then click **OK**.
5. Double-click the **Allow all trusted apps to install** setting.

6. In the **Allow all trusted apps to install** window, click **Enabled** and then click **OK**.

Setting the Group Policy to allow development of Windows apps without installing a developer license updates the following registry setting:

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Appx\AllowDevelopmentWithoutDeveloperLicense = 1**

Setting the Group Policy to allow trusted applications updates the following registry setting:

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Appx\AllowAllTrustedApps = 1**

## Add Apps

There are two ways to add apps. A user can add an app package, which will make the app available to just that user. Or the app can be installed in the Windows image, which will make the app available to every user of the Windows image at first logon or at the next logon, if the user account is already created. This second case is referred to as provisioning an app package.

### Add an App Package

You can install an app package (.appx or .appxbundle) on a per-user basis by using the *add-appxpackage* PowerShell cmdlet. There is no limit to the number of LOB apps you can add for each user.

### Add a LOB app to a user account

- At the Windows PowerShell prompt on a Windows 8 or Windows Server 2012 computer, add an .appx (or .appxbundle) file package. Include any required dependency app packages when you add the app. For example, type:

```
add-appxpackage C:\app1.appx -DependencyPath C:\winjs.appx
```

For more information, see [App Installation Cmdlets in Windows PowerShell](#).

### Add a provisioned LOB app to a Windows image

Apps that are installed in the Windows image are called *provisioned* apps. Provisioned apps are staged in the image and are scheduled to be installed for every user of the Windows image at first logon or at the next logon, if the user account is already created.

You can add these apps to a Windows image when you boot into audit mode before you deploy the image by using the DISM app provisioning commands. For more information about audit mode, see [Audit Mode Overview](#).

Provisioned apps are specific to the PC and will not roam with the user. You can only install 24 provisioned apps in an image.

On a Windows image that has already been deployed, you should instead use the Add-AppxPackage cmdlet in PowerShell. If you do use the DISM app provisioning commands on a deployed Windows image with active users, you should log all users off of the image, so that you are the only user logged on, before you run the command.

**Note** On Windows 8, to update a provisioned app, you must first remove the provisioned app and then deploy the new version of the app. The update will then be applied the next time each user logs in.

On Windows 8.1 and newer, you no longer need to remove the provisioned app prior to deploying the new version of the provisioned app.

### Add a provisioned LOB app to a Windows image

- Use the Deployment Image Servicing and Management (DISM) command-line tool or PowerShell cmdlets

to add the LOB app without a Microsoft Store license. For example, at an elevated command prompt, type:

```
DISM /Online /Add-ProvisionedAppxPackage /PackagePath:C:\App1.appx /SkipLicense
```

Or, at a Windows PowerShell prompt, type:

```
Add-AppxProvisionedPackage -Online -FolderPath C:\Appx -SkipLicense
```

For more information, see [DISM App Package \(.appx or .appxbundle\) Servicing Command-Line Options](#) or [DISM Cmdlets](#). For information about DISM supported platforms, see [DISM Supported Platforms](#).

**Note** The computer does not have to be joined to a domain or have an activated sideloading product key before you install provisioned LOB apps. However, the apps will not run until the computer meets this sideloading requirement. For more information, see [Customize the Start Screen](#).

### **Update a provisioned LOB app once it is has been added to a Windows image**

On Windows 8, to update a provisioned app, you must first remove the provisioned app and then deploy the new version of the app. The update will then be applied the next time each user logs in.

On Windows 8.1 and newer, to update a provisioned app, you will need to update the app for each user that has signed on to the Windows image provisioned with the app:

### **Update a provisioned LOB app to a Windows image**

1. Use the PowerShell to update the LOB app without a Microsoft Store license. This must be done for each user that has signed in to the PC running the Windows image. For example, if you have installed the original version of the app, 1.0.0.0, that now needs to be updated to version 1.0.0.1, then at a PowerShell session, type:

```
Add-AppxPackage -Path App1_1.0.0.2 -DependencyPath C:\appx\WinJS.appx
```

Where `C:\appx\WinJS.appx` is the path to the dependency package.

2. Once you have updated your app, you can verify the version of the updated app. From a PowerShell session, type:

```
Get-AppxPackage | Out-GridView
```

## Add Multiple Languages for Apps

To prepare a multi-lingual image, sign-in to the image, install any desired app resource packs (including language) and then use Copy Profile to capture the image.

### **Preparing a multi-lingual image for an app**

1. Create an unattend.xml with the following contents to c:\unattend.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<unattend xmlns="urn:schemas-microsoft-com:unattend">
    <settings pass="specialize">
        <component name="Microsoft-Windows-Shell-Setup" processorArchitecture="x86"
publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS"
xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <CopyProfile>true</CopyProfile>
            <RegisteredOrganization />
            <RegisteredOwner />
        </component>
    </settings>
    <cpi:offlineImage cpi:source="catalog:d:/desktop/x86 clgs/install_windows vista ultimate.clg"
xmlns:cpi="urn:schemas-microsoft-com:cpi" />
</unattend>
```

**Note** See [Change the language used in apps](#) for information about setting the language and installing updates from the Microsoft Store.

2. Sign-in to a local administrator user account from OOBE on clean image.

**Important** When adding a specific language to a Windows app, you would also want to [Add Language Packs to Windows](#) for the same languages as you did for the Windows app.

3. Add the desired languages to the current user's language preference list.

- 4. Install app updates using a Microsoft Store account (MSA account)**

- a. Sign-in to the Microsoft Store with an MSA account.

**Note** Microsoft Store only. Don't convert the local account to MSA.

If you do not have an MSA account, you can update apps without a Microsoft Store account.

- b. Check for updates and install new language resource packs.
  - c. Sign out of the Microsoft Store and remove the MSA account.

5. Open an elevated command prompt and type:

```
Sysprep.exe /generalize /oobe /reboot /unattend:C:\unattend.xml
```

Then press enter.

6. You should see the PC boot to OOBE. Any languages that you have added prior to Copy Profile should be present at this point.

#### **Install app updates without using a Microsoft Store account (MSA account)**

1. After the PC has finished installing, open an elevated command prompt.
2. Type **Start ms-windows-store:Updates**
3. You will see the Microsoft Store Updates page. You should see the pending updates displayed.
4. Tap **Install** to install the updates.

## Inventory Apps

You can list the LOB apps installed in on offline or online Windows image and get additional information about the packages.

## List LOB Apps per user account

1. You can get a list of the Windows apps installed for a specific user account on the computer. You must open PowerShell with administrator privileges to list the packages for a user other than the current user. For example, at the PowerShell prompt, type:

```
Get-AppxPackage -AllUsers
```

2. You can get a list of packages installed for a specific user. You must open PowerShell with administrator privileges to list the packages for a user other than the current user. For example, at the PowerShell prompt, type:

```
Get-AppxPackage -Name Package1 -User domain\username
```

3. You can also get the manifest of an app package (.appx) which includes information such as the package ID. For example, at the PowerShell prompt, type:

```
Get-AppxPackageManifest -Package Package1
```

4. You can use the pipeline to get the manifest for an app package (.appx) if you don't know the full name of the package. For example, at the PowerShell prompt, type:

```
(Get-AppxPackage -Name "*WinJS*" | Get-AppxPackageManifest).package.applications.application.id
```

## List LOB apps that are provisioned in a Windows image

- You can get a list of the packages that are provisioned in a Windows image that will be installed for each new user by using Dism.exe or PowerShell. For example, at a PowerShell prompt, type:

```
Get-AppxProvisionedPackage -Path c:\offline
```

Or, at a command prompt, type:

```
DISM.exe /Image:C:\test\offline /Get-ProvisionedAppxPackages
```

For more information, see [Take Inventory of an Image or Component Using DISM](#).

## Remove Apps

You can remove individual instances of an app, or remove the provisioning setting of an app.

### Remove LOB apps per user account

- You can remove a single app for the current user only. For example, at a command prompt, type:

```
Remove-AppxPackage Package1
```

### Remove provisioned LOB apps in a Windows image

- When you remove a provisioned app, the app will not be installed for new user accounts. For the currently logged in user and other user accounts that are active on the computer, the app will not be removed from

those accounts. The app will need to be uninstalled for those existing apps.

For example, to remove a provisioned LOB app, MyAppxPkg, from a Windows image, at an elevated PowerShell prompt, type:

```
Remove-AppxProvisionedPackage -Online -PackageName MyAppxPkg
```

Or, at a command prompt, type:

```
DISM.exe /Online /Remove-ProvisionedAppxPackage /PackageName:microsoft.app1_1.0.0.0_neutral_en-us_ac4zc6fex2zjp
```

## Related topics

[App Installation Cmdlets in Windows PowerShell](#)

[DISM App Package \(.appx or .appxbundle\) Servicing Command-Line Options](#)

[App Packaging Tools](#)

[AppX Module Cmdlets](#)

[Change the language used in apps](#)

[DISM Cmdlets](#)

[DISM Supported Platforms](#)

[Enterprise guide to installing Universal Windows 8 apps on Windows Embedded 8 Industry](#)

[Get a Developer License](#)

[Group Policy for Beginners](#)

[Group Policy Techcenter](#)

[Customize the Start Screen](#)

[Managing Client Access to the Microsoft Store](#)

[Microsoft Volume Licensing](#)

[Remote Server Administration Tools for Windows 8.1](#)

[What is a Microsoft Store App?](#)

[Windows 8 Licensing Guide](#)

# Preinstall Apps Using DISM

1/24/2019 • 9 minutes to read

## NOTE

Interested in preinstalling Microsoft Store apps, but you aren't an OEM? For information about sideloading apps for organizations, see [Sideload Apps with DISM](#).

This topic covers how to preinstall apps so they are included as part of a Windows image.

## TIP

Hardware Support Apps (HSAs) are Microsoft Store Apps, and the following instructions apply.

## Work with app packages

Use DISM to offline provisioning an app into an image. You can use DISM from the Command Prompt, or the DISM cmdlets in Windows PowerShell.

In previous versions of Windows 10, preinstalled Universal apps and Microsoft Store apps had to be pinned to the Start menu. Windows would remove apps that were preinstalled but not pinned to the Start menu.

Starting with Windows 10, version 1803, apps can be preinstalled without being pinned to the Start Menu when you use `DISM /add-provisionedappxpackage` with the `/region` option. When you preinstall an app, you can choose to leave the app out of your LayoutModification.xml and the app will successfully install without appearing as a Start Menu tile. When a list of regions is NOT specified, the app will be provisioned only if it is pinned to start layout.

### Extract the package files

1. Browse to the folder where you saved the app packages that you downloaded from the Partner Dashboard.
2. Right-click each .zip folder containing your app package files. Click **Extract All** and select a location to save the package file folders.

The folder contains the all of the unpacked files for the package, including a main package, any dependency packages, and the license file.

## IMPORTANT

Don't modify the folder once you have extracted the package files. If you change, add, or remove any files in the folder, the app will fail either during installation or launch. Even browsing the folder may cause problems.

You'll need to use the license file from the package files to test your provisioned image. Creating your own custom data file will not allow you to accurately test an app preinstalled by an OEM.

For offline provisioning of an app into an image, you can use either the Dism.exe tool or the DISM cmdlets in Windows PowerShell to add an app from a folder of unpacked files.

### Preinstall a Microsoft Store-signed app with DISM

1. Open the Command Prompt as administrator.

2. Mount a Windows image for that you want to service:

```
Dism /Mount-Image /ImageFile:c:\images\myimage.wim /Index:1 /mountdir:c:\test\offline
```

3. Add the app to the mounted image. Use the `/PackagePath` and `/DependencyPackagePath` options.

- `Packagepath` is the path to the .appx bundle or package file
- `DependencyPackagePath` is the path for specifying each dependency package. You can have more than one dependency per command.
- **New in Windows 10, version 1803:** Use the `Region` option when adding apps. `Region` allows you to add an app without having to pin the app to the Start Menu.

```
Dism /Image:c:\test\offline /Add-ProvisionedAppxPackage /PackagePath:c:\downloads\package.appxbundle  
/DependencyPackagePath:c:\downloads\dependency1.appx  
/DependencyPackagePath:c:\downloads\dependency2.appx /LicensePath=c:\downloads\package_License1.xml  
/region=all"
```

See [DISM app package servicing command-line options](#) for information about working with app packages, including the new `/region` option..

4. Pin the app to the Start Menu.

- If you didn't specify `/region` when preinstalling the app, pin the app to the Start Menu with [LayoutModification.xml](#).

or

- If you did specify `/region` when preinstalling the app, and you want to pin the app to the Start Menu for specific regions, use the `RequiredStartGroups Region="region1|region2"` element in [LayoutModification.xml](#) to specify the regions where you want the app to appear.

5. Save changes and unmount the image. At the command prompt, type:

```
Dism /Unmount-Image /mountdir:c:\test\offline /commit
```

**NOTE**

Microsoft Store apps don't run in audit mode. To test your deployment, run Windows and create a new user profile. For more information about audit mode, see [Audit mode overview](#).

**Important** If you are preinstalling a mobile broadband device app, you must insert the SIM card in the PC before you run the specialize phase of Sysprep. For more information about preinstalling a mobile broadband device app, see [Preinstall the Necessary Components for a Mobile Broadband Application Experience](#).

## Update or remove packages

You can remove a preinstalled app, including the license and custom data files, from a Windows image by using the DISM.exe tool or the DISM cmdlets in Windows PowerShell. You should remove the old version of the app before installing a new one.

### Remove a preinstalled app by using DISM

1. Open the Deployment Tools Command Prompt, installed with the Windows ADK, with administrator privileges. From the Start screen, type **Deployment and Imaging Tools Environment**, right-click the

icon, and select **Run as Administrator**.

2. Mount the offline image for servicing. At the command prompt, type:

```
Dism /Mount-Image /ImageFile:c:\images\myimage.wim /Index:1 /mountdir:c:\test\offline
```

3. Find the full package name of the app that you want to remove. At the command prompt, type:

```
Dism /Image:C:\test\offline /Get-ProvisionedAppxPackages
```

4. Remove the app from the mounted image. For example, at the command prompt, type:

```
Dism /Image:c:\test\offline /Remove-ProvisionedAppxPackage  
/PackageName:microsoft.devx.appx.app1_1.0.0.0_neutral_en-us_ac4zc6fex2zjp
```

5. If you want to update the app, you can preinstall the updated version of the Microsoft Store-signed app. At a command prompt, type:

```
Dism /Image:c:\test\offline /Add-ProvisionedAppxPackage/FolderPath:c:\downloads\appxpackage
```

6. Save changes and unmount the image. At the command prompt, type:

```
Dism /Unmount-Image /mountdir:c:\test\offline /commit
```

## Use custom data files

Apps that are preinstalled on a PC can access custom data specific to the installation. This custom data is added to the app during preinstallation and becomes available at runtime. Custom data enables developers to customize an app's features and functionality, including providing reporting capabilities.

### Add a custom data file to a Windows image

You must specify the custom data file when you preinstall the app by using the DISM tool and through Windows PowerShell using the **Add-AppxProvisionedPackage** cmdlet. The following command shows how to do this using the DISM tool:

```
Dism /Image:C:\test\offline /Add-ProvisionedAppxPackage / FolderPath:f:\Apps\Fabrikam_KnowMyPC  
/CustomDataPath:f:\Contoso_Promotion.xml
```

If a custom data file already exists in the data store for an app—for example, if the package has already been added to the image—the existing file is overwritten. If the installation fails, the file isn't restored.

**Note** You can release updates to an app through the Microsoft Store without losing the custom data file. However, if a user deletes the app, the custom data file is no longer available, even if the user reinstalls the app.

### Test custom data for preinstalled apps

Apps that are preinstalled on a PC can access custom data specific to the installation. This custom data is added to the app during preinstallation and becomes available to the app at runtime. Custom data enables developers to customize an app's features and functionality, including providing reporting capabilities.

The Custom.data file appears at the app's installed location. The name Custom.data is hard-coded and can't be modified. Your app can check for the existence of this file to determine if the app was preinstalled on the PC. Here's an example of how to access the Custom.data file.

```

var outputDiv = document.getElementById("CustomData");
Windows.ApplicationModel.Package.current.installedLocation.getFileAsync
  ("microsoft.system.package.metadata\\Custom.data").then(function (file) {
    // Read the file
    Windows.Storage.FileIO.readTextAsync(file).done(function (fileContent) {
      outputDiv.innerHTML =
        "App is preinstalled. CustomData contains:<br /><br />" +
        fileContent;
    },
    function (error) {
      outputDiv.innerText = "Error reading CustomData " + error;
    });
  },
  function (error) {
    outputDiv.innerText = "CustomData was not available. App not preinstalled";
  });

```

Your Custom.data file can include any content and be in any format your app requires. The preinstallation process simply makes it available to your app. Developers can supply the data file to the preinstallation partner, or you can agree to a format that enables the partner to generate the content.

### Test your custom data

When you're building and debugging your app in Microsoft Visual Studio, you can't access the Custom.data file from the app's installed location because the app isn't preinstalled yet. You can simulate using your Custom.data file by putting a test Custom.data file in the app itself, and then loading and testing the app local file. To do this, modify the code sample from:

```
("microsoft.system.package.metadata\\Custom.data").then(function (file) {
```

to:

```
("Custom.data").then(function (file) {
```

After you have verified your file format and content, you can change the location of the Custom.data file to the final location, as shown in the original example above.

### To test your Custom.data file

1. Open the Deployment Tools Command Prompt, installed with the Windows ADK, with administrator privileges. From the Start screen, type **Deployment and Imaging Tools Environment**, right-click the icon, and select **Run as Administrator**.
2. Add the application with the custom data file:

```
dism /online /Add-ProvisionedAppxPackage /PackagePath:.\CustomData_1.0.0.1_AnyCPU_Debug.appx
/CustomDataPath:.\Test.txt /SkipLicense
```

Where `/PackagePath:.\CustomData_1.0.0.1_AnyCPU_Debug.appx` points to your local app test package, and where `/CustomDataPath:.\Test.txt` points to your Custom.data file. Be aware that the file name you provide here isn't used after the data is installed in your app.

The app now has a tile on the **Start** screen of the PC used to test the app. The app should be able to access the Custom.data file. If additional debugging is needed, attach a debugger after starting the app from the **Start** screen.

**Note** You might be required to sign out and sign in again to see the app on the **Start** screen.

3. After you're done testing your app, you must remove the preinstalled package to continue using your Dev environment. To remove the preinstalled package using Windows PowerShell, you can use the **Get-AppxPackage** cmdlet to provide the full app package name through the pipeline to the **Remove-ProvisionedAppxPackage** cmdlet:

```
Get-AppxPackage *CustomData* | Remove-ProvisionedAppxPackage
```

Where `*CustomData*` is the known part of your app's name

## Preinstall a Microsoft Store device app or mobile broadband app

You can preinstall the necessary components for a Microsoft Store device app or a mobile broadband app using the Deployment Image Servicing and Management (DISM) platform.

**Note** This article is intended for OEMs, who will be supporting a Microsoft Store device app or the mobile broadband app on their devices.

For each type of app, two things should be preinstalled to provide the correct Microsoft Store device app or mobile broadband app:

- Microsoft Store device app, preinstall:
  1. The device metadata package
  2. The app
- Microsoft Store mobile broadband app, preinstall:
  1. The service metadata package
  2. The app

**Important** Although metadata packages and the corresponding apps are parsed immediately after the OOBE process completes, a user might be able to launch the app before the metadata package is parsed. In this case, the user will see an access denied error. To avoid this, apply both the metadata package and the app to the system image.

## Preinstall the device metadata or service metadata package

### To preinstall a device metadata or service metadata package

1. If you are preinstalling a Microsoft Store device app then you should have acquired the device metadata package. If you are preinstalling a mobile broadband app then you should have acquired the service metadata package.

**Note** Device metadata packages and service metadata packages use the same file name extension (.devicemetadata-ms).

2. Copy the device metadata or service metadata package (devicemetadata-ms file) to your system image in the **%ProgramData%\Microsoft\Windows\DeviceMetadataStore** folder. You can do this in one of the following ways:

- Online before running Sysprep
- Offline after running Sysprep by using DISM. To do this:
  - a. Mount the offline image for servicing.

```
Dism /Mount-Image /ImageFile:C:\test\images\myimage.wim /index:1 /MountDir:C:\test\offline
```

- b. Copy the metadata package files to the device metadata store of the mounted image. For example, to copy the **0ECF2029-2C6A-41AE-9E0A-63FFC9EAD877.devicemetadata-ms** metadata package file to the device metadata store,

**ProgramData\Microsoft\Windows\DeviceMetadataStore:**

```
copy 0ECF2029-2C6A-41AE-9E0A-63FFC9EAD877.devicemetadata-ms  
C:\test\offline\ProgramData\Microsoft\Windows\DeviceMetadataStore
```

- c. Save the changes and unmount the image.

```
dism /Unmount-Image /mountdir: c:\test\offline /commit
```

For more info about offline image servicing, see [DISM Overview](#).

For more info about service metadata, see [Service metadata](#).

## Preinstall the Microsoft Store device app or mobile broadband app

### To preinstall the Microsoft Store device app or mobile broadband app

1. Mount the offline image for servicing.

```
dism /Mount-Image /ImageFile:C:\test\images\myimage.wim /index:1 /MountDir:C:\test\offline
```

2. Add the Microsoft Store device app or mobile broadband app to the image.

```
dism /Image:<mounted folder> /Add-ProvisionedAppxPackage /FolderPath:<appxpackage path>
```

3. Save the changes and unmount the image.

```
dism /Unmount-Image /mountdir: c:\test\offline /commit
```

# Export or Import Default Application Associations

1/18/2019 • 3 minutes to read

When building Windows 10 images, you can update the default programs associated with file name extensions or protocols.

To do this, create an app association file and import it into your images using DISM.

## Generate a default app associations XML File

Deploy your Windows image to a test computer and configure the programs that are included in your image. You can log into Windows and use Control Panel to select your default app associations. You can export the default app associations that you have configured to an XML file on a network share or removable media so that you can import them into the WIM or VHD file before you deploy it to your destination computers.

### Set default app associations

1. Install your Windows image to a test computer. For more information about how to apply a Windows image, see [Apply Images Using DISM](#).
2. Start the test computer and complete Windows Setup.
3. Open **Settings**. In **Find a setting**, search for and open **Default apps**.
4. Configure default apps by file name extension or by app. For example, to set an installed photo viewing app as the default program that is used to open all of the file types and protocols that it supports, click **Set defaults by app**, select the photo viewing app > **Manage**, and then select the file types and associations for the app.

### Export default app association settings

1. On your test computer, open a Command Prompt as administrator.
2. Export the default app association settings from the test computer to an .xml file on a network share or USB drive:

```
Dism /Online /Export-DefaultAppAssociations:"F:\AppAssociations.xml"
```

## Add or Remove Default Application Association Settings to a Windows Image

You can change the default app association settings in a WIM or VHD file before you deploy it to your destination computers. You can also add and remove default app association settings from an online image.

### Import default app association settings

1. On your technician computer, open a Command Prompt as administrator.
2. Mount a Windows image from a WIM or VHD file:

```
Dism /Mount-Image /ImageFile:C:\test\images\install.wim /Name:"Windows" /MountDir:C:\test\offline
```

3. Import the .xml file that has the default app association settings to the Windows image:

```
Dism.exe /Image:C:\test\offline /Import-DefaultAppAssociations:F:\AppAssociations.xml
```

### Review the default app association setting in an image

1. On your technician computer, open a Command Prompt administrator.
2. List the app associations that have been applied to the mounted image:

```
Dism.exe /Image:C:\test\offline /Get-DefaultAppAssociations
```

### Remove default app association settings

1. On your technician computer, open a Command Prompt as administrator.
2. Remove the custom default app association that have been added to the mounted image:

```
Dism.exe /Image:C:\test\offline /Remove-DefaultAppAssociations
```

### Unmount the Windows image

1. On your technician computer, open a Command Prompt as administrator.
2. Unmount the image, committing changes:

```
Dism /Unmount-Image /MountDir:C:\test\images\install.wim /commit
```

## Tips

- **Refresh this file with each major version.** Some recommended apps can manage more extensions with each new Windows 10 version available. It's a good practice to refresh your XML. For example, in Windows 10, version 1703, Microsoft Edge registers the epub extension. If you're using an XML file from Windows 10, version 1607, epub is missing. As a result, you will get an **An app default was reset** notification for epub.
- **Manually editing the file:** Don't just delete app associations you're not interested in. Otherwise on the first boot, the missing entries may trigger a flurry of notifications on the first boot saying "**An app default was reset**".

To prevent notifications, you can import multiple app association files. For example, you can import one file that includes all of the current Windows version's associations, and import your own separate file that includes a set of custom associations that you maintain one at a time.

1. From a new test computer, export a file that just includes the default file associations, for example, F:\1809-DefaultAppAssociations.xml.
2. Modify your app associations as before, and export the file again. Use a text/XML editor to modify just the associations you're interested in, for example, F:\InternetExplorerAssociations.xml.
3. Import both sets of associations into your images:

```
Dism.exe /Image:C:\test\offline /Import-DefaultAppAssociations:F:\1809-DefaultAppAssociations.xml  
Dism.exe /Image:C:\test\offline /Import-  
DefaultAppAssociations:F:\InternetExplorerAssociations.xml
```

# Microsoft .NET Framework 3.5 Deployment Considerations

1/24/2019 • 2 minutes to read

.NET Framework 3.5 is not included by default in Windows 10 or Windows Server 2016, but you can download and deploy it for application compatibility. This section describes these deployment options.

## In this section:

- [Deploy .NET Framework 3.5 by using Group Policy Feature on Demand setting](#)
- [Deploy .NET Framework 3.5 by using Deployment Image Servicing and Management \(DISM\)](#)
- [Enable .NET Framework 3.5 by using Windows PowerShell](#)
- [Enable .NET Framework 3.5 by using Control Panel and Windows Update \(Windows 8 only\)](#)
- [Enable .NET Framework 3.5 by using the Add Roles and Features Wizard](#)
- [.NET Framework 3.5 deployment errors and resolution steps](#)

## Introduction

Windows 10 or Windows Server 2016 include .NET Framework 4.6, which is an integral Windows component that supports building and running the next generation of applications and web services. The .NET Framework provides a subset of managed types that you can use to create Microsoft Store apps for Windows by using C# or Visual Basic. For more information, see [.NET Framework](#).

Only the metadata that is required to enable the .NET Framework 3.5 is contained in the default Windows image (`\sources\install.wim`). The actual binaries are not in the image. This feature state is known as *disabled with payload removed*.

You can get the .NET Framework 3.5 payload files from Windows Update or the installation media in the `\sources\sxs` folder. For more information, see [Installing the .NET Framework 3.5](#). After the .NET Framework 3.5 feature is enabled, the files are serviced just like other operating system files from Windows Update.

If you upgrade from Windows 7 (which includes .NET Framework 3.5.1 [by default](#)) to Windows 10, or from Windows Server 2008 R2 (which has .NET Framework 3.5.1 feature installed) to Windows Server 2016, .NET Framework 3.5 is automatically enabled.

## Related topics

[Windows Server Installation Options](#)

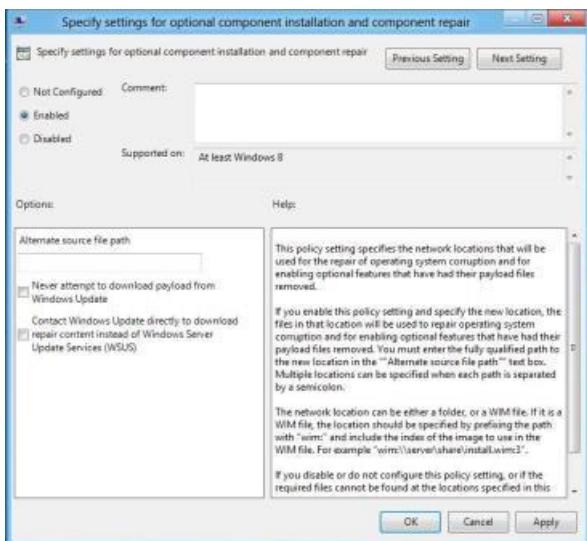
# Deploy .NET Framework 3.5 by using Group Policy Feature on Demand setting

1/18/2019 • 3 minutes to read

For environments that use Active Directory and Group Policy, the Feature on Demand (FoD) policy setting option provides the most flexibility for the installation of .NET Framework 3.5. This Group Policy setting specifies the network locations to use to enable optional features that have had their payload files removed, and for file data and registry repair operations from failed update installations. If you disable or do not configure this setting, or if the required files cannot be found at the locations that are specified in this policy setting, the files are downloaded from Windows Update (if this is allowed by the policy settings for the computer). The Group Policy setting **Specify settings for optional component installation and component repair** is located at **Computer Configuration\Administrative Templates\System** in Group Policy Editor.

## Requirements

- Active Directory Domain infrastructure that supports Windows 8 and Windows Server® 2012
- Access rights to configure Group Policy
- Target computers need network access and rights to use alternate sources, or an Internet connection to use Windows Update



**Figure 1 Group Policy Setting for Features on Demand and Feature Store Repair**

When this policy is enabled, a network location (for example, a file server) can be specified for both repair of the feature file store, and to enable features that have their payload removed. The **Alternate source file path** can point to a **\sources\sxs** folder or a Windows image (WIM) file using the WIM: prefix. The advantage of a WIM file is that it can be kept current with updates, and provide a current repair source and .NET Framework 3.5 binaries. The repair WIM can be different than the initial WIM file that is used for installation. The user or process that tries to enable an optional Windows feature requires appropriate access rights to file shares and/or WIM files.

If you select **Never attempt to download payload from Windows Update**, Windows Update is not contacted during an installation or repair operation.

If you select **Contact Windows Update directly to download repair content instead of Windows Server Update Services (WSUS)**, any attempt to add features (for example, .NET Framework 3.5) or repair the feature

file store, uses Windows Update to download files. Target computers require Internet and Windows Update access for this option.

**Note** Windows Server Update Services (WSUS) is not supported as a source for FoD or feature file store repair.

For Windows 8 and Windows Server 2012, WSUS is not supported as a source for feature installation (for example, adding .NET Framework 3.5 feature files) or feature file store repair operations. WSUS core scenarios include centralized update management and patch management automation, which enables administrators to manage the distribution of updates that are released through Microsoft Update to computers in their network. FoD and feature file store repair rely on download of individual files to perform update or repair operations. For example, if a single file becomes corrupted, then only that file (which could be as small as a few kilobytes) is downloaded from the repair source. WSUS can use either full or express files to perform servicing update operations; however, these files are not compatible with FoD or feature file store repair.

If an alternate source path is used to repair images, consider the following guidelines:

- **Servicing updates**

Keep any repair source current with the latest servicing updates. If you are using an image from a WIM file for FoD, you can use the Deployment Image Servicing and Management (DISM) tool to service the image. For more information, see [Mount and Modify an Image Using DISM](#). If you are using an online Windows installation that is shared on your local network as a repair image, make sure that the computer has access to Windows Update.

- **Multilingual images**

You must include all relevant language packs with your repair source files for the locales that your image supports. If you restore a feature without all localization files that the Windows installation requires for that feature, installation fails. You can install additional language packs after a feature is restored.

## Related topics

[Microsoft .NET Framework 3.5 Deployment Considerations](#)

# Deploy .NET Framework 3.5 by using Deployment Image Servicing and Management (DISM)

1/24/2019 • 3 minutes to read

You can use the Deployment Image Servicing and Management (DISM) command-line tool to create a modified image to deploy .NET Framework 3.5.

## IMPORTANT

For images that will support more than one language, you must add .NET Framework 3.5 binaries before adding any language packs. This order ensures that .NET Framework 3.5 language resources are installed correctly in the reference image and available to users and applications.

## Using DISM with Internet connectivity

### Requirements

- Internet connection
- Access to Windows Update. If the PC or server is behind a firewall or uses a proxy server, see [KB900935 - How the Windows Update client determines which proxy server to use to connect to the Windows Update Web site](#).
- Windows 8, Windows Server 2012, or the [Windows Assessment and Deployment Kit \(ADK\)](#) tools.
- Installation media
- Administrator user rights. The current user must be a member of the local Administrators group to add or remove Windows features.

### For an online reference image that can access Windows Update

1. Open a command prompt with administrator user rights (Run as Administrator) in Windows 8 or Windows Server 2012.
2. To Install .NET Framework 3.5 feature files from Windows Update, use the following command:

```
DISM /Online /Enable-Feature /FeatureName:NetFx3 /All
```

Use `/All` to enable all parent features of the specified feature. For more information on DISM arguments, see [Enable or Disable Windows Features Using DISM](#).

3. On Windows 8 PCs, after installation .NET Framework 3.5 is displayed as enabled in **Turn Windows features on or off** in Control Panel. For Windows Server 2012 systems, feature installation state can be viewed in Server Manager.

### For an offline reference image

1. Run the following DISM command (image mounted to the **c:\test\offline** folder and the installation media in the **D:\drive**) to install .NET 3.5:

```
DISM /Image:C:\test\offline /Enable-Feature /FeatureName:NetFx3 /All /LimitAccess /Source:D:\sources\sxs
```

Use */All* to enable all parent features of the specified feature.

Use */LimitAccess* to prevent DISM from contacting Windows Update/WSUS.

Use */Source* to specify the location of the files that are needed to restore the feature.

To use DISM from an installation of the Windows ADK, locate the Windows ADK servicing folder and navigate to this directory. By default, DISM is installed at **C:\Program Files (x86)\Windows Kits\8.0\Assessment and Deployment Kit\Deployment Tools**. You can install DISM and other deployment and imaging tools, such as Windows System Image Manager (Windows SIM), on another supported operating system from the Windows ADK. For information about DISM-supported platforms, see [DISM Supported Platforms](#).

- Run the following command to look up the status of .NET Framework 3.5 (offline image mounted to **c:\test\offline**):

```
DISM /Image:c:\test\offline /Get-Features /Format:Table
```

A status of **Enable Pending** indicates that the image must be brought online to complete the installation.

## Using DISM with no Internet connectivity

You can use DISM to add .NET Framework 3.5 and provide access to the **\sources\SxS** folder on the installation media to an installation of Windows that is not connected to the Internet.

### WARNING

If you're not relying on Windows Update as the source for installing the .NET Framework 3.5, make sure to use sources from the same corresponding Windows operating system version. Using a source path that doesn't correspond to the same version of Windows won't prevent a mismatched version of .NET Framework 3.5 from being installed. This can cause the system to be in an unsupported and unserviceable state.

### Requirements

- Windows 8, Windows Server 2012, or the [Windows ADK](#) tools.
- Installation media
- Administrator user rights. The current user must be a member of the local Administrators group to add or remove Windows features.

### Steps

- Open a command prompt with administrator user rights (Run as Administrator).
- To install .NET Framework 3.5 from installation media located on the **D:** drive, use the following command:

```
DISM /Online /Enable-Feature /FeatureName:NetFx3 /All /LimitAccess /Source:d:\sources\sxs
```

Use */All* to enable all parent features of the specified feature.

Use */LimitAccess* to prevent DISM from contacting Windows Update/WSUS.

Use */Source* to specify the location of the files that are needed to restore the feature.

For more information on DISM arguments, see [Enable or Disable Windows Features Using DISM](#).

On Windows 8 PCs, after installation, .NET Framework 3.5 is displayed as enabled in **Turn Windows features on**

**or off** in Control Panel.

## Related topics

[Microsoft .NET Framework 3.5 Deployment Considerations](#)

# Enable .NET Framework 3.5 by using Windows PowerShell

1/18/2019 • 2 minutes to read

For a Windows Server installation that is not connected to the Internet, you can use Windows PowerShell to add .NET Framework 3.5 and provide access to the **\sources\sxs** folder on the installation media. The **\sources\sxs** folder can be copied to network share (for example, **\network\share\sxs**) to make it easily accessible to multiple computers. The target computer account **DOMAIN\SERVERNAME\$** must have at least read access to the network share.

## Requirements

- Windows Server 2012 or Windows Server 2016
- Installation media
- Administrator user rights. The current user must be a member of the local Administrators group to add or remove Windows features.
- Target Computers might need network access and rights to use either alternate sources or an Internet connection to use Windows Update.

## Steps

1. Start Windows PowerShell in the Administrator Command Prompt by typing:

```
powershell
```

2. To install .NET Framework 3.5 from installation media located on a network share, use the following command:

```
Install-WindowsFeature Net-Framework-Core -source \\network\share\sxs
```

Where **\network\share\sxs** is the location of the source files.

For more information about the `Install-WindowsFeature` cmdlet, see [Install-WindowsFeature](#).

3. To verify installation, run the following command:

```
Get-WindowsFeature
```

The **Install State** column should show **Installed** for the **.NET Framework 3.5 (includes .NET 2.0 and 3.0)** feature.

## Related topics

[Microsoft .NET Framework 3.5 Deployment Considerations](#)

# Enable .NET Framework 3.5 by using the Add Roles and Features Wizard

1/18/2019 • 2 minutes to read

You can use Server Manager to enable .NET Framework 3.5 for a local or remote installation of Windows Server 2012 R2.

## Requirements

- Windows Server 2012 R2
- Installation media
- Administrator user rights. The current user must be a member of the local Administrators group to add or remove Windows features.
- Target Computers might need network access and rights to use either alternate sources or an Internet connection to use Windows Update.

## Steps

1. In Server Manager, click **Manage** and then select **Add Roles and Features** to start the Add Roles and Features Wizard.
2. On the **Select installation type** screen, select **Role-based or feature-based installation**.
3. Select the target server.
4. On the **Select features** screen, check the box next to **.Net Framework 3.5 Features**.
5. On the **Confirm installation selections** screen, a warning will be displayed asking **Do you need to specify an alternate source path?**. If the target computer does not have access to Windows Update, click the **Specify an alternate source path** link to specify the path to the **\sources\sxs** folder on the installation media and then click **OK**. After you have specified the alternate source, or if the target computer has access to Windows Update, click the **X** next to the warning, and then click **Install**.

If you are using Server Manager in Windows Server 2012 to add a role or feature to a remote server, the remote server's computer account (DOMAIN\ComputerName\$) requires access to the alternate source file path because the deployment operation runs in the SYSTEM context on the target server.

## Related topics

[Microsoft .NET Framework 3.5 Deployment Considerations](#)

# .NET Framework 3.5 deployment errors and resolution steps

1/24/2019 • 2 minutes to read

This topic describes common errors you can encounter when you use Features on Demand to enable or deploy .NET Framework 3.5, and recommended steps to resolve the issues.

## Features on Demand Error Codes

Error Code	Name	Description	Resolution Steps
0x800F081F	CBS_E_SOURCE_MISSING	The source files could not be found. Use the <b>Source</b> option to specify the location of the files that are required to restore the feature. For more information about how to specify a source location, see <a href="#">Configure a Windows Repair Source</a> .	Verify that the source specified has the necessary files. The source argument should point to the <b>\sources\sxs folder</b> on the installation media or the Windows folder for a mounted image (for example, <b>c:\mount\windows</b> for an image mounted to <b>c:\mount</b> ).
0x800F0906	CBS_E_DOWNLOAD_FAILURE	The source files could not be downloaded. Use the <b>Source</b> option to specify the location of the files that are required to restore the feature. For more information about how to specify a source location, see <a href="#">Configure a Windows Repair Source</a> .  Windows couldn't connect to the Internet to download necessary files. Make sure that the system is connected to the Internet and click <b>Retry</b> .	Verify that the computer or server has connectivity to Windows Update, and that you are able to browse to <a href="https://update.microsoft.com">https://update.microsoft.com</a> . If WSUS is used to manage updates for this computer, verify that the Group Policy setting <b>Contact Windows Update directly to download repair content instead of Windows Server Update Services (WSUS)</b> is enabled.

Error code	Name	Description	Resolution steps
0x800F0907	CBS_E_GROUPOLICY_DISALLOWED	<p>DISM failed. No operation was performed. For more information, review the log file at <b>%WINDIR%\logs\DISM\dism.log</b>.</p> <p>Due to network policy settings, Windows couldn't connect to the Internet to download files required to complete the requested changes.</p>	<p>Contact your network administrator for assistance with the <b>Specify settings for optional component installation and component repair</b> Group Policy setting.</p>

## Related topics

[Microsoft .NET Framework 3.5 Deployment Considerations](#)

# Battery Life

1/18/2019 • 2 minutes to read

In this section, you will learn about managing battery life when you deploy Windows 8 and Windows Server® 2012 on different hardware and software platforms.

## In This Section

<a href="#">Managing Battery Life and Power Consumption Overview</a>	Describes considerations that can help you to meet battery life goals, and lists common Windows® power policy settings that can affect battery life.
<a href="#">Set the Default Power Plan</a>	Describes how to import a power plan and how to set a power plan to the active power plan.
<a href="#">Create a Custom Power Plan</a>	Describes how to create a power plan by using Control Panel, how to export the power plan, and how to import the power plan on a destination computer.
<a href="#">Fine-Tune a Custom Power Plan</a>	Describes how to configure a customized Windows power plan by using powercfg command-line options.
<a href="#">Test Battery Life and Power Consumption</a>	Describes how to test power consumption.

## Related topics

[Mobile Battery Life Solutions: A Guide for Mobile Platform Professionals](#)

[Windows Performance Toolkit](#)

[Power Policy Configuration and Deployment in Windows](#)

# Managing Battery Life and Power Consumption Overview

1/18/2019 • 4 minutes to read

Windows®-based laptops must meet energy-efficiency regulatory requirements, such as the United States Environmental Protection Agency (EPA) Energy Star program. Furthermore, surveys have shown that longer battery life for laptops continues to be a leading request from consumers.

Hardware and software factors such as a low-capacity battery, a processor-intensive driver, or a poorly configured power setting can cause a significant reduction in battery life. When you design your system, you should experiment with multiple configurations of each of these factors to find the best balance of battery life and performance.

## Hardware

This section lists a few of the common hardware design considerations that can affect battery life.

- **Battery capacity.** Check with your battery manufacturer to determine the battery capacity.
- **Other hardware components.** Ask your hardware component manufacturers for power-consumption test results for each hardware component.

For information on each of these battery-life factors, see [Mobile Battery Life Solutions: A Guide for Mobile Platform Professionals](#).

## Software

This section lists a few of the common software design considerations that can affect battery life.

- **Drivers.** As you add each new driver to the system, observe the driver's impact on power consumption. A single poorly performing driver can greatly affect system performance.
- **Applications, services, and other software.** As you add each new software application to the system, observe the application's impact on power consumption. A single poorly performing application can greatly affect system performance.
- **Windows power policy settings.** Optimize Windows power policy settings to balance performance needs and battery life. For more information, see the section: [Windows Power Policy Settings](#).

For more information about each of these battery life factors, see [Mobile Battery Life Solutions: A Guide for Mobile Platform Professionals](#).

## Windows Power Policy Settings

This section lists a few of the common configurable settings that can affect battery life. Test these and other settings to create an optimal power plan for your system.

Settings can be specific to whether the computer is plugged in (AC) or on battery power (DC). You can configure the following settings:

- **Display brightness**

The most effective way to reduce the power consumption on a mobile computer when the display is in use

is to reduce the display brightness. The attached display is the largest power consumer. The display uses up to 40 percent of the overall system power consumption.

By default, Windows significantly reduces the display brightness when a mobile computer is on battery power. Depending on your hardware and the needs of your users, you can adjust the default display brightness setting lower to increase battery life, or higher to make the display easier to read.

- **Display timeout**

Mobile PC battery life can be significantly extended by using a short display idle timeout.

**Note Display dimming:** Mobile computers that run Windows 8.1 and Windows Server 2012 R2 will dim the display 15 seconds before the **Display timeout**. This value is no longer configurable.

- **Hard disk timeout**

Although the hard disk drive is not the primary power consumer in the typical mobile computer, you may be able to save power by increasing the hard drive timeout.

When the hard drive is idle for a period of time, the hard drive's motor stops. The next time that the computer needs to access the hard drive, the system response may be slow while the hard drive begins to spin again.

Depending on your hardware and the needs of your users, you can adjust the default hard disk timeout lower to increase battery life, or higher to increase the availability of the hard disk.

- **Sleep mode**

By default, if the processor is idle, and the end user is not using the computer, Windows enters low-power sleep mode or hibernate mode. The next time that the computer needs processor power, system response may be slow while the processor recovers.

Depending on your hardware and the needs of your users, you can adjust the default sleep timer lower to increase battery life, or higher to increase the availability of the processor.

- **Wireless adapter power-save modes**

By default, Windows configures the 802.11 power-save mode to **Maximum Performance** for both AC and battery power. This configuration keeps the wireless adapter active, even when data is not being transferred. This alleviates compatibility problems between some wireless adapters and access points that are not compatible with 802.11 power-save modes.

If you create custom power policies to save more power and help extend battery life, consult with the manufacturer of the wireless adapter about the effects of changing the power policy value to **Maximum Power Saving** or **Medium Power Saving**.

You can manually modify the power settings for each built-in power configuration. To learn more about these settings and other common configurable power settings, see [Mobile Battery Life Solutions: A Guide for Mobile Platform Professionals](#) and [Power Policy Configuration and Deployment in Windows](#).

## Related topics

[Mobile Battery Life Solutions: A Guide for Mobile Platform Professionals](#)

[Set the Default Power Plan](#)

[Create a Custom Power Plan](#)

[Windows Performance Toolkit](#)



# Set the Default Power Plan

1/18/2019 • 2 minutes to read

Use these instructions to set a default power plan when deploying Windows 8 or Windows Server® 2012 PCs. A power plan is also known as a *power scheme*.

**Note** This page gives information about manufacturing PCs.

To modify a power plans on your own PC, see [Power Plans: Frequently asked questions](#).

## To set the default power plan

1. On your technician computer, open an elevated command prompt.
2. If you want to use a power plan from another computer, import the power plan.

For example, to import a power plan that is named OutdoorPlan, type the following at a command prompt:

```
powercfg -IMPORT C:\OutdoorPlan.pow
```

3. Type the following to find the GUID for all the power plans on the computer:

```
powercfg -LIST
```

The computer returns the list of available power plans. The following examples refer to these plans as *guidPlan1* and *guidPlan2*.

```
Existing Power Schemes (* Active)
-----
Power Scheme GUID: {guidPlan1} (Balanced) *
Power Scheme GUID: {guidPlan2} (Power saver)
```

4. Note the GUIDs that are listed next to the power plans that you want to change.
5. Set the power plan that you want to set as the default as the active power plan. For example, you can use the following command:

```
powercfg -SETACTIVE {guidPlan2}
```

where *guidPlan2* is the name of the power plan.

This command can be run by using a custom command in an answer file, or by opening an elevated command prompt in audit mode.

## To confirm that the default power plan

1. Click **Start**, and then select **Control Panel**.
2. Click **Hardware and Sound**, and then select **Power Options**.
3. Review each power plan.

The **Power Options** Control Panel opens, and the power plans appear.

4. Verify that the correct plan is set as the active power plan. The computer shows an asterisk (\*) next to the active power plan.

## Related topics

[Add a Custom Command to an Answer File](#)

[Boot Windows to Audit Mode or OOBE](#)

[Create a Custom Power Plan](#)

[Power Policy Configuration and Deployment in Windows](#)

# Create a Custom Power Plan

1/24/2019 • 2 minutes to read

A *power plan* is a collection of hardware and system settings that manages how computers use and conserve power. A power plan is also known as a *power scheme*. You can create custom power plans that are optimized for specific computers.

By default, Windows 8 and Windows Server® 2012 include three power plans: **Balanced**, **Power Saver**, and **High Performance**. You can customize these existing plans for your systems, create new plans that are based on the existing plans, or create a new power plan from scratch.

Optimizing Windows power plans can help improve battery life. However, a single poorly performing application, device, or system feature can significantly reduce battery life. For information about factors that influence battery life, see [Managing Battery Life and Power Consumption Overview](#).

## In this topic

[Creating a customized power plan](#)

[Listing the available power plans](#)

[Deploying a power plan](#)

### **Creating a customized power plan**

1. Click **Start**, and then select **Control Panel**.
2. Click **Hardware and Sound**, and then select **Power Options**.
3. The **Power Options** Control Panel opens, and the power plans appear.
4. Click **Create a power plan**.
5. Follow the on-screen instructions to create and customize a power plan file that is based on an existing plan. Name your power plan "OutdoorPlan".

**Note** You can manage most common power plan settings through Control Panel. To fine-tune settings that do not appear in Control Panel, see [Fine-Tune a Custom Power Plan](#).

### **Listing the available power plans**

- On your technician computer, at an elevated command prompt, type the following:

```
powercfg -LIST
```

The computer will return the list of available power plans. In the following example, these plans are *Balanced*, *Power saver*, and *OutdoorPlan*.

```
Existing Power Schemes (* Active)
-----
Power Scheme GUID: {guidPlan1} (Balanced) *
Power Scheme GUID: {guidPlan2} (Power saver)
Power Scheme GUID: {guidPlan3} (OutdoorPlan)
```

Note the GUIDs that are listed next to the power plans that you want to capture.

## Deploying a power plan

After you have created power plans that work for your system, you can deploy the power plans to your destination computers.

To export the OutdoorPlan power plan that you created on your technician computer, open an elevated command prompt, and then type the following

```
powercfg -EXPORT C:\OutdoorPlan.pow {guidPlan-New}
```

This creates a new power plan file.

To learn more, see [Set the Default Power Plan](#).

## Related topics

[Managing Battery Life and Power Consumption Overview](#)

[Test Battery Life and Power Consumption](#)

[Set the Default Power Plan](#)

# Fine-Tune a Custom Power Plan

1/18/2019 • 3 minutes to read

A power plan is a collection of hardware and system settings that manages how computers use and conserve power. You can create custom power plans that are optimized for specific computers.

You can manage most common power plan settings through Control Panel. For more information, see [Create a Custom Power Plan](#). To fine-tune hardware-specific configurations that are not configurable through Control Panel, use the PowerCfg tool.

## Manually Modifying a Power Plan

You can customize all configurable Windows power options by using the `powercfg` command from an elevated command prompt. This includes hardware-specific configurations that are not configurable through Control Panel.

### To list the available power plans

- On your technician computer, at an elevated command prompt, type the following:

```
powercfg -LIST
```

The computer will return the list of available power plans. In the following examples, these plans are *Balanced* and *Power saver*.

```
Existing Power Schemes (* Active)
-----
Power Scheme GUID: {guidPlan1} (Balanced) *
Power Scheme GUID: {guidPlan2} (Power saver)
```

Note the GUIDs that are listed next to the power plans that you want to change. You will need these GUIDs to manually update settings and capture the power plans.

### To set the power plan to be modified as active

- To modify a plan, use the GUID of the power plan that you want to change to set that power plan as the active power plan. For example:

```
powercfg -SETACTIVE {guidPlan2}
```

### To adjust the settings

- This section describes how to manually configure other power configuration settings by using the `powercfg` command. Test these settings to create an optimal power plan for your system.

#### Find information about the existing power setting.

- At an elevated command prompt, type the following:

```
powercfg -QUERY
```

The computer displays information for all of the power settings for this plan.

- b. Find the GUID for the subgroup of the setting that you want to change. For example, to modify a display setting, find the GUID for the Display subgroup:

```
Subgroup GUID: {guidSubgroup-Display} (Display)
```

- c. Find the GUID for the setting that you want to change. For example, to modify the Display Brightness setting, find the GUID for the (Display brightness) setting:

```
Power Setting GUID: {guidPowerSetting-Brightness} (Display brightness)
```

- d. Review the information from the query command, review the possible settings, and determine a value that works for your computer.

**Note** You must enter these values by using decimal integers. However, the values appear on the screen as hexadecimal values that are specific to the setting.

For example, to set the maximum display brightness to 50 percent brightness, enter the value as 50.

When you use the `powercfg -QUERY` command to confirm the setting, the value appears as 0x00000032.

```
Power Setting GUID: {guidPowerSetting-Brightness} (Display brightness)
Minimum Possible Setting: 0x00000000
Maximum Possible Setting: 0x00000064
Possible Settings increment: 0x00000001
Possible Settings units: %
Current AC Power Setting Index: 0x00000064
Current DC Power Setting Index: 0x00000032
```

2. Adjust the value for the power setting for times when the computer is plugged in. For example, to set the display brightness level to 100 percent when the computer is plugged in, type the following:

```
powercfg -SETACVALUEINDEX {guidPlan-New} {guidSubgroup-Display} {guidPowerSetting-Brightness} 100
```

3. Adjust the value for the power setting for times when the computer is on battery power. For example, to set the display brightness level to 75 percent when the computer is on battery power, type the following:

```
powercfg -SETDCVALUEINDEX {guidPlan-New} {guidSubgroup-Display} {guidPowerSetting-Brightness} 75
```

4. Use the **Query** command to verify the setting. For example:

```
powercfg -QUERY
```

The computer shows the new power setting index in hexadecimal notation. For example:

```
Power Setting GUID: {guidPowerSetting-Brightness} (Display brightness)
Minimum Possible Setting: 0x00000000
Maximum Possible Setting: 0x00000064
Possible Settings increment: 0x00000001
Possible Settings units: %
Current AC Power Setting Index: 0x00000064
Current DC Power Setting Index: 0x0000004b
```

The hexadecimal value 0x00000064 represents 100 percent display brightness when the computer is

plugged in. The hexadecimal value 0x0000004b represents 75 percent display brightness when the computer is using battery power.

## Related topics

[Create a Custom Power Plan](#)

[Set the Default Power Plan](#)

# Test Battery Life and Power Consumption

1/18/2019 • 2 minutes to read

Compare the overall system power to the power that the system consumes when you use a clean installation. With preinstalled applications and power policies, some computers have shown a 40 percent decrease in battery performance compared with a clean Windows® installation. However, through careful engineering, computers can achieve equal or improved performance over a clean Windows installation.

## Determining the causes of heavy battery use

You can determine the causes of heavy battery use by looking at the way that the system uses resources. You can find many system problems by looking at processor idle time. Use a performance tool, such as Windows Assessments or the Windows Performance Toolkit, to investigate when the system has spikes in power use. Performance tools can help you target the applications that are causing increased processor use.

For more information, see [Windows Assessment Toolkit](#) and [Windows Performance Toolkit](#).

## Related topics

[Set the Default Power Plan](#)

[Create a Custom Power Plan](#)

# Device Drivers

1/18/2019 • 12 minutes to read

You can add device drivers to a Windows image before, during, or after you deploy the image. When planning how to add drivers to your Windows deployment, it's important to understand how driver folders are added to the image, how driver ranking affects deployment, and the digital signature requirements for drivers.

In this topic:

- [Adding Drivers](#)
- [Managing Driver Folders](#)
- [Understanding Driver Ranking](#)
- [Understanding Digital Signature Requirements](#)
- [Additional Resources](#)

## Adding Drivers

You can add device drivers to a Windows image:

- [Before deployment on an offline Windows image](#)
- [During an automated deployment](#)
- [After deployment on a running operating system](#)

For more information, see [Understanding Servicing Strategies](#).

### Add drivers before deployment on an offline Windows image by using DISM

Offline servicing occurs when you modify a Windows image entirely offline without booting the operating system. You can add, remove, and enumerate drivers on an offline Windows image by using the DISM command-line tool. DISM is installed with Windows and is also distributed in the Windows Assessment and Deployment Kit (Windows ADK). For more information about DISM, see the [DISM - Deployment Image Servicing and Management Technical Reference for Windows](#).

When you add a driver to an offline image, it's either staged or reflected in the image:

- **Boot-critical drivers** are reflected. In other words, the files are copied into the image according to what's specified in the .inf file. The PC completes installation tasks during the initial boot, including updating the Critical Devices Database (CDDB) and the registry.
- **Drivers that aren't boot critical** are staged. In other words, they're added to the driver store. After Windows starts, PnP detects the device and installs the matching driver from the driver store.

You can use DISM commands to add or remove drivers on a mounted or applied Windows or Windows Preinstallation Environment (Windows PE) image.

**Note** You can't use DISM to remove inbox drivers (drivers that are installed on Windows by default). You can use it only to remove third-party or out-of-box drivers.

You can also use DISM commands to apply an unattended answer file to a mounted or applied Windows image.

For more information, see [Add and Remove Drivers to an Offline Windows Image](#).

If you're using DISM, you can add only .inf drivers to an offline Windows image. Drivers that display the Designed for Windows logo are provided as .cab files. You must expand the .cab file before you install the .inf file if you're using DISM for the installation. You must install a driver that's packaged as a .exe file or another file type on a running Windows operating system. To run a .exe or Windows Installer (.msi) driver package, you can add a custom command to an answer file to install the driver package. For more information, see [Add a Custom Command to an Answer File](#).

### Add drivers during an automated deployment by using Windows Setup and an answer file

You can use an unattended answer file to add drivers to an image when you use Windows Setup for deployment. In this answer file, you can specify the path of a device driver on a network share (or a local path). You accomplish this by adding the Microsoft-Windows-PnpCustomizationWinPE or Microsoft-Windows-PnpCustomizationNonWinPE components and specifying the configuration passes where you want to install them. When you run Windows Setup and specify the name of the answer file, out-of-box drivers are staged (added to the driver store on the image), and boot-critical drivers are reflected (added to the image so that they'll be used when the computer boots). Setup uses the answer file. By adding device drivers during the **windowsPE** or **offlineServicing** configuration passes, you can add out-of-box device drivers to the Windows image before the computer starts. You can also use this method to add boot-critical device drivers to a Windows image. For more information, see [Add Device Drivers to Windows During Windows Setup](#). For more information about how Windows Setup works, see the [Windows Setup Technical Reference](#).

If you want to add boot-critical drivers to Windows PE, use the **windowsPE** configuration pass to reflect the drivers before the Windows PE image is booted. The difference between adding boot-critical drivers during the **windowsPE** configuration pass and adding them during the **offlineServicing** configuration pass is that during the **windowsPE** configuration pass, boot-critical drivers are reflected for Windows PE to use. During the **offlineServicing** configuration pass, the drivers are staged to the driver store on the Windows image.

Methods for adding device drivers by using Windows Setup include these:

- Using an answer file to add drivers during the **offlineServicing** configuration pass of Setup.
- Using an answer file to add drivers during the **windowsPE** configuration pass of Setup.
- For Windows Server, placing drivers in the \$WinPEDriver\$ directory to be installed automatically during the **windowsPE** configuration pass of Setup. All drive letters with a value of C or greater are scanned for a \$WinPEDriver\$ directory. The drive must be accessible to the hard disk during Setup. Make sure that the drive does not require a storage driver to be loaded before it can be accessed.

For more information about these and other configuration passes, see [Windows Setup Configuration Passes](#).

When you're using Windows Deployment Services for deployment in Windows Server, you can add device drivers to your server and configure them to be deployed to clients as part of a network-based installation. You configure this functionality by creating a driver group on the server, adding packages to it, and then adding filters to define which clients will install those drivers. You can configure drivers to be installed based on the client's hardware (for example, manufacturer or BIOS vendor) and the edition of the Windows image that's selected during the installation. You can also configure whether clients install all packages in a driver group or only the drivers that match the installed hardware on the client. For more information about how to implement this functionality, see the Windows Deployment Services documentation.

### Add drivers after deployment on a running operating system by using PnUtil or an answer file

You can use the PnUtil tool to add or remove drivers on a running operating system. Alternatively, you can use an answer file to automate the installation of the drivers when the computer is booted in audit mode. These methods can be helpful if you want to maintain a simple Windows image, and then add only the drivers that are required for a specific hardware configuration. For more information about how to use audit mode, see [Boot Windows to Audit Mode or OOBE](#).

Methods for adding device drivers online to a running operating system include these:

- Using PnPUtil to add or remove PnP drivers. For more information, see [Use PnPUtil at a command line to install a Plug and Play device](#).
- Using an answer file to automate the installation of PnP drivers when the computer is booted in audit mode. For more information, see [Add a Driver Online in Audit Mode](#).

## Drivers for Windows 10 S

Drivers in Windows 10 S must meet certain requirements. See [Windows 10 S driver requirements](#) to learn about the types of drivers you can add to Windows 10 S.

## Managing Driver Folders

If you're adding multiple drivers, you should create separate folders for each driver or driver category. This makes sure that there are no conflicts when you add drivers that have the same file name. After the driver is installed on the operating system, it's renamed to Oem\*.inf to ensure unique file names in the operating system. For example, the staged drivers named MyDriver1.inf and MyDriver2.inf are renamed to Oem0.inf and Oem1.inf after they're installed.

When you specify a device-driver path in an answer file, all .inf drivers in the specified directory and subdirectories are added to the driver store of the Windows image,

`%SystemRoot%\System32\DriverStore\FileRepository`. For example, if you want all of the drivers in the C:\MyDrivers\Networking, C:\MyDrivers\Video, and C:\MyDrivers\Audio directories to be available in your Windows image, specify the device-driver path, C:\MyDrivers, in your answer file. If you're not using an answer file, you can use the **/recurse** command in DISM. For more information about the **/recurse** command, see [DISM Driver Servicing Command-Line Options](#). This command makes sure that all drivers in each subdirectory will be added to the driver store in your Windows image.

If all drivers in the specified directory and subdirectories are added to the image, you should manage the answer file or your DISM commands and these directories carefully. Do your best to address concerns about increasing the size of the image through unnecessary driver packages.

If it isn't practical to manage your driver shares so that only the required drivers are added to your image, you can use the Driver Package Installer (DPIInst) tool to add drivers that aren't boot critical online. DPIInst selectively installs drivers that aren't boot critical only if the hardware is present or if the driver package is a better match for the device.

## Understanding Driver Ranking

One of the most common issues in deploying drivers happens when a driver is successfully imported into the driver store but, after the system is online, PnP finds a higher-ranking driver and installs that driver instead.

The Windows PnP manager ranks these driver package properties in order of importance:

1. Signing
2. PnP ID match
3. Driver date
4. Driver version

For example, if a device has a better PnP ID match but is unsigned, a signed driver that has a compatible ID match takes precedence. An older driver can outrank a newer driver if the older driver has a better PnP ID match or signature.

For more information about driver ranking, see [How Windows Ranks Drivers](#).

# Understanding Digital Signature Requirements

Signed device drivers are a key security feature in Windows. Drivers that are installed on x64-based computers must have a digital signature. Although it isn't required, we recommend making sure that drivers are signed before you install them on x86-based computers.

All boot-critical drivers must contain embedded signatures. A digital signature isn't required for Plug and Play (PnP) drivers. But when an unsigned PnP driver is installed on a running operating system, administrator credentials are required, and you can't install such drivers on 64-bit operating systems.

There are two ways that a driver can be signed:

- Kernel mode and boot-critical drivers are digitally signed via a method called embedded signing. By using embedded signatures, every binary in the driver package is signed. Embedded signatures improve boot loading performance. For drivers that are not PnP, signatures should be embedded so that they're not lost during an upgrade of the operating system.
- Digitally signed PnP drivers contain a catalog (.cat) file that's digitally signed. The catalog file contains a hash of all the files in the driver's .inf file for installation. A signed catalog file is all that's required to correctly install most PnP drivers.

Either of these sources can sign drivers:

- Windows Hardware Quality Labs (WHQL), which makes sure that your drivers qualify for the Windows Hardware Certification Program. WHQL creates a signed driver catalog. For boot-critical drivers, you should add embedded signatures instead of relying on the catalog. Embedded signatures in boot-critical driver image files optimize boot performance of the operating system by eliminating the requirement to locate the appropriate catalog file when the operating system loader verifies the driver signature.
- A certification authority (CA), by using a Software Publishing Certificate (SPC). For boot-critical and x64-based kernel drivers, Microsoft provides an additional certificate that can be used to cross-sign the drivers. Drivers that aren't boot critical don't have to be cross-signed by Microsoft or embedded. You can use the Windows Kernel Mode Code Signing process if you need the flexibility of signing the drivers yourself. For information about digital signatures for kernel modules on x64-based systems, see [64-Bit Driver Guidelines](#).

For testing, you can also use test certificates.

If you have received an unsigned driver from a vendor for testing, you can use a test signature to validate the driver and to test the installation. Test signing is the act of digitally signing an application by using a private key and a corresponding code-signing certificate that's trusted only in the confines of a test environment.

These are the primary ways to generate such test-signing certificates:

- Developers can generate their own self-signed certificates.
- A CA can issue certificates.

For either option, test-signing certificates must be clearly identified as appropriate only for testing. For example, the word "test" can be included in the certificate subject name, and additional legal disclaimers can be included in the certificate. Production certificates that are issued by commercial CAs must be reserved for signing only public beta releases and public final releases of software and internal line-of-business software.

For more information, see [Requirements for Device Driver Signing and Staging](#).

When you're adding test-signed driver packages to Windows, consider these points:

- You must install the test certificates on a running operating system. You can't install them offline.

- The certificate of the CA that issued the test certificate must be inserted in the Trusted Root Certification Authorities certificate store.

**Note** If the test certificate is self-signed—for example, by using the Certificate Creation Tool (MakeCert)—the test certificate must be inserted in the Trusted Root Certification Authorities certificate store.

- The test certificate that's used to sign the driver package must be inserted in the Trusted Publishers certificate store.
- You must add test certificates online (to a booted instance of the Windows image) before you can use the Deployment Image Servicing and Management (DISM) command-line tool to add test-signed drivers offline.
- DISM validates WHQL certifications only for boot-critical drivers. But, a DISM command-line option can override this behavior. For more information, see [DISM Driver Servicing Command-Line Options](#).
- To install and verify test-signed drivers on 64-bit operating systems, set the Windows boot configuration to test mode by using the BCDedit tool on the destination computer. Test mode verifies that the driver image is signed, but certificate path validation doesn't require the issuer to be configured as a trusted root authority. For the PnP driver installation and ranking logic to treat the driver correctly, the test certificate must be stored in the trusted certificate store of the operating system image. For information about test mode during development, see [64-Bit Driver Guidelines](#).

**Caution** If an unsigned or invalid boot-critical device driver is installed on an x64-based computer, the computer will not boot. The unsigned or invalid boot-critical device driver will cause a Stop error. You should remove the driver from either the critical device database (CDDB) or its reflected location in the image. If you're performing an upgrade, make sure that unsigned drivers and their associated applications, services, or devices are removed or updated with a signed driver.

If you don't enable test mode by using BCDedit, and you have a test-signed driver installed, your computer will not boot. If you use DISM to remove the driver, all instances of the reflected driver package might not be removed. So, we recommend that you don't deploy images that have test-signed drivers installed.

## Additional Resources

These websites provide more information about device-driver requirements:

- For more information about PnP driver deployment, see [PnP Device Installation Signing Requirements](#).
- For more information about digital signatures and developing drivers, see the relevant page on the [Windows Hardware Developer Central](#) website.

## Related topics

[Add a Device Driver Path to an Answer File](#)

[Add a Driver Online in Audit Mode](#)

[DISM Driver Servicing Command-Line Options](#)

[Add and Remove Drivers to an Offline Windows Image](#)

[Add Device Drivers to Windows During Windows Setup](#)

[Maintain Driver Configurations when Capturing a Windows Image](#)

[BCDboot Command-Line Options](#)

[Deployment Troubleshooting and Log Files](#)

# Maintain Driver Configurations when Capturing a Windows Image

1/18/2019 • 9 minutes to read

A common deployment scenario is to capture a single Windows image from a reference computer and then apply the image to a group of destination computers that have identical hardware configurations.

To save time during installation and to speed up the out-of-box experience (OOBE) for end users, you can instruct Windows Setup to maintain the driver configurations from the reference computer as part of the Windows image. You should do this only when the hardware on the reference computer and the hardware on the destination computers are identical. When you do this, Windows Setup maintains driver configurations during image capture and deployment.

## Instructing Windows Setup to Maintain Driver Configurations

Before you capture an image, generalize the computer by using an answer file that instructs Windows Setup to maintain the driver configurations.

### To maintain driver configurations by using an answer file

1. On your technician computer, open Windows System Image Manager (Windows SIM). Click **Start**, type **Windows System Image Manager**, and then select **Windows System Image Manager**.
2. Create a new answer file, or update an existing answer file. For more information, see [Create or Open an Answer File](#) and [Best Practices for Authoring Answer Files](#).
3. Add the Microsoft-Windows-PnpSysprep/**PersistAllDeviceInstalls** setting. For more information, see the [Overview](#) section in this topic.
4. If the computer has undetectable hardware, include the Microsoft-Windows-PnpSysprep/**DoNotCleanUpNonPresentDevices** setting. For more information, see the [Undetectable hardware](#) section in this topic.
5. Generalize the computer by using the answer file. For example:

```
Sysprep /generalize /unattend:C:\unattend.xml
```

## Overview

The Windows in-box driver packages include device drivers that support a wide variety of popular hardware. If your specific hardware requires additional device drivers to boot, you can preinstall additional device drivers on your Windows image. Independent Hardware Vendors (IHVs) often supply these additional device drivers together with their device hardware. For more information about how to add device drivers, see [Add a Driver Online in Audit Mode](#).

To prepare a Windows image for deployment to multiple computers, you must use the System Preparation (Sysprep) tool to generalize the Windows image. Generalizing a Windows image removes the computer-specific information and prepares the device drivers for first boot. This preparation includes these steps:

- Device state for hardware is removed.

- Boot-critical driver settings are reset to their default values.
- Device log files are deleted.

When you generalize the computer, use an answer file that has the Microsoft-Windows-PnpSysPrep\PersistAllDeviceInstalls setting to save time. This setting prevents Windows Setup from removing and reconfiguring the device state for identical hardware. On first boot, the detected device drivers are already preconfigured, potentially enabling a quicker first-boot experience.

**Important** Avoid using the **PersistAllDeviceInstalls** setting when the hardware and the hardware configuration on the reference computer are not identical to those of the destination computers. Even seemingly minor differences in hardware or hardware configuration can cause severe or easily overlooked problems. For more information, see the [Troubleshooting Hardware Configuration Differences](#) section of this topic.

It's a good practice not to use the **PersistAllDeviceInstalls** setting on your primary reference image. Instead, for each group of computers that have a different hardware configuration, first load your primary reference image onto a new reference computer that has the planned hardware configuration. Next, capture a new image of this setup and use the **PersistAllDeviceInstalls** setting.

For more information about how to generalize the Windows image, see [Sysprep \(Generalize\) a Windows installation](#).

## Best Practices for Driver Revisions and Driver Ranking

Don't maintain multiple versions or revisions of the same driver package in the same image. Use offline or online servicing tools to update drivers.

Normally, when Windows Setup boots a computer and multiple versions of a driver package exist on that computer, Setup determines which driver to install by using driver ranking. But when you use the **PersistAllDeviceInstalls** setting, the normal driver-ranking processes don't occur. So, devices that use outdated drivers may remain installed. For more information about driver ranking, see [How Windows Ranks Drivers](#) on MSDN.

If you must add a device driver to an image that uses the **PersistAllDeviceInstalls** setting, you can update your device drivers by using one of the following methods:

- Use offline servicing tools, like the Deployment Image Servicing and Management (DISM) tool or an unattended answer file. For more information, see [Add and Remove Drivers to an Offline Windows Image](#).
- Use online servicing methods or tools, like an unattended answer file. For more information, see [Add a Driver Online in Audit Mode](#).

## Troubleshooting Hardware Configuration Differences

For the **PersistAllDeviceInstalls** setting to work correctly, the hardware configuration must be identical on the reference computer and on the destination computers. Hardware configuration includes the following components:

- **Hardware make and model.**
- **Firmware.** Updates, revisions, and configuration differences can cause some devices to report different criteria for matching device drivers or to use different resources. For example:
  - Peripheral Component Interconnect (PCI)-based devices can embed different subsystem revision numbers in their reported hardware IDs.
  - BIOS revisions can change the Advanced Configuration and Power Interface (ACPI) namespace. This causes Windows Setup to report existing devices differently or to introduce existing devices as new devices.

- BIOS system configuration differences can cause system devices to claim different memory, I/O, direct memory access (DMA), or interrupt request (IRQ) resources.
- **Physical location.** Hardware configurations must use the same slot, port, or socket numbers to connect to external devices. For example:
  - PCI expansion cards must be inserted in the same slot numbers.
  - USB devices must be connected or wired to the same port numbers on the same USB host controllers and integrated hubs.
  - Storage devices must be connected to the same storage controllers and channels.

### Low-risk, medium-risk, and high-risk differences in hardware configuration

When you use the **PersistAllDeviceInstalls** setting, any hardware differences can potentially cause problems. But some differences are more likely to cause problems than others.

#### Low-risk differences

For the following types of hardware differences, you may be able to work around potential driver conflicts and still use the **PersistAllDeviceInstalls** setting:

- CPU clock speed
- Amount of memory
- Hard disk capacity
- External input devices, like keyboards and mouse devices
- Monitors

#### Medium-risk differences

For the following types of hardware differences, we recommend that you don't use the **PersistAllDeviceInstalls** setting:

- Video cards
- Storage drives and media readers, like optical drives and card readers
- Internal or integrated bus devices, like USB or 1394 devices

When these types of hardware differences exist, using this setting may not reduce installation time, even if you work around potential driver conflicts.

#### High-risk differences

For major hardware differences, don't use the **PersistAllDeviceInstalls** setting. These differences include:

- Motherboard chipset or CPU brand
- Storage controllers
- Form-factor differences, like a change from a desktop to a laptop or from a laptop to a desktop
- Keyboard layout differences, like a change from a standard 101-key keyboard to a Japanese 106-key keyboard
- Any other devices that are in the enumeration path of the Windows boot volume

#### Types of problems that can occur with a hardware configuration change

Even seemingly minor differences in hardware or hardware configuration can cause severe or easily overlooked problems, like these:

- System instability
- Inability to use some of the basic or extended functionality of a device
- Extended boot times and extended installation times
- Misnamed devices in the Devices and Printers folder, Device Manager, and other device-related user interfaces
- Severe system problems that leave the computer in a non-bootable state

#### **Hardware configuration differences that can cause system boot failures**

When the boot-critical hardware is not identical on the reference computer and destination computers, using the **PersistAllDeviceInstalls** setting can cause severe system problems that can leave the computer in a non-bootable state.

Boot-critical driver packages can belong to any of the following Windows device setup classes, as identified by the ClassGUID directive in the <Version> section of the .inf files in their driver packages.

SYSTEM-SUPPLIED DEVICE SETUP CLASS	CLASSGUID
System	{4D36E97D-E325-11CE-BFC1-08002BE10318}
Computer	{4D36E966-E325-11CE-BFC1-08002BE10318}
Processor	{50127DC3-0F36-415E-A6CC-4CB3BE910B65}
PCMCIA	{4D36E977-E325-11CE-BFC1-08002BE10318}
HDC	{4D36E96A-E325-11CE-BFC1-08002BE10318}
SCSIAdapter	{4D36E97B-E325-11CE-BFC1-08002BE10318}
DiskDrive	{4D36E967-E325-11CE-BFC1-08002BE10318}
CDROM	{4D36E965-E325-11CE-BFC1-08002BE10318}
FDC	{4D36E969-E325-11CE-BFC1-08002BE10318}
FloppyDisk	{4D36E980-E325-11CE-BFC1-08002BE10318}
Volume	{71A27CDD-812A-11D0-BEC7-08002BE2092F}
USB	{36FC9E60-C465-11CF-8056-444553540000}
SBP2	{D48179BE-EC20-11D1-B6B8-00C04FA372A7}

SYSTEM-SUPPLIED DEVICE SETUP CLASS	CLASSGUID
1394	{6BDD1FC1-810F-11D0-BEC7-08002BE2092F}
Enum1394	{C459DF55-DB08-11D1-B009-00A0C9081FF6}
Keyboard	{4D36E96B-E325-11CE-BFC1-08002BE10318}
Mouse	{4D36E96F-E325-11CE-BFC1-08002BE10318}
HIDClass	{745A17A0-74D3-11D0-B6FE-00A0C90F57DA}
Ports	{4D36E978-E325-11CE-BFC1-08002BE10318}

For more information about these device setup classes, see [System-Supplied Device Setup Classes](#) on MSDN.

### Undetectable hardware

When you deploy a new computer to an end user, some hardware, like a removable device or a device that has an on/off switch, may not be present or detected during first boot. By default, on first boot, Windows Setup removes the preconfigured device state for undetected hardware.

To deploy hardware that may not be present or detected on first boot, add any applicable device drivers to the reference image, connect or turn on the applicable devices so that Windows can install them, and use the Microsoft-Windows-PnpSysprep/**DoNotCleanUpNonPresentDevices** setting when you capture the image.

**Important** Using the **DoNotCleanUpNonPresentDevices** setting can lead to the unnecessary storage of excess device states and contribute to slower boot times.

## Troubleshooting Driver Conflicts

To avoid driver conflicts between independent boot-critical driver packages, the IHV must make sure that each device driver uses different service names, registry key values, and binary file names.

### Example of a potential driver conflict

In the following example, a fictitious IHV named Fabrikam produces two types of storage controllers: StandardController and ExtremeController. Fabrikam assumes that only one type of storage controller is installed at a time on a particular computer.

The driver package defines the StandardController and ExtremeController configurations to use the same driver service name, storctrl. The storctrl driver service uses different service settings that change depending on which hardware (StandardController or ExtremeController) is installed. Because both StandardController and ExtremeController use the same service, they cannot coexist.

This sample shows the contents of the driver package file Storctrl.inf:

```

[Version]
Signature = "$WINDOWS NT$"
Class = SCSIAdapter
ClassGuid = {4D36E97B-E325-11CE-BFC1-08002BE10318}
...
[Manufacturer]
%Fabrikam% = Fabrikam,NTx86

[Fabrikam.NTx86]
%StandardController% = StandardController_DDInstall,PCI\VEN_ABCD&DEV_0001
%ExtremeController% = ExtremeController_DDInstall, PCI\VEN_ABCD&DEV_0002
...

[StandardController_DDInstall.Services]
AddService = storctrl,0x00000002,StandardController_ServiceInstall

[StandardController_ServiceInstall]
ServiceType = 1 ; SERVICE_KERNEL_DRIVER
StartType = 0 ; SERVICE_BOOT_START
ErrorControl = 1 ; SERVICE_ERROR_NORMAL
ImagePath = %12%\storctrl.sys
AddReg = StandardController_ServiceSettings

[StandardController_ServiceSettings]
HKR,Settings,LowPowerMode,0x00010001,1
HKR,Settings>ErrorCorrection,0x00010001,1
...

[ExtremeController_DDInstall.Services]
AddService = storctrl,0x00000002,ExtremeController_ServiceInstall

[ExtremeController_ServiceInstall]
ServiceType = 1 ; SERVICE_KERNEL_DRIVER
StartType = 0 ; SERVICE_BOOT_START
ErrorControl = 1 ; SERVICE_ERROR_NORMAL
ImagePath = %12%\storctrl.sys
AddReg = ExtremeController_ServiceSettings

[ExtremeController_ServiceSettings]
HKR,Settings,LowPowerMode,0x00010001,0
HKR,Settings>ErrorCorrection,0x00010001,4
...

```

If StandardController is on the reference computer and its settings are maintained during image capture, the storctrl driver service is preconfigured. If ExtremeController is on the destination computer, Windows may use the preconfigured settings and files that are intended for StandardController. This can cause unexpected results.

The IHV can help resolve the conflict by using one of these options:

- Create separate driver packages that have separate .inf files for each configuration, and import only the required driver package into the Windows image during deployment. For example, split Storctrl.inf into two separate .inf files, one version for StandardController and one version for ExtremeController, and import only the required driver package into the Windows image.
- Create another service in the driver package for each configuration. Give each service a different name (for example, storctrl and storctrlx) and point to a different binary image file (for example, Storctrl.sys and Storctrlx.sys).

## Related topics

# Add a Driver Online in Audit Mode

1/18/2019 • 2 minutes to read

You can use an answer file to automate the installation of device drivers when the computer is booted in audit mode.

## Adding a Device Driver

The **auditSystem** configuration pass processes unattended Setup settings while Windows is running in system context, before a user logs on to the computer in audit mode. The **auditSystem** configuration pass runs only if the computer is booted in audit mode. To add device drivers during the **auditSystem** configuration pass, add the **Microsoft-Windows-PnpCustomizationsNonWinPE** component to your answer file in the **auditSystem** configuration pass, and specify the path for each device driver. After you run Setup, boot Windows in audit mode. You can run the **Sysprep** command with the **/audit** option to configure the computer to start in audit mode the next time that it boots. Or, in the answer file, you can configure the Microsoft-Windows-Deployment\Reseal\Mode setting to **audit**. For more information, see [Unattended Windows Setup Reference](#).

### To add a device driver during the **auditSystem** configuration pass

1. Locate the .inf files that you want to install during audit mode for the device driver.
2. On your technician computer, open Windows System Image Manager (Windows SIM). Click **Start**, type **Windows System Image Manager**, and then select **Windows System Image Manager**.
3. Open your answer file and expand the **Components** node to display available settings.
4. Add the **Microsoft-Windows-PnpCustomizationsNonWinPE** component to your answer file in the **auditSystem** configuration pass.
5. Expand the **Microsoft-Windows-PnpCustomizationsNonWinPE** node in the answer file. Right-click **DevicePaths**, and then click **Insert New PathAndCredentials**.

A new **PathAndCredentials** list item appears.

6. For each location that you access, add a separate **PathAndCredentials** list item.
7. In the **Microsoft-Windows-PnpCustomizationsNonWinPE** component, specify the path of the device driver and the credentials that are used to access the file if the file is on a network share.

**Note** You can include multiple device-driver paths by adding multiple **PathAndCredentials** list items. If you add multiple list items, you must increment the value of **key** for each path. For example, if you add two separate driver paths, the first path uses the **key** value of **1**, and the second path uses the **key** value of **2**.

8. Save the answer file and close Windows SIM. The answer file must resemble this example:

```
<?xml version="1.0" encoding="utf-8" ?>
<unattend xmlns="urn:schemas-microsoft-com:unattend">
    <settings pass="auditSystem">
        <component name="Microsoft-Windows-PnpCustomizationsNonWinPE" processorArchitecture="x86"
publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS"
xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <DriverPaths>
                <PathAndCredentials wcm:keyValue="1" wcm:action="add">
                    <Credentials>
                        <Domain>Fabrikam</Domain>
                        <Password>MyPassword</Password>
                        <Username>MyUserName</Username>
                    </Credentials>
                    <Path>\\networkshare\share\drivers</Path>
                </PathAndCredentials>
            </DriverPaths>
        </component>
    </settings>
</unattend>
```

9. Boot in Windows Preinstallation Environment (Windows PE), run Windows Setup, and specify the name of the answer file. For example:

```
Setup /unattend:C:\unattend.xml
```

The specified answer file is cached to the system so that when you run audit mode, the computer applies settings in the answer file.

Setup finishes.

10. Run the **Sysprep** command with the **/audit** option to configure the computer to start in audit mode the next time that it boots. For example:

```
Sysprep /audit /reboot
```

When Windows reboots in audit mode, device drivers that you specified in the answer file are added.

You can use the Pnputil tool to add, remove, and enumerate drivers on a running operating system. For more information about how to use Pnputil to add or remove Plug and Play drivers, see [Install a Plug and Play Device](#).

## Related topics

[Device Drivers and Deployment Overview](#)

[Add Device Drivers to Windows During Windows Setup](#)

[DISM Driver Servicing Command-Line Options](#)

[Add and Remove Drivers to an Offline Windows Image](#)

[Audit Mode Overview](#)

[Boot Windows to Audit Mode or OOB](#)

# Add and Remove Drivers to an offline Windows Image

1/18/2019 • 5 minutes to read

You can use DISM to install or remove driver (.inf) files in an offline Windows or WinPE image. You can either add or remove the drivers directly by using the command prompt, or apply an unattended answer file to a mounted .wim, .ffu, .vhd, or .vhdx file.

When you use DISM to install a device driver to an offline image, the device driver is added to the driver store in the offline image. When the image is booted, Plug and Play (PnP) runs and associates the drivers in the store to the corresponding devices on the computer.

## NOTE

To add drivers to a Windows 10 image offline, you must use a technician computer running Windows 10, Windows Server 2016, or Windows Preinstallation Environment (WinPE) for Windows 10. Driver signature verification may fail when you add a driver to a Windows 10 image offline from a technician computer running any other operating system.

To learn how to add a driver on a running Windows PC, see [Add a driver online in audit mode](#) or [Install a plug and play device](#). To learn how to add a driver to a PC running WinPE, see [Drvload command line options](#).

## Driver types

- **.inf-style drivers:** Many drivers include an information file (with an .inf extension) to help install the driver. These can be installed using tools described in this topic.
- **.exe-style drivers:** Drivers without an .inf file often must be installed like typical Windows desktop applications. To learn how to add these, see [Add a driver online in Audit Mode](#)
- **Boot-critical drivers:** Graphics and storage drivers may sometimes need to be added to the Windows image (as shown in this topic), as well as the Windows PE image, and in the Windows recovery (WinRE) image.

## Add drivers to an offline Windows image

To add drivers to an offline image, you have to mount an image prior to adding drivers.

If you're adding drivers to a WinPE image, you can add them to the WinPE image in the output folder you specified when you [ran copype](#), for example: `C:\WinPE_amd64\media\sources\boot.wim`. This ensures that drivers will be included in WinPE each time you build WinPE media from that folder.

1. Mount a Windows image. For example:

```
Dism /Mount-Image /ImageFile:C:\test\images\install.wim /MountDir:C:\test\offline
```

See [Mount and modify a Windows image using DISM](#) for more info.

2. Add a driver to the image.

```
Dism /Image:C:\test\offline /Add-Driver /Driver:C:\drivers\mydriver.inf
```

To install all of the drivers from a folder and all its subfolders, point to the folder and use the **/Recurse**

option.

```
Dism /Image:C:\test\offline /Add-Driver /Driver:c:\drivers /Recurse
```

To see all DISM driver servicing command line options, see [DISM driver servicing command-line options](#).

#### WARNING

Using `/Recurse` can be handy, but it's easy to bloat your image with it. Some driver packages include multiple .inf driver packages, which often share payload files from the same folder. During installation, each .inf driver package is expanded into a separate folder. Each individual folder has a copy of the payload files. We've seen cases where a popular driver in a 900MB folder added 10GB to images when added with the `/Recurse` option.

To install an unsigned driver, use `**/ForceUnsigned**` to override the requirement that drivers installed on X64-based computers must have a digital signature.

...

```
Dism /Image:C:\test\offline /Add-Driver /Driver:C:\drivers\mydriver.inf /ForceUnsigned
```

...

3. Check to see if the driver was added. Drivers added to the Windows image are named Oem\*.inf. This guarantees unique naming for newly added drivers. For example, the files MyDriver1.inf and MyDriver2.inf are renamed Oem0.inf and Oem1.inf.

```
Dism /Image:C:\test\offline /Get-Drivers
```

4. Commit the changes and unmount the image.

```
Dism /Unmount-Image /MountDir:C:\test\offline /Commit
```

## Remove drivers from an offline Windows image

1. At an elevated command prompt, mount the offline Windows image:

```
Dism /Mount-Image /ImageFile:C:\test\images\install.wim /Name:"Windows 10 Home"  
/MountDir:C:\test\offline
```

2. Remove a specific driver from the image. Multiple drivers can also be removed on one command line.

```
Dism /Image:C:\test\offline /Remove-Driver /Driver:OEM1.inf /Driver:OEM2.inf
```

#### WARNING

Removing a boot-critical driver package can make the offline Windows image unbootable. For more information, see [DISM Driver Servicing Command-Line Options](#).

3. Commit the changes and unmount the image.

```
Dism /Unmount-Image /MountDir:C:\test\offline /Commit
```

# Add drivers to an offline Windows image by using an unattended answer file

1. Gather the device driver .inf files that you intend to install on the Windows image.

## NOTE

All drivers in the directory and subdirectories that are referenced in the answer file are added to the image. You should manage the answer file and these directories carefully to address concerns about increasing the size of the image with unnecessary driver packages.

2. Use Windows System Image Manager (Windows SIM) to create an answer file that contains the paths to the device drivers that you want to install.

- Add the `Microsoft-Windows-PnpCustomizationsNonWinPE\DriverPaths\PathAndCredentials\Credentials` component to your answer file in the **offlineServicing** configuration pass.

For each location that you intend to access, add a separate **PathAndCredentials** list item by right-clicking on **DriverPaths** in the *Answer File* pane and clicking **Insert New PathAndCredentials**.

See [Configure components and settings in an answer file](#) for information on how to modify an answer file.

3. For each path in `Microsoft-Windows-PnpCustomizationsNonWinPE`, specify the path to the device driver and the credentials that are used to access the file, if the file is on a network share.

## NOTE

When you include multiple device driver paths by adding multiple **PathAndCredentials** list items, you must increment the value of **Key** for each path. For example, you can add two separate driver paths where the value of **Key** for the first path is equal to **1** and the value of **Key** for the second path is equal to **2**.

4. Save the answer file and exit Windows SIM. The answer file must resemble the following sample.

```
<?xml version="1.0" ?><unattend xmlns="urn:schemas-microsoft-com:asm.v3"
xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State">
  <settings pass="offlineServicing">
    <component name="Microsoft-Windows-PnpCustomizationsNonWinPE" processorArchitecture="x86"
publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS">
      <DriverPaths>
        <PathAndCredentials wcm:keyValue="1">
          <Path>\networkshare\share\drivers</Path>
          <Credentials>
            <Domain>Fabrikam</Domain>
            <Username>MyUserName</Username>
            <Password>MyPassword</Password>
          </Credentials>
        </PathAndCredentials>
      </DriverPaths>
    </component>
  </settings>
</unattend>
```

5. Mount the Windows image that you intend to install the drivers to by using DISM:

```
Dism /Mount-Image /ImageFile:C:\test\images\install.wim /Index:1 /MountDir:C:\test\offline
```

If you're working with a VHD or FFU, specify `/Index:1`.

6. Apply the answer file to the mounted Windows image:

```
DISM /Image:C:\test\offline /Apply-Unattend:C:\test\answerfiles\myunattend.xml
```

For more information about how to apply an answer file, see [DISM Unattended Servicing Command-Line Options](#).

The .inf files referenced in the path in the answer file are added to the Windows image.

7. Check to see if the driver was added. Drivers added to the Windows image are named Oem\*.inf. This guarantees unique naming for newly added drivers. For example, the files MyDriver1.inf and MyDriver2.inf are renamed Oem0.inf and Oem1.inf.

For example, type:

```
Dism /Image:C:\test\offline /Get-Drivers
```

8. Unmount the .wim file and commit the changes. For example, type:

```
Dism /Unmount-Image /MountDir:C:\test\offline /Commit
```

If you need drivers for WinPE to see the local hard disk drive or a network, you must use the **windowsPE** configuration pass of an answer file to add drivers to the WinPE driver store and to reflect boot-critical drivers required by WinPE. For more information, see [Add Device Drivers to Windows During Windows Setup](#).

## Related topics

[Device Drivers and Deployment Overview](#)

[Add Device Drivers to Windows During Windows Setup](#)

[DISM - Deployment Image Servicing and Management Technical Reference for Windows](#)

# Add Device Drivers to Windows During Windows Setup

1/18/2019 • 2 minutes to read

To install Windows on some hardware designs, you may need to add device drivers to Windows Setup. You can add drivers to Windows Setup by using an answer file that specifies the path to the driver files. To do this in new installations, you add the Microsoft-Windows-PnpCustomizationWinPE component during the [windowsPE](#) configuration pass, add the driver paths, and then specify the answer file.

You can also modify existing images and add and remove drivers. You can service offline images in several ways. For example, you can add the Microsoft-Windows-PnpCustomizationsNonWinPE component during the [offlineServicing](#) configuration pass, add or remove the driver paths, and then specify the name of the answer file. For more information about how to modify drivers on an offline Windows image by using an answer file, and also other methods of adding drivers to and removing drivers from an existing image, see [Add and Remove Drivers to an Offline Windows Image](#).

## Add Drivers to New Installations (windowsPE)

For new installations, you add drivers during the [windowsPE](#) configuration pass.

This method initializes Windows Preinstallation Environment (Windows PE) and processes Windows PE settings from the answer file, as follows:

1. Windows stages the Windows PE drivers in the RAM driver store.. Windows loads boot-critical drivers that Windows PE requires to access the local disk and network. When you right-click **DevicePaths** and select **Insert New PathAndCredentials into Windows PE**, Windows PE processes other Windows PE customizations that the answer file specifies.
2. The Windows Setup process applies the Windows image. Boot-critical drivers appear on the Windows image before Setup installs that image. Other drivers that you added to the Windows PE driver store appear in the Windows image driver store. When Windows Setup processes the [offlineServicing](#) pass, Windows Setup also adds any drivers that the driver path specifies to the Windows image driver store.

### To add a device driver during the windowsPE pass

1. Use Windows System Image Manager (Windows SIM) to create an answer file that contains the paths to the device drivers that you intend to install.
  2. Add the **Microsoft-Windows-PnpCustomizationsWinPE** component to your answer file in the [windowsPE](#) configuration pass.
  3. Expand the **Microsoft-Windows-PnpCustomizationsWinPE** node in the answer file. Right-click **DevicePaths**, and then select **Insert New PathAndCredentials**.
- A new **PathAndCredentials** list item appears.
4. For each location that you access, add a separate **PathAndCredentials** list item.

You can include multiple device driver paths by adding multiple **PathAndCredentials** list items. If you add multiple list items, you must increment the **Key** value for each path. For example, if you add two separate driver paths, the first path uses the **Key** value of **1**, and the second path uses the **Key** value of **2**.

- Save the answer file, and then close Windows SIM. The answer file must resemble the following example:

```
<?xml version="1.0" encoding="utf-8" ?>
<unattend xmlns="urn:schemas-microsoft-com:unattend">
    <settings pass="windowsPE">
        <component name="Microsoft-Windows-PnpCustomizationsWinPE" processorArchitecture="x86"
publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS"
xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <DriverPaths>
                <PathAndCredentials wcm:keyValue="1" wcm:action="add">
                    <Credentials>
                        <Domain>Fabrikam</Domain>
                        <Password>MyPassword</Password>
                        <Username>MyUserName</Username>
                    </Credentials>
                    <Path>\\server\share\drivers</Path>
                </PathAndCredentials>
            </DriverPaths>
        </component>
    </settings>
</unattend>
```

- Boot to Windows PE.

- At a command prompt, run Windows Setup. Specify the name of the answer file. For example:

```
Setup /unattend:C:\unattend.xml
```

Windows Setup adds the device drivers in the \\server\\share\\drivers path to the system during the setup process.

For more information about drivers, see [Device Drivers and Deployment Overview](#) and [Add a Driver Online in Audit Mode](#). For more information about Windows components, see [Unattended Windows Setup Reference](#).

## Related topics

[Windows Setup Technical Reference](#)

[Boot from a DVD](#)

[Deploy a Custom Image](#)

[Boot Windows to Audit Mode or OOBE](#)

[Use a Configuration Set with Windows Setup](#)

[Add a Custom Script to Windows Setup](#)

# Configuration and settings

1/18/2019 • 2 minutes to read

This section covers various configuration options when deploying Windows.

## In this section

TOPIC	DESCRIPTION
<a href="#">Customize the Default User Profile by Using CopyProfile</a>	Use copyprofile to copy settings from a user profile to the default user
<a href="#">Work with Product Keys and Activation</a>	Learn about product keys and activation
<a href="#">High DPI Support for IT Professionals</a>	Learn how to configure PCs so they can use High DPI

# Add updates to a Windows image

1/25/2019 • 2 minutes to read

For many customizations, like adding .inf-style drivers, Windows updates or upgrading the edition, you can mount and edit the Windows image. Mounting an image maps the contents of a file to a temporary location where you can edit the files or use DISM to perform common deployment tasks.

## Notes

- **Add languages before major updates.** Major updates include hotfixes, general distribution releases, or service packs. If you add a language later, you'll need to re-add the updates.
- **Add major updates before apps.** These apps include universal Windows apps and desktop applications. If you add an update later, you'll need to re-add the apps.
- **For major updates, update the recovery image too:** These may include hotfixes, general distribution releases, service packs, or other pre-release updates. We'll show you how to update these later in [Lab 12: Update the recovery image](#).
- **If a Servicing Stack Update (SSU) is available, you must install it** before applying the most recent General Distribution Release (GDR) or any future GDRs. See [Windows 10 update history](#) to see the most recent GDR.

## Add a Windows update package to an image

1. Get a Windows update package. For example, get the latest cumulative update listed in [Windows 10 update history](#) from the [Microsoft Update catalog](#). If the update has any prerequisite updates, get those too.

To learn more, see <https://myoem.microsoft.com/oem/myoem/en/product/winemb/pages/comm-ms-updt-ctlg-trnstn.aspx>.

2. Add the updates to a [mounted](#) or online image. For packages with dependencies, either install the packages in order, or add them all at once using a single DISM command.

**Example:** adding a cumulative update to a mounted image:

```
Dism /Add-Package /Image:"C:\mount\windows" /PackagePath="windows10.0-kb4456655-x64_fca3f0c885da48efc6f9699b0c1eaf424e779434.msu" /LogPath=C:\mount\dism.log
```

**Example:** adding multiple updates:

```
Dism /Add-Package /Image:"C:\mount\windows" /PackagePath="C:\WindowsUpdates\windows10.0-kb00001-x64.msu" /PackagePath="C:\WindowsUpdates\windows10.0-kb00002-x64.msu" /PackagePath="C:\WindowsUpdates\windows10.0-kb00003-x64.msu" /LogPath=C:\mount\dism.log
```

3. Verify that the packages appear correctly:

```
Dism /Get-Packages /image:<path_to_image>
```

Review the resulting list of packages and verify that the list contains the package. For example:

Package Identity : Package\_for\_RollupFix~31bf3856ad364e35~amd64~~15063.250.1.1

State : Installed

Release Type : Security Update

Install Time : 04/29/2017 6:26 PM

The operation completed successfully.

#### **NOTE**

Each package will usually be a new KB, and will increase the build revision number of Windows on the device. The revision number of windows a device can be found in the following registry key:

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\UBR` .

4. If you're done updating your image, you can unmount it, committing changes.

# Customize the Default User Profile by Using CopyProfile

1/18/2019 • 5 minutes to read

You can use the `CopyProfile` setting to customize a user profile and then copy that profile to the default user profile. Windows uses the default user profile as a template to assign a profile to each new user. By customizing the default user profile, you can configure settings for all user accounts that are created on the computer. By using `CopyProfile`, you can customize installed applications, drivers, desktop backgrounds, internet explorer settings, and other configurations. Note that some settings are not preserved by using `CopyProfile`.

## NOTE

Using `CopyProfile` for Start menu customization isn't supported. Here are the ways to manage custom Start layouts in Windows 10:

- OEMs can use `layoutmodification.xml`. See [Customize the Start layout](#) for more information.
- IT pros can use the following resources learn about managing the Windows 10 Start Menu:
  - [Customize Windows 10 Start and taskbar with Group Policy](#)
  - [Windows 10 Start Layout Customization](#)

## Create an answer file with the `CopyProfile` setting

In [Windows SIM](#), create an answer file with `amd64_Microsoft-Windows-Shell-Setup_neutral\` added to **Pass 4 - Specialize**, and set `CopyProfile` to `True`. Save this answer file as `copyprofile.xml`. You'll use this answer file when you run `Sysprep`.

For more information, see [Best Practices for Authoring Answer Files](#) and [Unattended Windows Setup Reference Guide](#).

## Configure Default User Profile Settings

Configure user settings in audit mode and then generalize the Windows installation by using an answer file with `CopyProfile`. If you install Windows with another answer file, make sure that answer file doesn't have `CopyProfile` or any settings that create additional user accounts.

1. Install Windows on a reference computer and boot in audit mode. For more information, see [Boot Windows to Audit Mode or OOBE](#).

## IMPORTANT

Don't use a domain account, because the `CopyProfile` setting runs after the computer is removed from the domain when you run `Sysprep`. As a result, you'll lose any settings that you configured in a domain. If you change the default user profile and then join the computer to a domain, the customizations that you made to the default user profile will appear on new domain accounts.

2. Customize the built-in administrator account by installing applications, desktop shortcuts, and other settings.

**IMPORTANT**

There is a limit to the number of provisioned Windows Runtime-based apps that you can install. However you can create scripts to install additional non-provisioned apps. For more information, see [Sideloading Apps with DISM](#).

3. After your customizations are completed, insert the media that contains the CopyProfile answer file in the reference computer. For example, you can copy the answer file to a USB drive.
4. On the reference computer, open an elevated command prompt, and then type this command:

```
C:\Windows\System32\Sysprep\Sysprep /generalize /oobe /shutdown /unattend:F:\CopyProfile.xml
```

where *F* is the letter of the USB flash drive or other removable media. The **Sysprep** tool removes computer-specific information from the image, while preserving the user profile settings that you configured. For more information, see [Sysprep \(Generalize\) a Windows installation](#).

After the image is generalized, the computer shuts down, capture the image by booting to Windows PE and then capture the Windows installation by using DISM. For more information, see [WinPE: Create USB Bootable drive](#) and [Capture Images of Hard Disk Partitions Using DISM](#). After the image is captured, you can deploy it to a destination computer by using DISM. For more information, see [Apply Images Using DISM](#).

## Test the User Profile Customizations

After the customized image is deployed to a destination computer, you can test the user profile customizations. You can go through Out-Of-Box Experience (OOBE) to test the end user experience, or you can test the user customizations in audit mode.

**IMPORTANT**

Apps based on the Windows Runtime won't start in audit mode because audit mode uses the built-in administrator account. To run Windows Runtime-based apps you must modify a registry key before you can validate your Windows installation in audit mode.

### To test the user profile customizations after OOBE

1. Install Windows to a test computer.
2. After Windows is installed, go through OOBE and specify the computer name, user account name, and other items. After OOBE is complete, the Windows start screen appears.
3. Log onto the computer with the user account specified during OOBE and verify that your apps and customizations appear.

### To test the user profile customizations in audit mode

1. Boot to audit mode by using an answer file or by pressing Ctrl+Shift+F3 when OOBE starts. For more information, see [Boot Windows to Audit Mode or OOBE](#).
2. Verify that your customizations work as intended. To test Windows Runtime-based apps, modify the following registry key:
  - a. From an elevated command prompt, run Regedit.exe.
  - b. Browse to the  
**HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System**

`m\FilterAdministratorToken` registry key.

- c. Click **FilterAdministrationToken**, and then type **1** as the value data.
- d. Log off from the computer.
- e. Log back on to the computer, and start the Windows Runtime-based apps to verify that your customizations work as intended.
- f. After you complete validation of your Windows Runtime-based apps, reset the **FilterAdministratorToken** registry key to **0**.

## Troubleshooting CopyProfile

If the user profile settings aren't successfully copied:

1. Make sure that you set the `CopyProfile` setting only once during the deployment process.
2. When you customize user settings, only use only the built-in administrator account on the computer to avoid accidentally copying settings from the wrong profile.
3. Confirm that you didn't use a domain account.
4. Verify that there are no additional user accounts other than the built-in administrator account that you configured:
  - a. Click **Start**, and then type **Control Panel**.
  - b. Click **Control Panel**, and then click **Add or remove user accounts**.
  - c. Select any additional user account other than the built-in administrator account that you configured, and then delete it.

### NOTE

Delete all other user accounts on the computer before you customize the built-in administrator account.

5. Make sure that non-provisioned Windows Runtime-based apps that are stored in the tile layout are installed within two hours of user logon to preserve the tile layout on the **Start** screen, when apps are registered after the new user first logs on.
6. Some settings can be configured only by using the `CopyProfile` unattend setting, and other settings can be configured by using Group Policy.
  - Use Group Policy to configure settings that are reset by the new user logon process. You can also create scripts to define these user settings.  
-or-
  - Use the `CopyProfile` unattend setting instead. For more information, see [Unattended Windows Setup Reference](#).

## Related topics

[Sysprep \(System Preparation\) Overview](#)

[Sysprep Process Overview](#)

[Sysprep Command-Line Options](#)

# Work with Product Keys and Activation

1/18/2019 • 2 minutes to read

You can enter a product key during an automated installation of Windows by including it in your answer file.

You can also use product keys to select an image to install during an automated Windows installation.

## Select Which Windows Edition to Install

To select a Windows edition to install, you can do one of the following:

- Install Windows manually, without an answer file. Windows Setup installs the default edition from the Windows product DVD.
- Install Windows with an answer file, and include a product key in Microsoft-Windows-Setup\ UserData\ ProductKey\ `key`. Each product key is specific to a Windows edition. Entering the product key in this setting does not activate Windows.
- Install Windows with an answer file, and then manually type in a product key during Windows Setup. The product key selects a Windows edition to install.

**Warning** If you have multiple Windows images with the same Windows edition that are stored in the same Windows image file (.wim), you can use the setting: Microsoft-Windows-Setup\ ImageInstall\ OSImage\ InstallFrom\ `MetaData` to differentiate between them. You must still provide a product key using one of the methods listed in the previous list.

For information about managing Windows product keys when changing the Windows image to a higher edition, see [Change the Windows Image to a Higher Edition Using DISM](#).

## Activate Windows

To automatically activate Windows by using a product key, you can do one of the following:

- Use the Microsoft-Windows-Shell-Setup\ `ProductKey` unattend setting. You can use either a single-use product key or a Volume License Multiple Activation Key. For more information, see the [Volume Activation Planning Guide](#).

The product key used to activate Windows must match the Windows edition that you install. If you use a product key to select a Windows edition, we recommend using the same key to activate Windows, so that the edition you install is the same as the edition that you activate.

- Original Equipment Manufacturers (OEMs) can use OEM-specific activation tools.

**Warning** In most Windows deployment scenarios, you no longer have to use the `SkipRearm` answer file setting to reset the Windows Product Activation clock when you run the **Sysprep** command multiple times on a computer. The `SkipRearm` setting is used to specify the Windows licensing state. If you specify a retail product key or volume license product key, Windows is automatically activated. You can run the **Sysprep** command up to 8 additional times on a single Windows image. After running Sysprep 8 times on a Windows image, you must recreate your Windows image. For more information about Windows components and settings that you can add to an answer file, see the [Unattended Windows Setup Reference](#).

## Related topics

[Windows Deployment Options](#)

[How Configuration Passes Work](#)

[Sysprep \(Generalize\) a Windows installation](#)

[Change the Windows Image to a Higher Edition Using DISM](#)

# Change the Windows Image to a Higher Edition Using DISM

1/18/2019 • 2 minutes to read

You can use the Windows® edition-servicing commands to change one edition of Windows to a higher edition of Windows. The edition packages for each potential target edition are staged in the Windows image. This is referred to as an edition-family image. You can use the command-line options to list potential target editions. Because the target editions are staged, you can service a single image, and the updates will be applied appropriately to each edition in the image.

You need a product key to change the Windows edition online. Offline changes do not require a product key. If you change the image to a higher edition using offline servicing, you can add the product key by using one of the following methods:

- Enter the product key during the out-of-box experience (OOBE).
- Use an unattended answer file to enter the product key during the **specialize** configuration pass.
- Use Deployment Image Servicing and Management (DISM) and the Windows edition-servicing command-line option **/Set-ProductKey** after you set the edition offline.

For more information about product keys, see [Work with Product Keys and Activation](#).

## Find and Change Current Edition of Windows

You can find the edition of Windows your image is currently set to by mounting the image and running DISM commands on the mounted image.

### To find the current edition

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. At the command prompt, type the following command to retrieve the name or index number for the image that you want to modify.

```
Dism /Get-ImageInfo /ImageFile:C:\test\images
```

3. Type the following command to mount the offline Windows image.

```
Dism /Mount-Image /ImageFile:C:\test\images /Index:1 /MountDir:C:\test\offline
```

An index or name value is required for most operations that specify an image file.

4. Type the following command to find the edition of Windows your image is currently set to.

```
Dism /Image:C:\test\offline /Get-CurrentEdition
```

Note which edition of Windows your image is currently set to. If the image has already been changed to a higher edition you should not change it again. Use the lowest edition as a starting point.

5. Unmount the image or continue with the next procedure. To unmount your image, type the following command.

```
Dism /Unmount-Image /MountDir:C:\test\offline /Commit
```

## To change to a higher edition of Windows

1. Type the following command to mount the offline Windows image (if it is not already mounted).

```
Dism /Mount-Image /ImageFile:C:\test\images /Name:<Image_name> /MountDir:C:\test\offline
```

2. Type the following command to find the editions of Windows that you can change your image to.

```
Dism /Image:C:\test\offline /Get-TargetEditions
```

Note the edition-ID for the edition you want to change to.

**Note** You cannot set a Windows image to a lower edition. The lowest edition will not appear when you run the **/Get-TargetEditions** option. You should not use this procedure on an image that has already been changed to a higher edition.

3. Type the following command specifying the edition-ID to change the Windows image to a higher edition.

```
Dism /Image:C:\test\offline /Set-Edition:Professional
```

4. Type the following command to unmount the image and commit your changes.

```
Dism /Unmount-Image /MountDir:C:\test\offline /Commit
```

## Related topics

[Understanding Servicing Strategies](#)

[DISM Windows Edition-Servicing Command-Line Options](#)

[DISM - Deployment Image Servicing and Management Technical Reference for Windows](#)

# High DPI Support for IT Professionals

1/18/2019 • 2 minutes to read

Windows 8.1 has new features that improve the end user's experience with premium high density display panels. When using Windows 8.1, activities such as projecting to an external display and desktop scaling on the primary display are significantly improved from previous Windows releases. You will see the most benefit from these features when you are using a dense display, such as a 2560x1440 display with 225 DPI and 200% desktop scaling. This topic explains what is distinctive about these premium display parts, and what Windows 8.1 does to provide better support for them. It also explains some potential issues—including display fuzziness or blurriness—that might impact some users with lower density displays at 125% scaling, and how enterprise IT professionals can address them in Windows 8.1.

## In this section:

- [High DPI and Windows 8.1](#)
- [Fixing blurry text in Windows 8.1 for IT Professionals](#)
- [High DPI projection and multi-monitor configurations](#)
- [DPI-related APIs and registry settings](#)

# High DPI

1/18/2019 • 4 minutes to read

This topic introduces the key concepts of DPI and display scaling.

## In this topic:

- [Key concepts](#)
- [What's new about DPI](#)

## Key concepts

### What is DPI

Dots per inch (DPI) is the physical measurement of number of pixels in a linear inch of a display. DPI is a function of display resolution and size; a higher resolution or a smaller size will lead to higher DPI, and a lower resolution or a larger size will lead to lower DPI. When a display has a higher DPI, pixels are smaller and closer together, so that the user interface (UI) and other displayed content appears smaller than intended.

### What are Windows scale factors

Windows ensures that everything appears on the screen at a usable and consistent size by instructing applications (including the Windows desktop shell) to resize their content by a scale factor. This number depends on the display DPI as well as other factors that impact the user's perception of the display. Almost all desktop displays and most current laptop displays are in the range of 95-110 DPI; for these devices, no scaling is required, and Windows sets a scale factor of 100%. However, there are a number of new devices, particularly in the premium laptop and tablet markets, which have higher displays with over 200 DPI. For these devices, Windows sets higher scale factors to ensure that the user experience is comfortably viewable.

### Why this matters to users

Users typically spend hours reading and working on Windows devices, so it is important to ensure that the device they are looking at is optimized for their comfort. Therefore, it is important for Windows to present the content in the most readable way so that eye fatigue is reduced, and productivity is not impacted. As display technology improves, this can be delivered in a combination of higher DPI displays and better scaling in Windows. Windows 8 provides features that automatically adjust the default scaling to better match newer, more dense DPI displays.

### Why this matters to enterprises

As Windows devices improve, high density displays will become increasingly common in enterprise environments. Enterprises are also moving towards a more mobile workforce that use laptops in meetings to project, docking solutions when at the desk. To ensure optimum productivity, enterprise users should not need to manage how their screens lock when they project, or how their docking solutions present their workspace when they sit down at a desk. Windows does this automatically for most users, but there remain some edge cases which IT professionals in enterprise environments might need to help support. This topic describes how Windows automatically does the right thing in most instances, and where IT might need to step in and help the user out.

## What's new about DPI

### Display hardware market changes

With the advent of higher DPI displays, Windows devices will routinely feature DPIs that are much higher than what has been previously available. Instead of laptops with 13" and 1366x768 resolutions, you will see screen resolutions up to 3200x1800 at 13". For these laptops to be usable, Windows scaling has to move from 100%

(13.3" 1366x768) to 125% (13.3" 1600x900), 150% (10.6" 1920x1080) or 200% (13.3" 2560x1440). Before Windows 8, only 100%, 125% and 150% were automatically set to match the display; in Windows 8, 200% support has been added.

## Windows 8.1 changes

Windows 8 includes a number of feature changes that are specific for high DPI, as shown in *Table 1 Windows 8.1 DPI Changes*:

**Table 1 Windows 8.1 DPI Changes**

FEATURE	WINDOWS 8	WINDOWS 8.1
Support for 200% scaling	No	Yes
Per-monitor DPI	No	Yes
Scaling of existing DPI-aware applications	No	Yes
Per-monitor DPI aware applications	No	Yes
Viewing distance incorporated in default DPI calculation	No	Yes
Logoff free DPI change	No	Yes
Per-monitor DPI aware Internet Explorer	No	Yes
Auto-DPI configuration of Remote Desktop	No	Yes

The first two of the above features have the biggest impact on Windows 8 usability. In more detail:

- Improved 200% scaling support:** Windows 8.1 identifies high DPI display devices on a dynamic basis and natively supports up to 200% scale factors. Windows 8 only identified a high DPI display during first boot, and only supported up to 150% scaling without user customization. This feature ensures that users who buy premium laptops with high DPI displays will automatically receive the 200% scaling required to make content easily visible.
- Per-monitor DPI:** Windows 8.1 sets different scale factors for different displays, and can scale content appropriately. Windows 8 only sets a single scale factor that is applied to all displays. This feature ensures that users with High DPI devices (that is, 150% and 200% scaling laptops) who project or dock their devices with conventional 100% scaling projectors and desktop monitors display properly sized content on those screens.

## How these changes impact enterprise users

For the users on laptops with 100% scaling, the Windows 8.1 feature changes have no impact. For users who acquire new devices that have high DPI, Windows 8.1 provides a significant benefit.

It is possible that some users will acquire devices that fall in-between, featuring Windows scaling of 125%. These devices can require the user or the IT professional to configure them correctly or update/tweak applications to improve usability. The troubleshooting section of this topic can help IT professionals identify these systems, these applications, and undertake the right mitigation tactics.

## Related topics

[High DPI Support for IT Professionals](#)

# Fixing blurry text in Windows for IT Professionals

1/18/2019 • 5 minutes to read

Windows desktop apps fall broadly into two classes: apps that are *DPI-aware* and those that are not. DPI-aware apps actively let Windows know during application launch that they are capable of scaling themselves to work well on a high DPI display. These apps include: Internet Explorer, Office, Firefox, and .NET 2.0+ (including WPF) apps. These apps generally work well across a wide range of scale factors. Therefore, if your enterprise line of business apps are also DPI-aware, your users should not have a problem with any Windows displays or scale factors.

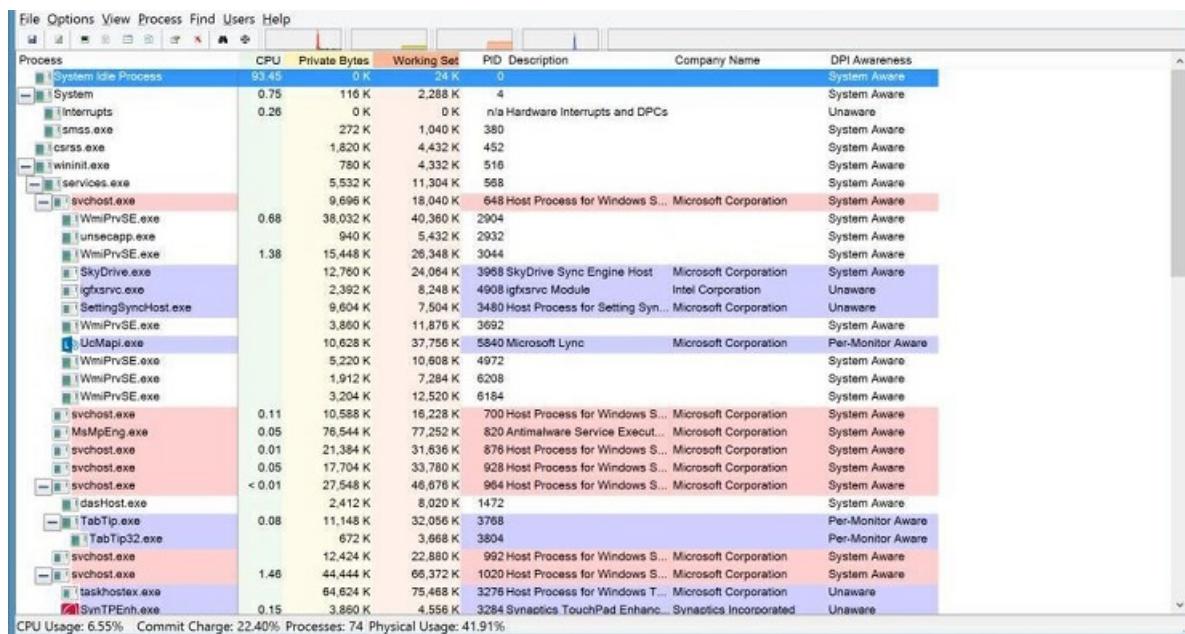
However, if an application is not DPI aware, and is running on a high DPI display, Windows scales the app by applying bitmap scaling to the application output. This ensures that the application is the correct size on a high DPI display. In most instances this will result in crisp and usable applications, but in some instances, the result is less crisp and might have a slightly fuzzy or blurry appearance because of the bitmap scaling.

## In this topic:

- [How to tell if an application is not DPI-aware](#)
- [What you can do about apps that aren't DPI-aware](#)
- [Tell Windows not to scale an app that's not DPI-aware](#)

## How to tell if an application is not DPI-aware

Use the [Process Explorer tool](#) to determine if an app is DPI-aware. *Figure 1 Process Explorer* shows this utility in use, with the column for **DPI Awareness** enabled. (By default, process explorer does not show the **DPI Awareness** column. To turn this column on, click the **View** menu, click **Select Columns**, check the box for **DPI Awareness**, and click **OK**.) The column titled **DPI Awareness** tells you whether a particular process is aware of DPI or not.



The screenshot shows the Microsoft Process Explorer interface with the 'DPI Awareness' column enabled. The table lists various processes along with their CPU usage, private bytes, working set, and the 'DPI Awareness' status. The 'DPI Awareness' column uses color coding: light blue for 'System Aware', red for 'Unaware', and purple for 'Per-Monitor Aware'. Processes shown include System Idle Process, System, Interrupts, smss.exe, csrss.exe, wininit.exe, services.exe, svchost.exe, WmiPrvSE.exe, unsecapp.exe, WmiPrvSE.exe, SkyDrive.exe, lgthsrvc.exe, SettingSyncHost.exe, WmiPrvSE.exe, UsMapi.exe, WmiPrvSE.exe, WmiPrvSE.exe, WmiPrvSE.exe, svchost.exe, dashHost.exe, TabTip.exe, TabTip32.exe, svchost.exe, svchost.exe, taskhostex.exe, and SynTPEnh.exe. The 'DPI Awareness' column highlights several processes as Unaware (red), such as the host processes for Windows S., Microsoft Corporation, and the Microsoft Lync process.

Process	CPU	Private Bytes	Working Set	Description	Company Name	DPI Awareness
System Idle Process	93.45	0 K	24 K	0		System Aware
System	0.75	116 K	2,288 K	4		System Aware
Interrupts	0.26	0 K	0 K	n/a Hardware Interrupts and DPCs		Unaware
smss.exe		272 K	1,040 K	380		System Aware
csrss.exe		1,820 K	4,432 K	452		System Aware
wininit.exe		780 K	4,332 K	516		System Aware
services.exe		5,532 K	11,304 K	568		System Aware
svchost.exe		9,696 K	18,040 K	648 Host Process for Windows S..., Microsoft Corporation		System Aware
WmiPrvSE.exe	0.68	38,032 K	40,360 K	2904		System Aware
unsecapp.exe		940 K	5,432 K	2932		System Aware
WmiPrvSE.exe	1.38	15,448 K	26,348 K	3044		System Aware
SkyDrive.exe		12,760 K	24,064 K	3988 SkyDrive Sync Engine Host	Microsoft Corporation	System Aware
lgthsrvc.exe		2,392 K	8,248 K	4908 lgthsrvc Module	Intel Corporation	Unaware
SettingSyncHost.exe		9,604 K	7,504 K	3480 Host Process for Setting Syn...	Microsoft Corporation	Unaware
WmiPrvSE.exe		3,860 K	11,876 K	3692		System Aware
UsMapi.exe		10,628 K	37,756 K	5840 Microsoft Lync	Microsoft Corporation	Per-Monitor Aware
WmiPrvSE.exe		5,220 K	10,608 K	4972		System Aware
WmiPrvSE.exe		1,912 K	7,284 K	6208		System Aware
WmiPrvSE.exe		3,204 K	12,520 K	6184		System Aware
svchost.exe	0.11	10,588 K	16,228 K	700 Host Process for Windows S..., Microsoft Corporation		System Aware
MalMpEng.exe	0.05	76,544 K	77,252 K	820 Antimalware Service Execut...	Microsoft Corporation	System Aware
svchost.exe	0.01	21,384 K	31,836 K	876 Host Process for Windows S...	Microsoft Corporation	System Aware
svchost.exe	0.05	17,704 K	33,780 K	928 Host Process for Windows S...	Microsoft Corporation	System Aware
dashHost.exe	< 0.01	27,548 K	46,876 K	964 Host Process for Windows S...	Microsoft Corporation	System Aware
dashHost.exe		2,412 K	8,020 K	1472		System Aware
TabTip.exe	0.08	11,148 K	32,056 K	3788		Per-Monitor Aware
TabTip32.exe		672 K	3,668 K	3804		Per-Monitor Aware
svchost.exe		12,424 K	22,880 K	992 Host Process for Windows S...	Microsoft Corporation	System Aware
svchost.exe	1.46	44,444 K	66,372 K	1020 Host Process for Windows S...	Microsoft Corporation	System Aware
taskhostex.exe		64,624 K	75,468 K	3276 Host Process for Windows T...	Microsoft Corporation	Unaware
SynTPEnh.exe	0.15	3,860 K	4,556 K	3284 Synaptics TouchPad Enhanc...	Synaptics Incorporated	Unaware

**Figure 1 Process Explorer**

Windows distinguishes between three classes of applications.

**Table 1 DPI Awareness Apps**

DPI AWARENESS	EXAMPLES	BEHAVIOR
Unaware	<b>Mmc.exe</b> (Microsoft Management Console and its plugins)	Windows bitmap scales the application to any high DPI displays that are attached to the system; can be fuzzy at 125% and 150% scale factors.
System-aware	Office apps	Application scales itself at launch to the system DPI (usually the same as the primary display DPI); Windows scales the app to any displays that do not match this.
Per-monitor-aware	Internet Explorer 11	Application dynamically scales itself to the display DPI.

## What you can do about apps that aren't DPI-aware

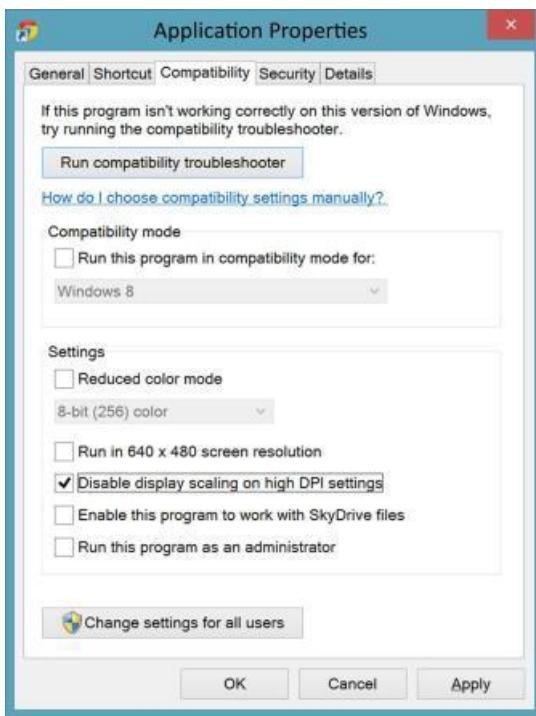
### Run the latest version of the app or ask the application vendor to update their app to be DPI aware

Microsoft recommends that all applications become DPI-aware. It is possible that newer versions of your applications are already DPI-aware. If they are not, you can ask your app vendor to update their app to be DPI aware. Microsoft provides developer resources that can help them update their app, including the following:

- [Making Your Desktop Apps Shine on High- DPI Displays \(BUILD 2013 presentation\)](#)
- [Writing DPI-Aware Desktop Applications in Windows](#)
- [Dynamic DPI sample](#)

### Tell Windows not to scale an app that's not DPI-aware

In the cases where users cannot deal with the bitmap scaling of apps that aren't DPI-aware (for example, 125% scaling and fuzzy applications), individual Windows desktop applications can be shimmed to not be scaled. Users can do this by using the **Compatibility** tab of the application's **Properties** UI. For example, *Figure 2 Application Properties* shows how a user can disable bitmap scaling:



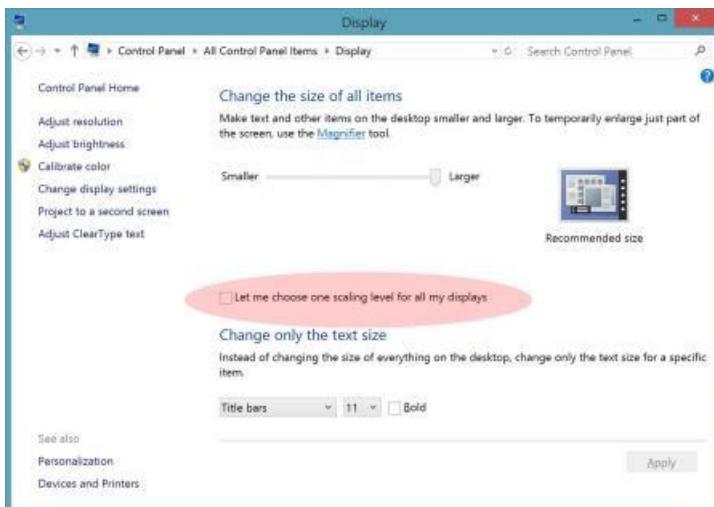
**Figure 2 Application Properties**

You can manage bulk-shimming of applications by using the Compatadmin tool, which is available in the Application Compatibility Toolkit that is included in the Windows Assessment and Deployment Kit (ADK). You can download the Windows ADK from [Windows Assessment and Deployment Kit \(ADK\)](#). For more information about how to use the Compatadmin tool, see [How to use the Compatibility Administrator utility in Windows](#).

**Important** Disabling display scaling can result in content that is too small to be read or interacted with reliably; it can also produce visual artifacts such as clipped or overlapped content. These issues depend on details of how the app was written. Consequently, we recommend only changing this setting if absolutely required. This shim should not be applied to apps that do not require it, or to devices that do not require it.

### Use Windows 8 DPI scaling (not generally recommended)

Windows includes a Windows 8-compatibility scaling mode that can be deployed to address all visual blurring issues with certain displays. Note that using the compatibility mode turns off all the benefits of the Windows DPI features. This method should only be used as a last resort, if the enterprise environment includes too many apps that aren't DPI-aware to be mitigated by applying application shimming. Users can access this mode in the DPI CPL UI by checking the box that says **Let me choose one scaling level for all my displays**:



**Figure 3 Scaling Level Option**

This setting can also be applied during deployment if you have many specific apps that need remediation and you

plan a large scale roll-out to low or mid-density displays. You can customize your image in Audit mode before deployment. See [Audit Mode Overview](#). See also the next section that explains how to programmatically perform device detection and registry customization.

### Understanding high DPI, display types, and Windows scaling

Windows scales apps that aren't DPI-aware dynamically by resizing the bitmap generated by the application.

Bitmap scaling works best when scaled at integer multiples (for example, 1x, 2x, 3x), but can have visual artifacts that are often perceived as blurry/fuzzy at non-integer multiples (for example, 125%, 150%).

Windows supports a full spectrum of screen sizes, resolutions, and therefore DPI. There will be some DPI ranges that result in less than optimal Windows scaling for apps that aren't DPI-aware.

*Table 2 Scaling Values* describes the possible issues that users can encounter at different Windows scaling values:

**Table 2 Scaling Values**

SCALE FACTOR	100% MAINSTREAM	125% VALUE	150% PREMIUM	200% PREMIUM
Scaling benefit	N/A	Small size improvement	Significant size improvement	Critical size improvement
Bitmap scaling of unaware apps	N/A	Most noticeable fuzziness	Less noticeable fuzziness	Clear and crisp
Scaling of aware apps	N/A	Clear and crisp	Clear and crisp	Clear and crisp

As shown in the preceding table, most of the issues manifest at the 125% scaling ratio. For this reason, any mitigation should target apps that aren't DPI-aware on 125% scaling systems only.

For information about how to identify 125% systems or how to revert to Windows 8 scaling behavior for a 125% system, see [DPI-related APIs and registry settings](#).

## Related topics

[High DPI Support for IT Professionals](#)

# High DPI projection and multi-monitor configurations

1/18/2019 • 2 minutes to read

Many enterprise users use secondary displays for such purposes as docking, projection, or extending their desktop to a secondary display.

These scenarios do not impact the guidance for 150% and 200% devices, but for users with 125% display devices who also use a desktop docking station or secondary monitor, we recommend the Windows 8 compatibility mode that is described in [Fixing blurry text in Windows 8.1 for IT Professionals](#). Additional guidance about compatible devices and projectors is provided in this topic.

## Projection experiences

Windows 8.1 has optimized support for projection experiences. In previous versions of Windows, the user of a high DPI device might see content that was too big on the low DPI projector, making it difficult to get all the appropriate content on screen for presentation purposes. There are two projection modes: *Duplicate* and *Extend*. This section describes how Windows supports each of these modes.

### **Duplicate mode (default for projection and typically used for projection)**

The default projection mode is called Duplicate mode. (Type **Win+P** at the keyboard to see a list of the four multi-monitor display modes: **PC screen only**, **Duplicate**, **Extend**, and **Second** screen only.) In Duplicate mode, the same content is presented on the laptop display as on the projector. This makes it easiest for the presenter to interact directly with the content being displayed on the screen, particularly with laptop or tablet that supports touch. In this mode, Windows will look at both displays, try to find the best common resolution, and then put both displays into that resolution. In Windows 8.1, if this resolution change has an impact on the display scale factor, Windows will then rescale based on the new scale factor, thereby ensuring the best projection experience.

### **Extend mode (typical for multi-monitor desktop scenarios)**

In the Extend mode, the projector is treated as a separate display from the primary display. This mode is typical for users using a multi-monitor setup or docking scenario. The user can drag or move content to the separate display by using the mouse or touchpad. This is not the default option but some users prefer this setting (to give just one example, because it allows the user to separate note taking from their presentation). In this mode, Windows 8.1 associates an appropriate scale factor for each display, and when the user moves content to the projector, Windows will rescale it appropriately, again ensuring the best projection experience.

### **What this means for the IT Professional**

For projection scenarios, per-monitor scaling is required to provide a usable projection experience for 150% and 200% displays. In some cases, users who have 125% devices might have issues with apps that aren't DPI-aware being fuzzier when projected. See [Fixing blurry text in Windows 8.1 for IT Professionals](#) for guidance on how to turn off per-app DPI scaling in these cases.

**Important** Projectors work best in duplicate mode if they support resolutions and video modes that are similar to the device that is projecting. For example, if the dominant portable devices in the enterprise have 1366x768 and 1920x1080 displays, the projectors that are used should support the same resolutions for the best duplicate mode experiences.

## Related topics

[High DPI Support for IT Professionals](#)

# DPI-related APIs and registry settings

1/18/2019 • 4 minutes to read

If you need to perform deployment customizations, the following sections explain the registry keys and system parameters that your post-installation scripts might need to access.

## In this topic:

- [Primary display native resolution](#)
- [Primary display DPI scale factor](#)
- [Scaling mode](#)
- [Scaling override in Windows 8.1 scaling mode](#)
- [System-wide scale factor in Windows 8 scaling mode](#)

## Primary display native resolution

*Table 1 Windows 8.1 Scaling Levels*, while by no means exhaustive, provides information on the Windows 8.1 scaling level for a number of common displays. **Panel DPI** indicates the physical pixel density of the panel, and **Scaling level** indicates the scale factor that will be used for this display.

**Table 1 Windows 8.1 Scaling Levels**

DISPLAY SIZE	DISPLAY RESOLUTION	HORIZONTAL (PIXELS)	VERTICAL (PIXELS)	PANEL DPI	SCALING LEVEL
10.6"	FHD	1920	1080	208	150%
10.6"	HD	1366	768	148	100%
11.6"	WUXGA	1920	1200	195	150%
11.6"	HD	1366	768	135	100%
13.3"	WUXGA	1920	1200	170	150%
13.3"	QHD	2560	1440	221	200%
13.3"	HD	1366	768	118	100%
15.4"	FHD	1920	1080	143	125%
15.6"	QHD+	3200	1800	235	200%
17"	FHD	1920	1080	130	125%
23"	QFHD (4K)	3840	2160	192	200%
24"	QHD	2560	1440	122	125%

To programmatically find this information for any device, you can write a utility program that reports back data. The native primary resolution is retrieved by calling the API [GetDeviceCaps\(\) function](#), using the hdc for the desktop and the HORZRES and VERZRES indices:

```
// Get desktop dc
desktopDc = GetDC(NULL);
// Get native resolution
horizontalResolution = GetDeviceCaps(desktopDc,HORZRES);
verticalResolution = GetDeviceCaps(desktopDc,VERZRES);
```

For more information about GetDC, see [GetDC\(\) function](#).

## Primary display DPI scale factor

Similarly, you can get the pixel density by using the LOGPIXELSX and LOGPIXELSY indices:

```
// Get desktop dc
desktopDc = GetDC(NULL);
// Get native resolution
horizontalDPI = GetDeviceCaps(desktopDc,LOGPIXELSX);
verticalDPI = GetDeviceCaps(desktopDc,LOGPIXELSY);
```

These results are returned in a coordinate system in which 96 corresponds to 100%, as shown in *Table 2 DPI Scale Factors*.

**Table 2** **DPI Scale Factors**

DPI	SCALE FACTOR
96	100
120	125
144	150
192	200

**Note** This API will return different results depending on the DPI awareness mode of your application. Configuring the awareness mode requires adding XML to the application manifest, as detailed below:

DPI AWARENESS MODE	MANIFEST SETTING	RETURNED VALUE
None	None	96 for all displays, regardless of the scale factor
System DPI Aware	<dpiAware>True</dpiAware>	The DPI of the primary display at the time the Windows session was started (when the user first logged in to Windows)

DPI AWARENESS MODE	MANIFEST SETTING	RETURNED VALUE
Per-Monitor DPI Aware	<dpiAware>True/PM</dpiAware>	The DPI of the primary display at the time the Windows session was started (when the user first logged in to Windows). To obtain the DPI of the display that the application is located on, use <a href="#">GetWindowDpi()</a> or <a href="#">GetDpiForMonitor()</a>

For more information about this manifest setting, see [SetProcessDPIAware function](#).

## Scaling mode

The **Control Panel\ Appearance and Personalization\Display** user interface (UI) includes a checkbox: **Let me choose one scaling level for all my displays**, which controls whether the system applies a single scale factor to all displays (as in Windows 8 and earlier versions of Windows), or different scale factors that take into account the pixel density of each display (the Windows 8.1 default). This checkbox configures the **HKCU\Control Panel\Desktop\Win8DpiScaling** registry key in Windows 8.1.

**Table 3 HKCU\Control Panel\Desktop\Win8DpiScaling Values**

KEY VALUE	MEANING
0	Different scale factors for each display: Windows 8.1 default. Content that is moved from one display to another will be the right size, but can be bitmap-scaled.
1	Same scale factor is applied to all displays: Windows 8 and earlier Windows versions behavior. Content that is moved from one display to another might be the wrong size.

## Scaling override in Windows 8.1 scaling mode

When the **Let me choose one scaling level for all my displays** checkbox is cleared and the system is running in the Windows 8.1 scaling mode, the user is provided with a slider that lets them override the current scale factors, from Smaller, to Medium, to Larger. This setting is configured in the **HKCU\Control Panel\Desktop\DesktopDPOVERRIDE** registry key.

**Table 4 HKCU\Control Panel\Desktop\DesktopDPOVERRIDE Values**

KEY VALUE	MEANING
<0	Reduce each display scale factor from the default by this value (for example, if the default was 150% scaling, -1 corresponds to 125%, -2 to 100%).
0	Use the default value for each display.
0>	Increase each display factor by this value (using the previous example, +1 corresponds to 200% scaling).

All display scale factors in this mode are constrained to be one of these four values: 100%, 125%, 150%, 200%. In addition, after scaling is applied, applications expect to have at least 720 effective lines of resolution (that is, the physical vertical resolution of the display divided by the scale factor); this can further limit the range of allowed display scale factors. *Table 5 Display Values* shows which values are allowed for different sized displays:

**Table 5 Display Values**

VERTICAL LINES	SUPPORTED SCALE FACTORS
<900	100%
≥ 900 and <1080	100%, 125%
≥1080 and <1440	100%, 125%, 150%
≥1440	100%, 125%, 150%, 200%

## System-wide scale factor in Windows 8 scaling mode

When the **Let me choose one scaling level for all my displays** checkbox is checked, the user can specify a scale factor that applies to all displays, regardless of each display's pixel density. By using the custom setting, the user can select values other than 100%, 125%, 150%, 200%, although they are limited to the range (100%-500%). This setting is configured in the **HKCU\Control Panel\Desktop\LogPixels** registry key.

**Table 6 HKCU\Control Panel\Desktop\LogPixels Values**

KEY VALUE	MEANING
96	100% scaling on every display
120	125% scaling on every display
144	150% scaling on every display
192	200% scaling on every display
<other>	<other> 96/100 scaling on every display

## Related topics

[Documentation for developing High DPI applications](#)

[High DPI Support for IT Professionals](#)

# Features

1/18/2019 • 2 minutes to read

This section covers how you can add and remove or enable and disable features from a Windows installation.

## In this section

TOPIC	DESCRIPTION
<a href="#">Features On Demand</a>	How to work with Features on Demand
<a href="#">Enable or Disable Windows Features Using DISM</a>	How to work with legacy Windows features
<a href="#">Configure a Trusted Image Identifier for Windows Defender</a>	How to add a trusted image identifier to Windows Defender
<a href="#">Configure Windows System Assessment Test Scores</a>	How to generate a WinSAT formal file prior to shipping it to the end-users
<a href="#">Add or Remove Packages Offline Using DISM</a>	How to work with (.cab) packages

# Features On Demand

5/15/2019 • 4 minutes to read

## Overview

Features on Demand (FODs) are Windows feature packages that can be added at any time. Common features include [language resources](#) like handwriting recognition or [other features](#) like the .NET Framework (.NetFx3). When Windows 10 or Windows Server needs a new feature, it can request the feature package from Windows Update.

You can also preinstall FODs so they're ready if a user needs them. FODs are distributed as .cab files on the Feature on Demand ISO and you can use DISM to add a FOD to a Windows image. If you're using the FOD ISO to preinstall FODs, make sure you're using the FOD ISO that matches your Windows image version.

### NOTE

Add language packs, FODs, and apps, in that order, prior to installing an update. If you add an update prior to adding language packs, FODs, and apps you'll have to reinstall the update.

## Types of Features on Demand

Starting with Windows 10, version 1809 and Windows Server 2019, Windows has two different types of Features on Demand:

- **FODs without satellite packages:** FODs with all language resources packaged into the same package. These FODs are distributed as a single .cab file.

They can be added using either DISM `/Add-Capability` or `/Add-Package`.

- **FODs with satellite packages:** Language-neutral FODs that have language and/or architecture resources in separate packages (satellites). When you install this type of FOD, only the packages that apply to the Windows image are installed, which reduces disk footprint. These FODs are distributed as a set of several .cab files, but are installed by specifying a single `/capabilityname`. These are new for Windows 10, version 1809.

They can only be added using `DISM /Add-Capability` (and not `/Add-Package`).

FODs with satellites require a well-formed FOD repository. This can either be the full FOD repository on the ISO, or a custom repository created with `DISM /export-source`. They cannot be added by pointing to a directory with a handful of FOD files hand-copied from the repository, because DISM requires additional metadata to make the right connections.

See [Available Features on demand](#) for more information about which FODs have satellites. Language FODs don't have satellites.

### TIP

While you can add non-satellite FODs using `/add-package`, we recommend adding all FODs with `/add-capability` because you can use it to add both satellite and non-satellite FODs.

## Adding or removing Features on Demand

### Using DISM /add-package to add or remove FODs

You can add non-satellite Features on demand to a Windows image with `DISM /add-package`. The FODs that you can add with `/add-package` include the language FODs, the FODs that come preinstalled with Windows, and the FODs that we recommend for preinstallation.

OEMs can generally add FODs to their Windows images using `DISM /add-package`, unless they need to preinstall a FOD that has satellites.

COMMAND	DESCRIPTION	EXAMPLE
<code>/add-package</code>	Adds a package, including a FOD .cab to an image. <code>/add-package</code> can only be used to add FODs that don't have satellite packages.	<code>DISM.exe /image:C:\mount\Windows /add-package /packagepath:E:\Microsoft-Windows-Holographic-Desktop-FOD-Package~31bf3856ad364e35~amd64~~.cab</code>
<code>/Get-Packages</code>	Get a list of all packages in an image.	<code>DISM /image:C:\Mount\Windows /get-packages</code>
<code>/Get-Packageinfo</code>	Get information of a package in an image.	<code>DISM /image:C:\mount\Windows /get-packageInfo /packagename:Microsoft-Windows-Xps-Xps-Viewer-Opt-Package~31bf3856ad364e35~amd64~~10.0.17134.1000</code>

COMMAND	DESCRIPTION	EXAMPLE
/Remove-Package	Removes a package from an image. Note: Don't remove a package that other packages depend on. For example, if you have the French handwriting and basic packages installed, don't remove the basic package.	DISM.exe /image:C:\mount\Windows /Remove-Package /PackageName:Microsoft.Windows.Calc.Demo~6595b6144cc

### Using DISM /add-capability to add or remove FODs

You can also install FODs with `DISM /add-Capability`. You have to use `/add-capability` to preinstall satellite FODs. If you're preinstalling a FOD onto an offline image, use the `/source` option to tell Windows where to find the FOD installation files. The `/source` could be a FOD repository or a mounted FOD ISO. You can use multiple `/source` arguments in a command.

If you're adding a FOD to an online image, `/add-capability` downloads features from Windows Update and adds them to the image. If you don't want to install from Windows Update, you can use `/LimitAccess`, which tells DISM to not check Windows Update or Windows Server Update Services for the capability source files.

See [DISM Capabilities Package Servicing Command-Line Options](#) for more information.

- Use the `/online` option to add the capability to your PC.
- Use `/Image:<mount path>` with the `/source` option to add the capability to a mounted Windows image.

COMMAND	DESCRIPTION	EXAMPLE
/add-capability	<p>Adds a capability to an image.</p> <p>For packages with dependencies this also pulls dependent packages. For example, if you add the Speech package, you'll also get the Text-to-speech and Basic packages in addition to Speech.</p> <p>DISM looks for source files in the following locations:</p> <ol style="list-style-type: none"> <li>1. If <b>/Source</b> is specified, DISM looks in the specified locations first.</li> <li>2. If <b>/Source</b> is not specified, or if the source files are not found in the specified locations, DISM checks the locations specified by group policy.</li> <li>3. If the files still aren't found, and if DISM is running against an online image, and if <b>/LimitAccess</b> is not specified, it looks for the files on Windows Update.</li> </ol>	<pre>DISM.exe /image:C:\mount\Windows /add-capability /CapabilityName:Language.Basic~~en-US~0.0.1.0 /Source:E:</pre> <pre>DISM.exe /Online /add-capability /CapabilityName:Language.Basic~~en-US~0.0.1.0</pre>
/Get-Capabilities	Get capabilities in the image.	<pre>DISM /image:C:\Mount\Windows /Get-Capabilities</pre>
/Get-CapabilityInfo	Get information of a capability in the image.	<pre>DISM /image:C:\mount\Windows /Get-CapabilityInfo /CapabilityName:Language.Basic~~en-US~0.0.1.0</pre>
/Remove-Capability	<p>Removes a capability from an image.</p> <p><b>Note:</b> You cannot remove a capability that other packages depend on. For example, if you have the French handwriting and basic capabilities installed, you can't remove the basic capability.</p>	<pre>DISM.exe /image:C:\mount\Windows /Remove-Capability /CapabilityName:Language.Basic~~en-US~0.0.1.0</pre>

To see all available DISM commands for capabilities, see [DISM Capabilities Package Servicing Command-Line Options](#).

## Related topics

- [Available Features on Demand](#)
- [Language and region Features on Demand](#)
- [Add languages to Windows](#)
- [DISM capabilities package servicing command-line options](#)

# Available Features on Demand

1/25/2019 • 13 minutes to read

The following Features on Demand are available for Windows 10 and Windows Server. Some FODs come preinstalled with Windows and are all you need to ship a Windows device. You can, however, add additional FODs to support additional hardware features or enable certain use cases.

You can use either `DISM /image:<path_to_image> /get-capabilities` or `DISM /online /get-capabilities` to see which Features on Demand are available in your image of Windows 10. To see how to add Features on Demand, see [Features on Demand](#).

To see available Features on Demand for languages and regions, see [Language and region Features on Demand](#)

## Preinstalled FODs

The following Features on Demand come preinstalled in a Windows image. Don't remove these FODs from the Windows image:

### Internet Explorer

Internet Explorer Features on Demand enable preinstallation of Internet Explorer. Internet Explorer will not be enabled on a device that doesn't include this Feature on Demand.

**Feature:** Internet Explorer 11

- **Capability name:** Browser.InternetExplorer~~~~0.0.11.0
- **CAB name:** Microsoft-Windows-InternetExplorer-Optional-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 1.66 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1703 and later

### Math Recognizer

This feature on demand adds the Math Input Panel, Control and Recognizer.

**Feature:** Math Recognizer

- **Capability name:** MathRecognizer~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-TabletPCMath-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 17.46 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1809 and later

### OneSync

This Feature on Demand is a mail, contacts, and calendar sync component. Not including this Feature on Demand on your Windows image will cause UWP apps such as Mail, People, and Calendar to not be able to properly sync.

**Feature:** Exchange ActiveSync and Internet Mail Sync Engine

- **Capability name:** OneCoreUAP.OneSync~~~~0.0.1.0
- **CAB name:** Microsoft-OneCore-ApplicationModel-Sync-Desktop-FOD-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 12.83 MB

- **Satellites:** None
- **Availability:** Windows 10, version 1709 and later

### OpenSSH client

The OpenSSH client Feature on Demand enables the use of OpenSSH on a Windows PC.

#### Feature: OpenSSH Client

- **Capability name:** OpenSSH.Client~~~~~0.0.1.0
- **CAB name:** OpenSSH-Client-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 5.28 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1709 and later

### Quick Assist

Microsoft Quick Assist allows users to share their screen with Microsoft support agents or other trusted people. This FOD is preinstalled, but not available on the FOD ISO.

#### Feature: Microsoft Quick Assist

- **Capability name:** App.Support.QuickAssist~~~~~0.0.1.0
- **CAB name:**
- **Install size:** 1.51 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1607 and later

## FODs that are not preinstalled, but may need to be preinstalled

The following FODs are not preinstalled in a Windows image, but we recommend preinstalling them to enable certain scenarios.

### .NET Framework

.NET framework adds support for 3rd party .NET3.x apps. If this Feature on Demand is not installed, .NET3.x apps won't work.

**Recommendation:** Preinstall these Features on Demand on devices that are preloaded with apps that require .NET3.x.

#### Feature: NetFx3

- **Capability name:** NetFX3~~~~~
- **CAB name:** Microsoft-Windows-NetFx3-OnDemand-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 301 MB
- **Satellites:** None
- All versions of Windows 10

### Mixed Reality

This Feature on Demand enables Mixed Reality (MR) devices to be used on a PC. If this Feature on Demand is not present, MR devices may not function properly.

**Note:** Make sure to add this feature on demand prior to adding an update.

**Recommendation:** Recommended for MR-Ready badged PCs, otherwise do not preinstall.

#### **NOTE**

The Mixed Reality Feature on Demand has a large installation size. This FOD also requires the installation of an additional data assets package, if an updated asset package is available. Updates to the Mixed Reality FOD are available via regular Windows LCUs. The data assets update package, when available, is a separate download from the [Microsoft Update Catalog](#).

#### **Feature:** Windows Mixed Reality

- **Capability name:** Analog.Holographic.Desktop~~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-Holographic-Desktop-FOD-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 1.93 GB
- **Satellites:** None
- **Availability:** Windows 10, version 1703 and later

#### **Retail Demo Experience (RDX)**

These Features on Demand add the retail demo package to a device, for use in retail stores. These features are available as .cab packages on the FOD ISO, and should be added with `DISM /add-package`. See [Add or remove packages offline with DISM](#).

**Recommendation:** Preinstall on devices that will be shipped to a retail store. Install the language-neutral FOD, as well as at least one language-specific FOD to ensure that any Windows 10 device can be used as a demo device. Windows will automatically remove these FODs after a user goes through a non-RetailDemo OOBE. See [Retail demo experience](#) for more information about configuring RDX.

#### **Feature:** Language-neutral Retail Demo Content

- **CAB name:** Microsoft-Windows-RetailDemo-OfflineContent-Content-Package.cab

#### **Feature:** Language-specific Retail Demo Content

- **CAB name:** Microsoft-Windows-RetailDemo-OfflineContent-Content-*language-country*-Package.cab

#### **Server Core App Compatibility**

[Server Core App Compatibility](#) significantly improves the app compatibility of the Windows Server Core installation option by including a subset of binaries and packages from Windows Server with Desktop Experience, without adding all components of the Windows Server Desktop Experience graphical environment. This FOD is available on the Server FOD ISO.

#### **Feature:** Server Core App Compatibility

- **CAB name:** Microsoft-Windows-Server-AppCompat-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Availability:** Windows Server 2019 (Server Core only)

## FODs that are not preinstalled

The following is a list of additional FODs you can add to your Windows image to enable additional functionality based on your hardware configuration or to support additional use cases.

#### **Accessibility**

This Feature on Demand enables Braille devices to work with the inbox Narrator screen reader. Without this Feature on Demand, Braille drivers and translation tables will be missing, causing Braille to not function properly.

**Recommendation:** Don't include these Features on Demand in your image, as doing so could conflict with Braille device rights restrictions.

This Feature on Demand's installation can be triggered by a user from the Windows Settings app.

#### **Feature** Accessibility - Braille Support

- **Capability name:** Accessibility.Braille~~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-Accessibility-Braille-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 13.47 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1703 and later

#### **Developer Mode**

An on-device diagnostic platform used via a browser. Installs a SSH server on the device for UWP remote deployment as well as Windows Device Portal.

Enabling Developer Mode will attempt to auto-install this Feature on Demand. On devices that are WSUS-managed, this auto-install will likely fail due to WSUS blocking FOD packages by default. If this Feature on Demand is not successfully installed, device discovery and Device Portal can't be enabled, preventing remote deployment to the device.

**Recommendation:** In general, don't preinstall on devices. If you are building an image for "developer edition" devices, where the primary market for the device is developers or users who plan on developing or testing UWPs, consider preinstalling.

#### **Feature** Windows Developer Mode

- **Capability name:** Tools.DeveloperMode.Core~~~~~0.0.1.0
- **CAB name:** Microsoft-OneCore-DeveloperMode-Desktop-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 13.20 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1607 and later

#### **Graphics Tools**

Used for Direct3D application development. It is typically installed by AAA game engine developers, enterprise graphics software developers, or niche hobbyists.

**Recommendation:** Don't install. This Feature on Demand is only needed by specific users who can trigger installation through Visual Studio when certain optional packages are chosen at install.

#### **Feature:** Graphics Tools

- **Capability name:** Tools.Graphics.DirectX~~~~~0.0.1.0
- **CAB name:** Microsoft-OneCore-Graphics-Tools-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 69.01 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1607 and later

#### **Microsoft WebDriver**

This Feature on Demand enables automated testing of Microsoft Edge and WWA's or WebView controls. This was previously available as a [separate download](#).

**Recommendation:** In general, don't preinstall on devices. If you are building an image for "developer edition" devices, where the primary market for the device is developers or users who plan on testing websites in Microsoft Edge or web content in UWPs, consider preinstalling.

#### **Feature:** Microsoft WebDriver

- **Capability name:** Microsoft.WebDriver~~~~~0.0.1.0
- **CAB name:** Microsoft-WebDriver-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 909 KB
- **Satellites:** None
- **Availability:** Windows 10, version 1809 and later

### **MSIX Packaging Tool Driver**

MSIX Packaging tool driver monitors the environment to capture the changes that an application installer is making on the system to allow MSIX Packaging Tool to repackage the installer as MSIX package.

**Recommendation:** Don't preinstall on devices.

**Feature:** MSIX Packaging Tool Driver

- **Capability name:** Msix.PackagingTool.Driver~~~~~0.0.1.0
- **CAB name:** Msix-PackagingTool-Driver-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 37.74 KB
- **Satellites:** None
- **Availability:** Windows 10, version 1809 and later

### **Networking tools**

The following tools are available as Features on Demand

**Recommendation:** Don't preinstall on devices.

**Feature:** RAS Connection Manager Administration Kit (CMAK)

- **Description:** Create profiles for connecting to remote servers and networks
- **Capability name:** RasCMAK.Client~~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-RasCMAK-Client-Package~31bf3856ad364e35~amd64~en-US~~.cab
- **Satellites:** Yes
- **Install size:** 702.89 KB
- **Availability:** Windows 10, version 1809 and later

**Feature:** RIP Listener

- **Description:** Listens for route updates sent by routers that use the Routing Information Protocol version 1 (RIPV1)
- **Capability name:** RIP.Listener~~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-RasRip-Package~31bf3856ad364e35~amd64~en-US~~.cab
- **Satellites:** Yes
- **Install size:** 45.11 KB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Simple Network Management Protocol (SNMP)

- **Description:** This feature includes SNMP agents that monitor the activity in network devices and report to the network console workstation
- **Capability name:** SNMP.Client~~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-SNMP-Client-Package~31bf3856ad364e35~amd64~en-US~~.cab
- **Satellites:** Yes
- **Install size:** 1.11 MB
- **Availability:** Windows 10, version 1809 and later

#### **Feature:** SNMP WMI Provider

- **Description:** Enables WMI clients to consume SNMP information through the CIM model as implemented by WMI
- **Dependencies:** SNMP.Client
- **Capability name:** WMI-SNMP-Provider.Client~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-WMI-SNMP-Provider-Client-Package~31bf3856ad364e35~amd64~en-US~.cab
- **Satellites:** Yes
- **Install size:** 1.58 MB
- **Availability:** Windows 10, version 1809 and later

#### **OpenSSH Server**

The OpenSSH server Feature on Demand enable the use of OpenSSH on a Windows PC.

**Recommendation:** Don't include this Feature on Demand on your image.

#### **Feature:** OpenSSH Server

- **Capability name:** OpenSSH.Server~~~~0.0.1.0
- **CAB name:** OpenSSH-Server-Package~31bf3856ad364e35\~amd64~~.cab
- **Install size:** 5.61 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1709 and later

#### **Printing**

These Features on Demand are for devices running Windows Server as a Print Server role which supports Azure AD joined devices. If this FOD is not installed, then a Windows Server acting as a Print Server will only support the printing needs of traditional domain joined devices. Azure AD joined devices will not be able to discover corporate printers.

**Recommendation:** Only preinstall the Features on Demand on Windows Server devices running as a Print Server role.

#### **Feature:** Enterprise Cloud Print

- **Capability name:** Print.EnterpriseCloudPrint~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-Printing-EnterpriseCloudPrint-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 3.28 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1709 and later

#### **Feature:** Mopria Cloud Service

- **Capability name:** Print.MopriaCloudService~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-Printing-MopriaCloudService-Package~31bf3856ad364e35~amd64~~.cab
- **Install size:** 11.13 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1709 and later

#### **Remote server administration tools (RSAT)**

Remote Server Administration Tools for Windows 10 lets IT administrators manage Windows Server from a remote computer running the full release version of Windows 10.

**Recommendation:** OEMs shouldn't preinstall these Features on Demand.

**Feature:** Active Directory Domain Services and Lightweight Directory Services Tools

- **Description:** For remotely managing AD DS and AD LDS on Windows Server
- **Dependencies:** Rsat.ServerManager.Tools
- **Capability name:** Rsat.ActiveDirectory.DS-LDS.Tools~~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-ActiveDirectory-DS-LDS-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 17.01 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** BitLocker Drive Encryption Administration Utilities

- **Description:** Tools for managing BitLocker Drive Encryption Features
- **Dependencies:** Rsat.ActiveDirectory.DS-LDS.Tools
- **Capability name:** Rsat.BitLocker.Recovery.Tools~~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-BitLocker-Recovery-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 52.35 KB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Active Directory Certificate Services Tools

- **Description:** Remotely manage AD CS on Windows Server
- **Capability name:** Rsat.CertificateServices.Tools~~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-CertificateServices-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 5.35 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** DHCP Server Tools

- **Description:** DHCP MMC snap-in, SHCP server netsh context, and Windows PowerShell module for DHCP Server
- **Capability name:** Rsat.DHCP.Tools~~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-DHCP-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 7.09 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** DNS Server Tools

- **Description:** DNS MMC snap-in, dnscmd.exe command-line tool, and Windows PowerShell module for DHCP Server
- **Capability name:** Rsat.Dns.Tools~~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-DNS-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 6.15 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Failover Clustering Tools

- **Description:** Failover Cluster Manager snap-in, the Cluster-Aware Updating interface, and the Failover Cluster module for Windows PowerShell
- **Dependencies:** Rsat.FileServices.Tools
- **Capability name:** Rsat.FailoverCluster.Management.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-FailoverCluster-Management-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 34.35 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** File Services Tools

- **Description:** For remotely managing the File Services role on Windows Server
- **Dependencies:** Rsat.ServerManager.Tools
- **Capability name:** Rsat.FileServices.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-FileServices-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 17.14 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Group Policy Management Tools

- **Capability name:** Rsat.GroupPolicy.Management.Tools~~~0.0.1.0
- **Description:** Group Policy Management Console, GP Management Editor, and GP Started GPO Editor
- **CAB name:** Microsoft-Windows-GroupPolicy-Management-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 18.54 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** IP Address Management (IPAM) Client

- **Description:** Used to connect to and manage a remote IPAM server
- **Dependencies:** Rsat.ServerManager.Tools
- **Capability name:** Rsat.IPAM.Client.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-IPAM-Client-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 872.56 KB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Data Center Bridging LLDP Tools

- **Description:** Includes PowerShell tools for remotely managing LLDP agents on Windows Server
- **Capability name:** Rsat.LLDP.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-LLDP-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** None
- **Install size:** 19.19 KB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Network Controller Management Tools

- **Description:** PowerShell tools for managing the Network Controller role on Windows Server

- **Capability name:** Rsat.NetworkController.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-NetworkController-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 767.20 KB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Network Load Balancing Tools

- **Description:** Network Load Balancing Manager snap-in, the Network Load balancing module for PowerShell, and the nlb.exe and wlbs.exe command-line tools
- **Capability name:** Rsat.NetworkLoadBalancing.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-NetworkLoadBalancing-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 859.35 KB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Remote Access Management Tools

- **Description:** Graphical and PowerShell tools for managing the Remote Access role on Windows Server
- **Dependencies:** Rsat.ServerManager.Tools, Rsat.GroupPolicy.Management.Tools
- **Capability name:** Rsat.RemoteAccess.Management.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-RemoteAccess-Management-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 27.70 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Remote Desktop Services Tools

- **Description:** Snap-ins for Remote Desktop Licensing Manager, Remote Desktop Licensing Diagnostics, and Remote Desktop Gateway Manager.
- **Capability name:** Rsat.RemoteDesktop.Services.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-RemoteDesktop-Services-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 3.60 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Server Manager

- **Description:** Server Manager console and PowerShell tools for remotely managing Windows Server
- **Dependencies:** None. NICTeaming.Tools.Legacy is already merged with this package.
- **Capability name:** Rsat.ServerManager.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-ServerManager-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 32.47 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Shielded VM Tools

- **Description:** Includes the Provisioning Data File Wizard and the Template Disk Wizard
- **Dependencies:** Rsat.ServerManager.Tools

- **Capability name:** Rsat.Shielde.d.VM.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-Shielded-VM-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 20.09 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Storage Replica Module for Windows PowerShell

- **Description:** Remotely manage the Storage Replica feature with PowerShell
- **Capability name:** Rsat.StorageReplica.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-StorageReplica-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 1.58 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Volume Activation Tools

- **Description:** Manage volume activation license keys on a KMS host or in Microsoft AAD
- **Capability name:** Rsat.VolumeActivation.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-VolumeActivation-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 556.39 KB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Windows Server Update Services Tools

- **Description:** Includes graphical and PowerShell tools for managing WSUS
- **Capability name:** Rsat.WSUS.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-WSUS-Tools-FoD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Lang
- **Install size:** 8.27 MB
- **Availability:** Windows 10, version 1809 and later

**Feature:** Storage Migration Service Management Tools

- **Description:** Provides management tools for storage migration jobs
- **Capability name:** Rsat.StorageMigrationService.Management.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-StorageMigrationService-Management-Tools-FOD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 303.98
- **Availability:** Windows 10, version 1809 and later

**Feature:** Systems Insights Module for Windows PowerShell

- **Description:** Provides the ability to manage the System Insights feature
- **Capability name:** Rsat.SystemInsights.Management.Tools~~~0.0.1.0
- **CAB name:** Microsoft-Windows-SystemInsights-Management-Tools-FOD-Package~31bf3856ad364e35~amd64~~.cab
- **Satellites:** Lang
- **Install size:** 123.43 KB
- **Availability:** Windows 10, version 1809 and later

## Storage

This Feature on Demand enables advanced storage capabilities. To add Windows Storage Management, you have to add both FODs to an image.

**Recommendation:** In general, OEMs shouldn't preinstall this FOD. OEMs and IT pros can consider preinstalling this FOD when they are building a PC that requires advanced storage capabilities.

### Feature: Windows Storage Management

- **Description:** Windows Storage Management allows you to manage a wide range of storage configurations, from single-disk desktops to external storage arrays.
- **Capability name:** Microsoft.Windows.StorageManagement~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-StorageManagement-FoD-Package~31bf3856ad364e35\~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 15.38 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1809 and later

### Feature: Windows Storage Management

- **CAB name:** Windows Storage Management allows you to manage a wide range of storage configurations, from single-disk desktops to external storage arrays.
- **Capability name:** Microsoft.OneCore.StorageManagement~~~~0.0.1.0
- **CAB name:** Microsoft-OneCore-StorageManagement-FoD-Package~31bf3856ad364e35\~amd64~~.cab
- **Satellites:** Yes
- **Install size:** 9.80 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1809 and later

## XPS Viewer

This Feature on Demand allows you to read, copy, print, sign, and set permissions for XPS documents.

**Recommendation:** In general, OEMs shouldn't preinstall this FOD. OEMs and IT pros can consider preinstalling this FOD when they are building a PC that requires advance storage capabilities.

### Feature: XPS Viewer

- **Capability name:** XPS.Viewer~~~~0.0.1.0
- **CAB name:** Microsoft-Windows-Xps-Xps-Viewer-Opt-Package~31bf3856ad364e35\~amd64~~.cab
- **Install size:** 16.91 MB
- **Satellites:** None
- **Availability:** Windows 10, version 1803 and later

## Related topics

[Features on Demand](#)

[Language and region Features on Demand](#)

# Enable or Disable Windows Features Using DISM

1/18/2019 • 5 minutes to read

The Deployment Image Servicing and Management (DISM) tool is a command-line tool that is used to modify Windows® images. You can use DISM to enable or disable Windows features directly from the command prompt, or by applying an answer file to the image. You can enable or disable Windows features offline on a WIM or VHD file, or online on a running operating system.

## To mount an offline image for servicing

1. Open a command prompt with administrator privileges.
2. To use DISM from an installation of the Windows Assessment and Deployment Kit (Windows ADK), locate the Windows ADK servicing folder and navigate to this directory. By default, DISM is installed at C:\Program Files (x86)\Windows Kits\10.0\Assessment and Deployment Kit\Deployment Tools\ in Windows 10, C:\Program Files (x86)\Windows Kits\8.1\Assessment and Deployment Kit\Deployment Tools\ in Windows 8.1 and C:\Program Files (x86)\Windows Kits\8.0\Assessment and Deployment Kit\Deployment Tools\ in Windows 8.

DISM is available in:

- Windows 10
- Windows 8.1
- Windows 8
- Windows Server 2016 Technical Preview
- Windows Server 2012 R2
- Windows Server 2012
- Windows Preinstallation Environment (WinPE) for Windows 10
- WinPE 5.0
- WinPE 4.0

You can install DISM and other deployment and imaging tools, such as Windows System Image Manager (Windows SIM), on another supported operating system from the Windows ADK. For more information, see [DISM Supported Platforms](#).

3. Use the `/Get-ImageInfo` option to retrieve the name or index number for the image that you want to modify. An index or name value is required for most operations that specify an image file.

For example, at the command prompt type:

```
Dism /Get-ImageInfo /ImageFile:C:\test\images\install.wim
```

4. Mount the offline Windows image. For example, type:

```
Dism /Mount-Image /ImageFile:C:\test\images\install.wim /Name:"Base Windows Image"  
/MountDir:C:\test\offline
```

## To find available Windows features in an image

1. List all of the features available in the operating system. For example, type:

```
Dism /online /Get-Features
```

To service an offline image, specify the location of the mounted image directory. For example, type:

```
Dism /Image:C:\test\offline /Get-Features
```

You can use `>featurelist.txt` to redirect the output of the command to a text file that is named featurelist.

2. Review the list of features to find the feature that you want to enable, disable, remove, or restore.
3. Use `/Get-FeatureInfo` to list information about the specific feature you are interested in. For example, type:

```
Dism /online /Get-FeatureInfo /FeatureName:TFTP
```

## To enable Windows features

1. Enable a specific feature in the image. You can use the `/All` argument to enable all of the parent features in the same command. For example, type:

```
Dism /online /Enable-Feature /FeatureName:TFTP /All
```

To service an offline image, specify the location of the mounted image directory. For example, type:

```
Dism /Image:C:\test\offline /Enable-Feature /FeatureName:TFTP /All
```

2. Optional: Get the status of the feature you have enabled. For example, type:

```
Dism /online /Get-FeatureInfo /FeatureName:TFTP
```

If the status is **Enable Pending**, you must boot the image in order to enable the feature entirely.

## To restore removed Windows features

1. Enable a specific feature in the image. If you do not specify a source, DISM will look in the default location specified by group policy for the required files needed to enable the feature. For more information, see [Configure a Windows Repair Source](#).

If the files are not found in the default location, DISM will contact Windows Update (WU) for the required files. You can use the `/LimitAccess` argument to prevent DISM from contacting WU.

If you specify multiple `/Source` arguments, the files are gathered from the first location where they are found and the rest of the locations are ignored.

For example, type:

```
Dism /Online /Enable-Feature /FeatureName:TFTP /Source:Z:\sources\SxS /Source:C:\test\mount\windows  
/LimitAccess
```

To service an offline image, specify the location of the mounted image directory. For example, type:

```
Dism /Image:C:\test\offline /Enable-Feature /FeatureName:TFTP /Source:C:\test\mount\windows
```

2. Optional: Get the status of the feature you have enabled. For example, type:

```
Dism /online /Get-FeatureInfo /FeatureName:TFTP
```

If the status is **EnablePending**, you must boot the image in order to enable the feature entirely.

## To disable Windows features

1. Disable a specific feature in the image. For example, type:

```
Dism /online /Disable-Feature /FeatureName:TFTP
```

To service an offline image, specify the location of the mounted image directory. For example, type:

```
Dism /Image:C:\test\offline /Disable-Feature /FeatureName:TFTP
```

2. Optional: Use **DISM /GetFeatureInfo** to get the status of the feature you have disabled. For example, type:

```
Dism /online /Get-FeatureInfo /FeatureName:TFTP
```

If the status is **DisablePending**, you must boot the image in order to disable the feature entirely.

## To remove Windows features for on-demand installation

1. Remove a specific feature in the image without removing the feature's manifest from the image. This option can only be used when servicing Windows 10, Windows 8.1, Windows 8, Windows Server 2016 Technical Preview, Windows Server 2012 R2, or Windows Server 2012. For more information, see [Configure a Windows Repair Source](#).

For example, type:

```
Dism /online /Disable-Feature /FeatureName:TFTP /Remove
```

To service an offline image, specify the location of the mounted image directory. For example, type:

```
Dism /Image:C:\test\offline /Disable-Feature /FeatureName:TFTP /Remove
```

2. Optional: Use **DISM /GetFeatureInfo** to get the status of the feature you have disabled. For example, type:

```
Dism /online /Get-FeatureInfo /FeatureName:TFTP
```

The status is **Disabled**. Beginning with Windows 10, the payload is not removed from Windows client SKUs in order to support push-button reset. The payload is removed from Windows Server SKUs.

## To enable or disable Windows features by using DISM and an answer file

1. In Windows SIM, open an existing catalog by clicking **Select a Windows Image** on the **File** menu and specifying the catalog file type (.clg) in the drop-down list, or create a new catalog by clicking **Create Catalog** on the **Tools** menu.
2. Expand the catalog in the **Windows Image** pane, and then expand **Packages**.
3. Expand **Foundation**, and right-click **Microsoft-Windows-Foundation-Package**.
4. Click **Add to Answer File**.
5. Click **Enabled** or **Disabled** next to the features that you intend to enable or disable. Click the arrow to select the opposite choice.

You might have to expand an item to see all its children. You must enable the parent if any one of its children are enabled.

**Note** You can't restore or remove a Windows feature for features on demand with an unattended answer file.

6. Click **Tools** on the main menu, and then click **Validate Answer File**.
7. Correct any errors that appear in the **Messages** pane, and save the answer file.
8. At the command prompt, type the following command to apply the unattended answer file to the image.

```
Dism /online /Apply-Unattend:C:\test\answerfiles\myunattend.xml
```

To service an offline image, specify the location of the mounted image directory. For example, type:

```
Dism /Image:C:\test\offline /Apply-Unattend:C:\test\answerfiles\myunattend.xml
```

## To commit changes on an offline image

- Commit the changes and unmount the image. For example, type:

```
Dism /Unmount-Image /MountDir:C:\test\offline /Commit
```

## Related topics

[DISM - Deployment Image Servicing and Management Technical Reference for Windows](#)

[DISM Operating System Package Servicing Command-Line Options](#)

[DISM Unattended Servicing Command-Line Options](#)

[Configure a Windows Repair Source](#)

# Configure a Trusted Image Identifier for Windows Defender

1/18/2019 • 2 minutes to read

Make your devices work faster right out of the box by adding a trusted image identifier to Windows Defender.

You can speed up the initial performance of your PCs and devices for the end user by adding a trusted image identifier to Windows Defender. Windows Defender can help to prevent, remove, and quarantine malware and spyware.

By default, Windows Defender performs a scan of each file on the device when the device accesses the file for the first time. This is known as an on-access scan. Optimization mechanisms, such as caching, help reduce unnecessary scans of files that have already been scanned. When Windows Defender performs a quick scan or a full scan (also known as on-demand scans), the rest of the files on the system will be marked as safe.

**Note** If you have already deployed a series of devices, and then later determine that there is a potential problem with the security of the image, contact your Depth Project Manager (PM) within the Windows Ecosystem Engagement team, and provide the unique identifier of the image. Microsoft will add this unique identifier into Windows Update. After a device with that unique identifier receives updates from Windows Update, Windows Defender performs scans on all of the files on that device.

## Adding a Trusted Image Identifier

For optimal performance, add this setting when you prepare the device for final deployment, after you perform a full scan of the final image.

### To add a trusted image identifier

1. Create a Windows Setup answer file (unattend.xml), and add the `Security-Malware-Windows-Defender\TrustedImageIdentifier` setting. To learn more, see the OEM deployment lab: [Customize Windows with an answer file](#).
2. For the `TrustedImageIdentifier` setting, specify a unique identifier for the image, such as a GUID or other unique value (example: "Contoso Laptop Model 1 2018-07-31").
3. [Apply the image to a new device, and boot it into audit mode](#), and scan the image by using Windows Defender or another scanning tool. This can help make sure that the image is safe.
4. Reseal the image, adding the answer file with the `TrustedImageIdentifier` setting.

```
C:\Windows\System32\Sysprep\sysprep /oobe /generalize /unattend:c:\recovery\oem\Unattend.xml /shutdown
```

5. Apply the image to new devices, and deliver them to customers.

The next time that the device starts, Windows identifies all of the files currently on the system, and skips these files during subsequent scans.

## Related topics

[Use Answer Files with Sysprep](#)

# Configure Windows System Assessment Test Scores

1/18/2019 • 7 minutes to read

The Windows® System Assessment Tests (WinSAT) are used to analyze the performance of several system components, including CPU, memory, disk, and graphics.

The WinSAT results are summarized in the **Performance Information and Tools** Control Panel item as Windows Experience Index (WEI) scores. These scores show consumers the performance characteristics of their systems.

WinEI scores are no longer generated during OOBE, nor are prep xml files used to create WinSAT formal files during OOBE. We recommended that you generate the WinSAT formal file on the system prior to shipping it to the end-users. This allows WinSAT scores to be available as soon as end-user boots their systems, and allows the optimizations that depend on these results to be immediately available. Because the assessments are not run during the out-of-box experience, the WinSAT and WEI scores are no longer generated when a user finishes OOBE. Instead, the scores can be generated at two other times, using other mechanisms besides prepopulating WinSAT on the system that will ship.

- End users can explicitly request an assessment by using the **Re-run the assessment** option in the **Performance Information and Tools** Control Panel item.
- When the system is idle, subsequent to the first boot, the remaining WinSAT assessments will run using the Maintenance Scheduler if they were not prepopulated.

## To run WinSAT on a complete system

Use the **prep** option with the WinSAT command-line tool to run assessments against component systems.

To run WinSAT per computer (for all systems):

1. Install Windows 8 and boot to audit mode. For more information about audit mode, see [Audit Mode Overview](#).
2. Add supplemental components, such as out-of-box drivers.
3. Run **WinSAT prep**.

This will generate the WinSAT prep .xml results files to the Datastore directory, located at:

`%WINDIR%\performance\winsat\datastore\`

4. [Optional] If you plan to capture this installation to deploy onto other computers, run **sysprep /generalize /audit /shutdown** and then capture the installation. Deploy the image to a PC that you intend to ship, and boot it.
5. Verify that Windows boots to audit mode, and then run **WinSAT moobe**.

This generates a WinSAT formal file from the matching prep files, and ensures that the WinSAT formal file is available when the end-user boots the system the first time. Windows scales some features based on the WinSAT formal file, and if this file is not present on the system, then the system might experience performance problems, including unnecessary storage device defragmentation, lack of optimized memory management and prefetching optimizations.

**Note** To reduce the time a PC spends on the factory floor, we recommend using **WinSAT prep** when you are creating your master Windows images. On the factory floor, you would only need to run **WinSAT moobe**. However, if you want to run both **WinSAT prep** and **WinSAT moobe** on the factory floor, you

can use **WinSAT formal** instead. This option creates the same set of files as running both **WinSAT prep0p** and **WinSAT moobe** and should be used in scenarios when you are not able to run **WinSAT prep0p** on your master Windows images.

6. Run the **sysprep /oobe** to configure Windows to boot to OOBE.

**Warning** Running **sysprep /generalize** after running **WinSAT moobe** will delete the results that **WinSAT moobe** created. We recommend that you run **WinSAT moobe** or **WinSAT formal** on the factory floor for each PC that you intend to ship to a customer.

The system is now ready to be shipped to a customer. The benefit of running all of the WinSAT assessments per computer image is that the customer's computer always has a complete set of WinSAT results. It also has the most accurate WinSAT results. In this use, accurate means that if the consumer used on-demand rating of a system, that system would get a rating equal to or greater than the rating that was prepopulated by WinSAT.

Pre-population is not meant to enable transferring WinSAT data among systems with very different capabilities, such as among laptops and desktops, because the data is not accurate across widely differing systems. Instead, it is meant to make it easier to re-use WinSAT data among similar systems; those systems that contain the same motherboard/chipset and similar CPU, video cards and disks.

The following procedure describes how to run WinSAT on selected configurations within a line of similar computers. This involves running the **WinSAT prep0p** commands multiple times.

## To run WinSAT for selective PC configurations and PC components

1. Identify the configurations that you intend to include in the PC, including video processors, memory, and storage devices.
2. Install Windows 8 and boot to audit mode. For more information about audit mode, see [Audit Mode Overview](#).
3. Add supplemental components, such as out-of-box drivers.
4. Run **WinSAT prep0p**.
5. Run **Sysprep /generalize /audit /reboot**. This will remove any non-prepop WinSAT .xml files.
6. Copy the resulting WinSAT prep0p .xml files from `%WINDIR%\performance\winsat\datastore` to the network share that you are using to store WinSAT results.
7. Upgrade one of the components. For example, increase the memory of one configuration in your set of computers.
8. Run **WinSAT prep0p -mem** test. Using the tool this way ensures that only tests relevant to the specified component will run. An additional .xml file is generated that shows the memory test results.
9. Restore the original memory configuration, and upgrade a different component, such as the video card.

**Note** Because WinSAT results can be used with configurations of the same level or higher, if you revert to the base configuration, the test results are relevant to a broader range of computers.

10. Re-run the test using the **WinSAT prep0p -graphics** command. Only tests relevant to the specified component run. An additional .xml file is generated for the Graphics results.
11. Store the new results files with the original .xml results files on your network share.
12. To prepopulate the WinSAT results for a new computer with similar components, copy the .xml files from the network share to the target computer's WinSAT Datastore directory:  
`%WINDIR%\performance\winsat\datastore`. You can copy the entire set of WinSAT prep0p files from the network share to the local WinSAT directory. WinSAT will find the correct set for the current computer.

13. On the new computer run `WinSAT moobe`. This generates a WinSAT formal file from the matching prepop files, and ensures that the WinSAT formal file is available when the end-user boots the system the first time. Windows scales some features based on the WinSAT formal file, and if this file is not present on the system, then the system might experience performance problems, including unnecessary storage device defragmentation, lack of optimized memory management and prefetching optimizations.

When running **WinSAT moobe** WinSAT examines the following directory for results files:

`%WINDIR%\performance\winsat\datastore`. If WinSAT does not discover a relevant set of .xml files, it will ignore the irrelevant files and treat the system as unrated. The DWM test will run immediately, and the other tests will run as a maintenance task, or when the end-user opts to run the tests from the **Performance Information and Tools** Control Panel item. If WinSAT finds a relevant set of prepop .xml files, it uses the files to generate a formal .xml file which will be available for use when the end-user boots the computer for the first time. This enables scaling of features and allows Windows to perform appropriate optimizations.

WinSAT determines relevance by using hardware IDs. This includes: CPUID, memory DIMM configuration, hard disk model and size, and video card PNP ID. If the relevant secondary assessment is not present, WinSAT will run both the primary and secondary assessments; for example, both CPU and memory.

The advantage of this second option, running on selective configurations, is that WinSAT assessments may be run on fewer configurations and copied to similar systems. The disadvantage is that if a set of WinSAT files is not relevant to the current system, those tests will be ignored and the system will be treated as unrated, and optimizations and feature scaling will not be performed when the end-user boots the computer.

## WinSAT Prepop Command-line Options

**The syntax for prepopulation is as follows:**

```
Winsat prepop [-datastore <directory>][-graphics | -cpu | -mem | -disk | -dwm]
```

The following command runs all WinSAT tests: `Winsat prepop`.

You can prepopulate only one subsystem, such as DWM, subject to the following dependencies:

- The DWM assessment can be run independently.
- The disk assessment can be run independently.
- The CPU assessment requires that a relevant memory assessment is present.
- The memory assessment requires that a relevant CPU assessment is present.
- The graphics assessment requires that relevant CPU and memory assessments are present.

**The syntax for moobe is as follows:**

```
Winsat moobe [-datastore <directory>]
```

**The WinSAT file naming pattern is as follows:**

For Windows 8, there is a `%type%` identifier, `Prepop`. This identifies datastore files that are a result of prepopulation. The naming pattern is:

```
%IdentifierDerivedFromDate% %Component%.Assessment(Prepop).WinSAT.xml
```

Where `%IdentifierDerivedFromDate%` is year-month-day and time represented as, for example, `0012-08-01 14.48.28` where the test was run on August 1, 2012 at 2:48:28 PM.

A WinSAT formal file created from running **winsat prep** followed by **winsat moobe**; or from running **winsat formal** uses the following naming pattern:

```
%IdentifierDerivedFromDate% Formal.Assessment(Initial).WinSAT.xml
```

## Related topics

[Windows Deployment Options](#)

# Add or Remove Packages Offline Using DISM

1/18/2019 • 3 minutes to read

Deployment Image Servicing and Management (DISM.exe) is a command-line tool that is used to update offline Windows® images. There are two ways to install or remove packages offline with DISM. You can either apply an unattend answer file to the offline image, or you can add or remove the package directly from the command prompt.

If you are installing multiple packages to a Windows image, and there are dependency requirements, the best way to ensure the correct order of the installation is by using an answer file. You can use DISM to apply the Unattend.xml answer file to the image. When you use DISM to apply an answer file, the unattend settings in the **offlineServicing** configuration pass are applied to the Windows image.

You must install the latest version of the Windows Assessment and Deployment Kit (Windows ADK), which contains all of the tools that are required, including DISM.

## To add packages to an offline image by using DISM

- At an elevated command prompt, locate the Windows ADK servicing folder, and type the following command to retrieve the name or index number for the image that you want to modify.

```
Dism /Get-ImageInfo /ImageFile:C:\test\images\install.wim
```

An index or name value is required for most operations that specify an image file.

- Type the following command to mount the offline Windows image.

```
Dism /Mount-Image /ImageFile:C:\test\images\install.wim /Name:"Windows 7 HomeBasic"  
/MountDir:C:\test\offline
```

- At a command prompt, type the following command to add a specific package to the image. You can add multiple packages on one command line. They will be installed in the order listed in the command line.

```
Dism /Image:C:\test\offline /Add-Package /PackagePath:C:\packages\package1.cab  
/PackagePath:C:\packages\package2.cab
```

- At a command prompt, type the following command to commit the changes and unmount the image.

```
Dism /Unmount-Image /MountDir:C:\test\offline /Commit
```

## To remove packages from an offline image by using DISM

- At an elevated command prompt, locate the Windows ADK servicing folder, and type the following command to retrieve the name or index number for the image that you want to modify.

```
Dism /Get-ImageInfo /ImageFile:C:\test\images\install.wim
```

An index or name value is required for most operations that specify an image file.

- Type the following command to mount the offline Windows image.

```
Dism /Mount-Image /ImageFile:C:\test\images\install.wim /Name:"Windows 7 HomeBasic"  
/MountDir:C:\test\offline
```

- Optional: Type the following command to list the packages in the image.

```
Dism /Image:C:\test\offline /Get-Packages
```

You can use `>featurelist.txt` to redirect the output of the command to a text file that is named FeatureList.

- Review the list of packages that are available in your mounted image and note the package identity of the package.
- At a command prompt, specify the package identity to remove it from the image. You can remove multiple packages on one command line.

```
DISM /Image:C:\test\offline /Remove-Package  
/PackageName:Microsoft.Windows.Calc.Demo~6595b6144ccf1df~x86~en~1.0.0.0 /PackageName:Microsoft-Windows-  
MediaPlayer-Package~31bf3856ad364e35~x86~~6.1.6801.0
```

You can use the **/PackagePath** option to point to the original source of the package, or to specify the path to the .cab file, or you can use the **/PackageName** option to specify the package by name as it is listed in the image. For more information, see [DISM Operating System Package Servicing Command-Line Options](#).

- At a command prompt, type the following command to commit the changes and unmount the image.

```
Dism /Unmount-Image /MountDir:C:\test\offline /Commit
```

## To add or remove packages offline by using DISM and an answer file

- Open Windows SIM.
- To add a new package, click **Insert** on the main menu, and select **Package(s)**. Browse to the package you want to add, and then click **Open**.
- To remove an existing package, select the package in the **Answer file** pane that you want to remove. In the **Properties** pane, change the **Action** property to **Remove**.
- Note** The packages must be added to the **offlineServicing** configuration pass.
- Validate and save the answer file.
- At an elevated command prompt, locate the Windows ADK servicing folder, and then type the following command to retrieve the name or index number for the image that you want to mount.

```
Dism /Get-ImageInfo /ImageFile:C:\test\images\install.wim
```

- Type the following command to mount the offline Windows image.

```
Dism /Mount-Image /ImageFile:C:\test\images\install.wim /name:"Windows 7 HomeBasic"  
/MountDir:C:\test\offline
```

An index or name value is required for most operations that specify an image file.

7. At a command prompt, type the following command to apply the unattended answer file to the image.

```
DISM /Image:C:\test\offline /Apply-Unattend:C:\test\answerfiles\myunattend.xml
```

8. At a command prompt, type the following command to commit the changes and unmount the image.

```
Dism /Unmount-Image /MountDir:C:\test\offline /Commit
```

For more information about Windows SIM, see [Windows Setup Technical Reference](#).

## Related topics

[DISM - Deployment Image Servicing and Management Technical Reference for Windows](#)

[DISM Operating System Package Servicing Command-Line Options](#)

[DISM Unattended Servicing Command-Line Options](#)

# Configure Oobe.xml

1/18/2019 • 2 minutes to read

Oobe.xml is a content file used to collect text and images for customizing Windows® OOBE. To build a single Windows image that contains multiple languages to deliver to more than one country or region, you can add multiple Oobe.xml files to customize the content based on the language and country/region selections of the customer.

## In This Section

[Oobe.xml Settings](#)

[How Oobe.xml Works](#)

## Related topics

[Windows Deployment Options](#)

# Oobe.xml Settings

1/18/2019 • 6 minutes to read

This topic describes the settings that can be set in Oobe.xml.

## Oobe.xml Settings

The following shows how elements are ordered in Oobe.xml. Not all elements and sections are required for Windows to process Oobe.xml.

```

<FirstExperience>
  <oobe>
    <oem>
      <name></name>
      <eulafilename></eulafilename>
      <computername></computername>
      <registration>
        <title></title>
        <subtitle></subtitle>
        <customerinfo>
          <label></label>
          <defaultValue></defaultValue>
        </customerinfo>
        <checkbox1>
          <label></label>
          <defaultValue></defaultValue>
        </checkbox1>
        <checkbox2>
          <label></label>
        </checkbox2>
        <checkbox3>
          <label></label>
        </checkbox3>
        <link1>
          <label></label>
        </link1>
        <link2>
          <label></label>
        </link2>
        <link3>
          <label></label>
        </link3>
        <hideSkip></hideSkip>
      </registration>
    </oem>
    <defaults>
      <language></language>
      <location></location>
      <keyboard></keyboard>
      <adjustForDST></adjustForDST>
    </defaults>
    <hidSetup>
      <title></title>
      <mouseImagePath></mouseImagePath>
      <mouseText></mouseText>
      <mouseErrorImagePath></mouseErrorImagePath>
      <mouseErrorText></mouseErrorText>
      <keyboardImagePath></keyboardImagePath>
      <keyboardErrorImagePath></keyboardErrorImagePath>
      <keyboardText></keyboardText>
      <keyboardPINText></keyboardPINText>
      <keyboardPINImagePath></keyboardPINImagePath>
      <keyboardErrorText></keyboardErrorText>
    </hidSetup>
  </oobe>
</FirstExperience>

```

The following tables show descriptions and values for elements available in Oobe.xml.

The following table shows description for OEM customization and registration pages.

ELEMENT	SETTING	DESCRIPTION	VALUE
<oem>			

ELEMENT	SETTING	DESCRIPTION	VALUE
	<name>	Optional. Text to describe the name of the OEM.	String.
	<eulafilename>	Optional. Text with the filename of the EULA file.	Absolute path to the EULA .rtf file. The EULA .html document must be in the same folder. Windows knows to look for the .html file in that location.  <b>Note:</b> .htm files are ignored.  <b>Important:</b> All HTML files in OOBEM must use UTF-8 encoding.  See <a href="#">OEM license terms</a> to learn about creating an .html EULA file.
	<computername>	Optional. Text to describe the name of the computer	String.
	<registration>	Optional. Additional details are below.	
<registration>			
	<title>	Required if registration element is used. Text to title the Registration page.	String of up to 25 characters.
	<subtitle>	Required if registration element is used. Text to describe the Registration page.	
<customerinfo>			
	<label>	Text to label customerinfo. Required for customerinfo to appear.	String of up to 250 characters. We strongly recommend that you use no more than 100 characters because this length of text will fit on one line.
	<defaultvalue>	Value to set customerinfo as selected or not. If this field is checked, information from the four input fields will be provided via asymmetric key encryption. If not checked, no information from the four input fields will be provided.	<input type="checkbox"/> True or <input type="checkbox"/> False. True means the check box default condition is selected. False means the check box default condition isn't selected.
<checkbox1>			

ELEMENT	SETTING	DESCRIPTION	VALUE
	<label>	Text to label checkbox1. Required for checkbox1 to appear.	String of up to 250 characters. We strongly recommend that you use no more than 100 characters because this length of text will fit on one line.
	<defaultvalue>	Value to set checkbox1 as selected or not selected.	True or False . True means the check box default condition is selected. False means the check box default condition isn't selected.
<checkbox2>			
	<label>	Text to label checkbox2. Required for checkbox2 to appear.	String of up to 250 characters. We strongly recommend that you use no more than 100 characters because this length of text will fit on one line.
	<defaultvalue>	Value to set checkbox3 as selected or not selected.	True or False . True means the check box default condition is selected. False means the check box default condition isn't selected.
<checkbox3>			
	<label>	Text to label checkbox3. Required for checkbox3 to appear.	String of up to 250 characters. We strongly recommend that you use no more than 100 characters because this length of text will fit on one line.
	<defaultvalue>	Value to set checkbox3 as selected or not selected.	True or False . True means the check box default condition is selected. False means the check box default condition isn't selected.
<link1>			
	<label>	Label for the link to the HTML file. Required for link1 to appear.	String of up to 100 characters.

ELEMENT	SETTING	DESCRIPTION	VALUE
	<link>	File must be named linkfile1.html. OOBE searches for these files under the oobe\info folder. OOBE searches for files under the appropriate locale and language specific subfolders of oobe\info.	linkfile1.html
<link2>			
	<label>	Label for the link to the HTML file. Required for link2 to appear.	String of up to 100 characters.
	<link>	File must be named linkfile2.html. OOBE searches for these files under the oobe\info folder. OOBE searches for files under the appropriate locale and language specific subfolders of oobe\info.	linkfile2.html
<link3>			
	<label>	Label for the link to the HTML file. Required for link3 to appear.	String of up to 100 characters.
	<link>	File must be named linkfile3.html. OOBE searches for these files under the oobe\info folder. OOBE searches for files under the appropriate locale and language specific subfolders of oobe\info.	linkfile3.html
<hideSkip>		Optional. Controls whether or not the Skip button is displayed to the user. Default is False, resulting in the skip button being visible.	True or False. True means the skip button is not visible to the user. False means the skip button is displayed as an option to the user.

The following table shows values for language and location.

ELEMENT	SETTING	DESCRIPTION	VALUE
<defaults>			

ELEMENT	SETTING	DESCRIPTION	VALUE
	<language>	Default language.	Language identifier (decimal), for example, French = 3084 . For values, see <a href="#">Available Language Packs for Windows</a> .
	<location>	Default location.	GeOID (decimal), example, Canada = 39 . For values, see <a href="#">Table of Geographic Locations</a>
	<keyboard>	Specify both locale and keyboard layout in hexadecimal, separated by a colon.	Specify both locale and keyboard layout in hexadecimal, separated by a colon. For example, for US English input/keyboard, use 0409:00000409. For values, see <a href="#">Default Input Profiles (Input Locales)</a>
	<adjustforDST>	Specifies whether to adjust for Daylight Saving Time.	True or False . True means adjust for Daylight Saving Time based on the time zone. False means always remain on Standard Time.

The following table shows values for HID setup.

ELEMENT	SETTING	DESCRIPTION	VALUE
<hidsetup>			
	<title>		
	<mouseImagePath>	Absolute path to the mouse pairing instruction image. The image must not be larger than 630 x 372 pixels. It's scaled to fit in portrait mode or on small form factors.	Absolute path to the image.
	<mouseText>	Help text that displays at the bottom of the page.	String
	<mouseErrorImagePath>	Absolute path to the mouse pairing error image. The image must not be larger than 630 x 372 pixels. It's scaled to fit in portrait mode or on small form factors.	

Element	Setting	Description	Value
	<mouseErrorText>	Error that displays to users along with mouse pairing error image.	String
	<keyboardImagePath>	Absolute path to the first keyboard pairing instruction image.  The image must not be larger than 630 x 372 pixels. It's scaled to fit in portrait mode or on small form factors.	
	<keyboardErrorImagePath>	Absolute path to the keyboard pairing error image.  The image must not be larger than 630 x 372 pixels. It's scaled to fit in portrait mode or on small form factors.	Absolute path to the image
	<keyboardText>	Specifies the text to prompt the user to pair the keyboard.	String
	<keyboardPINText>	Specifies the prompt text for the user to enter a pin for the keyboard.	String
	<keyboardPINImagePath>	Absolute path to the keyboard pairing instruction image.  The image must not be larger than 630 x 372 pixels. It's scaled to fit in portrait mode or on small form factors.	Absoulte path to image
	<keyboardErrorText>	Specifies the text to use when an error occurs when pairing the keyboard.	String

## How to Customize Oobe

### To customize Oobe by using Oobe.xml

1. Create a file named Oobe.xml and store this file in Windows\System32\Oobe\Info.
2. By using an XML editor or a text editor, such as Notepad, update Oobe.xml with the appropriate files, paths, and content.
3. Save your updated version of Oobe.xml in Windows\System32\Oobe\Info, or in the appropriate language-and locale-specific folders required for your customizations.

4. Test OOBE.

**Test OOBE**

- a. On the **Start** menu, point to **All Programs**, and then click **Accessories**.
- b. Right-click the command prompt shortcut, and click **Run as administrator**. Accept the **User Account Control** dialog box.
- c. Navigate to \Windows\System32\Sysprep
- d. Run **sysprep /oobe**.
- e. Start the computer.

## Related topics

[Configure Oobe.xml](#)

# How Oobe.xml Works

1/18/2019 • 4 minutes to read

**Oobe.xml** is a content file that you can use to organize text and images and to specify and preset settings for customizing the Windows first experience. You can use multiple **Oobe.xml** files for language- and region-specific license terms and settings so that users see appropriate information as soon as they start their PCs. By specifying information in the **Oobe.xml** file, OEMs direct users to perform only the core tasks that are required to set up their PCs.

Windows checks for and loads **Oobe.xml** in the following locations, in the following order:

1. %WINDIR%\System32\Oobe\Info\Oobe.xml
2. %WINDIR%\System32\Oobe\Info\Default\Oobe.xml
3. %WINDIR%\System32\Oobe\Info\Default\<language>\Oobe.xml
4. %WINDIR%\System32\Oobe\Info\<country/region>\Oobe.xml
5. %WINDIR%\System32\Oobe\Info\<country/region>\<language>\Oobe.xml

If you have customizations that span all countries/regions and languages, the Oobe.xml files can be placed in Location 1.

If you're shipping a single-region, single-language system, your custom **Oobe.xml** file should be placed in the \Info (Location 1) or \Default (Location 2) directory. Those locations are functionally equivalent.

If you're shipping to multiple countries/regions and your OOBE settings require customizations for individual countries/regions, each with a single language, all of your **Oobe.xml** files should be placed in Locations 4 and 5.

If you're shipping to multiple countries/regions with multiple languages, the following guidelines apply:

- Place country/region-specific information in Location 4.
- Place language-specific information for each respective country/region in Location 5.

## Single-language deployments

If you're delivering PCs to one country/region in a single language, you should place a single **Oobe.xml** file in %WINDIR%\System32\Oobe\Info. This file can contain all of your customizations to the Windows first experience.

For example, an English version of Windows that's delivered to the United States can have the following directory structure:

\%WINDIR%\System32\Oobe\Info\Oobe.xml

If you're delivering PCs to more than one country/region in a single language, and you plan to vary your customizations in different locations, place an **Oobe.xml** file in %WINDIR%\System32\Oobe\Info.

This file can contain the default regional settings that you plan to show to the user. You should also include a default set of customizations, in case the user selects a country/region that you haven't made specific customizations for. The **Oobe.xml** file should also contain the <eulafilename> node with the name of the customized license terms that you plan to use.

Place an **Oobe.xml** file for each country/region that contains unique customized content in

\%WINDIR%\System32\<country/region that you're deploying to>\<language that you're deploying in>. After the user has chosen a country/region, these files are used to display additional customizations.

For example, an English version of Windows delivered to the United States and Canada can have the following directory structure:

\%WINDIR%\System32\Oobe\Info\Oobe.xml (EULA file name and regional settings)

\%WINDIR%\System32\Oobe\Info\244\1033\Oobe.xml (United States custom content)

\%WINDIR%\System32\Oobe\Info\39\1033\Oobe.xml (Canada custom content)

## Multiple-language or region deployments

If you're delivering PCs to one or more countries/regions and are delivering PCs running Windows with additional language packs, place an **Oobe.xml** file in \%WINDIR%\System32\Oobe\Info. This file can contain the default regional settings that you plan to show to the user. You should also include a default set of customizations, in case the user selects a country/region that you haven't made specific customizations for. This **Oobe.xml** should also contain the <eulafilename> node with the name of the custom license terms that you plan to use.

Place an **Oobe.xml** file for each country/region that contains unique customized content in \%WINDIR%\System32\<country/region that you're deploying to>\<language that you're deploying in>. After the user has chosen a country/region, this file is used to display additional customizations.

For example, an English version of Windows that's delivered to the United States and Canada would use the following directory structure:

\%WINDIR%\System32\Oobe\Info\Oobe.xml (logo, EULA file name, and regional settings)

\%WINDIR%\System32\Oobe\Info\244\1033\Oobe.xml (United States custom content)

\%WINDIR%\System32\Oobe\Info\39\1033\Oobe.xml (Canada custom content)

If you're delivering PCs to one or more countries/regions and are delivering PCs running Windows with additional language packs, place an **Oobe.xml** file in \%WINDIR%\System32\Oobe\Info. This **Oobe.xml** file should contain the <eulafilename> node with the name of the customized EULA that you plan to use.

Place an **Oobe.xml** for each Windows language that you're including in \%WINDIR%\System32\Default\<language that you're deploying in>. These files should contain the default regional settings that you plan to show for a given language, as well as a default set of customizations, in case the user selects a country/region that you haven't made specific customizations for.

Place an **Oobe.xml** file for each country/region that contains customized content in \%WINDIR%\System32\<country/region that you're deploying to>\<language that you're deploying in>. After the user has chosen a country/region, this file is used to display your additional customizations.

For example, a version of Windows with English and French language packs that's delivered to the United States and Canada would use the following directory structure:

- Logo and EULA:

\%WINDIR%\System32\Oobe\Info\Oobe.xml (logo and EULA file name)

- Regional settings and fallback for content that's not localized for the specific country/region:

\%WINDIR%\System32\Oobe\Info\Default\1033\Oobe.xml (default regional settings and English content if the user chooses a country/region other than the United States or Canada)

\%WINDIR%\System32\Oobe\Info\Default\1036\Oobe.xml (default regional settings and French content if the user chooses a country/region other than United States or Canada)

- Country-specific or region-specific content in the appropriate languages

**\%WINDIR%\System32\Oobe\Info\244\1033\Oobe.xml** (United States custom content in English)

**\%WINDIR%\System32\Oobe\Info\244\1036\Oobe.xml** (United States custom content in French)

**\%WINDIR%\System32\Oobe\Info\39\1033\Oobe.xml** (Canada custom content in English)

**\%WINDIR%\System32\Oobe\Info\39\1036\Oobe.xml** (Canada custom content in French)

### **Country/region folder format**

To identify the country/region:

1. Look up the country/region GeolD identifier using the [Table of Geographical Locations](#) on **MSDN**. These values are presented in hexadecimal.
2. Convert the value from hexadecimal to decimal, and use that value for the folder name. For example, to create a folder for Chile (GeolD 0x2E), name the folder "46".

**\%WINDIR%\System32\Oobe\Info\46\Oobe.xml**

### **Language folder format**

To identify the language, use the decimal version of the Locale ID (LCID) value. For example, to create a Spanish folder, name the folder "3082".

**%WINDIR%\System32\Oobe\Info\Default\3082\Oobe.xml**

There are many more LCIDs than languages. A few LCIDs correlate to the languages that can be released with Windows. For more information about which languages release with Windows, at what level of localization, and their decimal identifiers, see [Available Language Packs](#).

# Secure Boot

1/18/2019 • 2 minutes to read

This section covers how to work with Secure Boot in Windows.

## In this section

TOPIC	DESCRIPTION
<a href="#">Windows Secure Boot Key Creation and Management Guidance</a>	
<a href="#">Secure Boot Key Generation and Signing Using HSM (Example)</a>	
<a href="#">UEFI Validation Option ROM Validation Guidance</a>	
<a href="#">Disabling Secure Boot</a>	How to disable Secure Boot
<a href="#">Secure Boot isn't configured correctly: troubleshooting</a>	How to troubleshoot Secure Boot
<a href="#">BCD System Store Settings for UEFI</a>	BCD system store settings for UEFI
<a href="#">Validating Windows UEFI Firmware Update Platform Functionality</a>	How to validate Windows UEFI firmware update platform functionality

# Windows Secure Boot Key Creation and Management Guidance

1/24/2019 • 40 minutes to read

**Vishal Manan, Architect, OEM Consulting**, [vmanan@microsoft.com](mailto:vmanan@microsoft.com)

**Arie van der Hoeven, Architect, OEM Consulting**, [ariev@microsoft.com](mailto:ariev@microsoft.com)

This document helps guide OEMs and ODMs in creation and management of the Secure Boot keys and certificates in a manufacturing environment. It addresses questions related to creation, storage and retrieval of Platform Keys (PKs), secure firmware update keys, and third party Key Exchange Keys (KEKs).

**Note:** These steps are not specific to PC OEMs. Enterprises and customers can also use these steps to configure their servers to support Secure Boot.

Windows requirements for UEFI and Secure Boot can be found in the [Windows Hardware Certification Requirements](#). This paper does not introduce new requirements or represent an official Windows program. It is intended as guidance beyond certification requirements, to assist in building efficient and secure processes for creating and managing Secure Boot Keys. This is important because UEFI Secure Boot is based on the usage of Public Key Infrastructure to authenticate code before allowed to execute.

The reader is expected to know the fundamentals of UEFI, basic understanding of Secure Boot (Chapter 27 of the [UEFI specification](#)), and PKI security model.

Requirements, tests, and tools validating Secure Boot on Windows are available today through the [Windows Hardware Certification Kit \(HCK\)](#). However, these HCK resources do not address creation and management of keys for Windows deployments. This paper addresses key management as a resource to help guide partners through deployment of the keys used by the firmware. It is not intended as prescriptive guidance and does not include any new requirements.

On this page:

- [1. Secure Boot, Windows and Key Management](#) contains information on boot security and PKI architecture as it applies to Windows and Secure Boot.
- [2. Key Management Solutions](#) is intended to help partners design a key management and design solution that fits their needs.
- [3. Summary and Resources](#) includes appendices, checklists, APIs, and other references.

This document serves as a starting point in developing customer ready PCs, factory deployment tools and key security best practices.

## 1. Secure Boot, Windows and Key Management

The UEFI (Unified Extensible Firmware Interface) specification defines a firmware execution authentication process called Secure Boot. As an industry standard, Secure Boot defines how platform firmware manages certificates, authenticates firmware, and how the operating system interfaces with this process.

Secure Boot is based on the Public Key Infrastructure (PKI) process to authenticate modules before they are allowed to execute. These modules can include firmware drivers, option ROMs, UEFI drivers on disk, UEFI applications, or UEFI boot loaders. Through image authentication before execution, Secure Boot reduces the risk of pre-boot malware attacks such as rootkits. Microsoft relies on UEFI Secure Boot in Windows 8 and above as

part of its Trusted Boot security architecture to improve platform security for our customers. Secure Boot is required for Windows 8 and above client PCs, and for Windows Server 2016 as defined in the Windows Hardware Compatibility Requirements.

The Secure Boot process works as follows and as shown in Figure 1:

1. **Firmware Boot Components:** The firmware verifies the OS loader is trusted (Windows or another trusted operating system.)
2. **Windows boot components: BootMgr, WinLoad, Windows Kernel Startup.** Windows boot components verify the signature on each component. Any non-trusted components will not be loaded and instead will trigger Secure Boot remediation.
  - **Antivirus and Antimalware Software initialization:** This software is checked for a special signature issued by Microsoft verifying that it is a trusted boot critical driver, and will launch early in the boot process.
  - **Boot Critical Driver initialization:** The signatures on all Boot-critical drivers are checked as part of Secure Boot verification in WinLoad.
3. **Additional OS Initialization**

#### 4. Windows Logon Screen

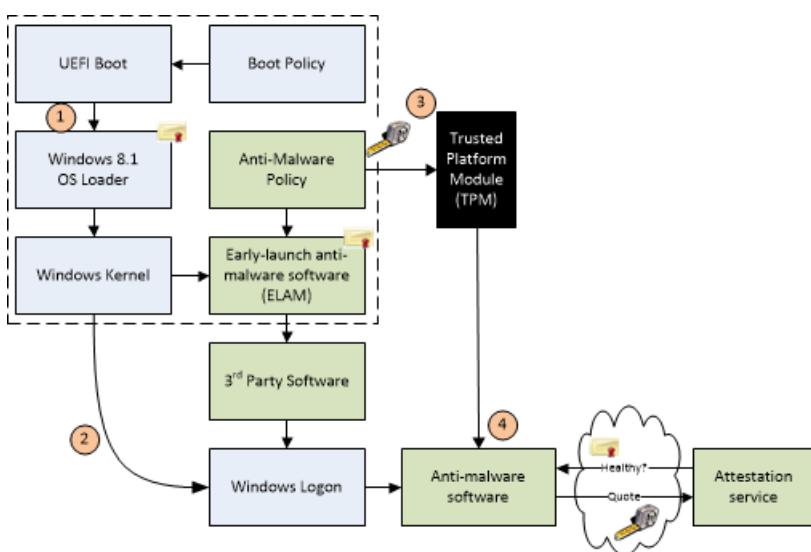


Figure 1: Windows Trusted Boot Architecture

Implementation of UEFI Secure Boot is part of Microsoft's Trusted Boot Architecture, introduced in Windows 8.1. A growing trend in the evolution of malware exploits is targeting the boot path as a preferred attack vector. This class of attack has been difficult to guard against, since antimalware products can be disabled by malicious software that prevents them from loading entirely. With Windows Trusted Boot architecture and its establishment of a root of trust with Secure Boot, the customer is protected from malicious code executing in the boot path by ensuring that only signed, certified "known good" code and boot loaders can execute before the operating system itself loads.

#### 1.1 Public-Key Infrastructure (PKI) and Secure Boot

The PKI establishes authenticity and trust in a system. Secure Boot leverages PKI for two high-level purposes:

1. During boot to determine if early boot modules are trusted for execution.
2. To authenticate requests to service requests include modification of Secure Boot databases and updates to platform firmware.

A PKI consists of:

- A certificate authority (CA) that issues the digital certificates.
- A registration authority which verifies the identity of users requesting a certificate from the CA.
- A central directory in which to store and index keys.
- A certificate management system.

## **1.2 Public Key Cryptography**

Public key cryptography uses a pair of mathematically related cryptographic keys, known as the public and private key. If you know one of the keys, you cannot easily calculate what the other one is. If one key is used to encrypt information, then only the corresponding key can decrypt that information. For Secure Boot, the private key is used to digitally sign code and the public key is used to verify the signature on that code to prove its authenticity. If a private key is compromised, then systems with corresponding public keys are no longer secure. This can lead to boot kit attacks and will damage the reputation of the entity responsible for ensuring the security of the private key.

In a Secure Boot public key system you have the following:

- **1.2.1 RSA 2048 Encryption**

RSA-2048 is an asymmetric cryptographic algorithm. The space needed to store an RSA-2048 modulus in raw form is 2048 bits.

- **1.2.2 Self-signed certificate**

A certificate signed by the private key that matches the public key of the certificate is known as a self-signed certificate. Root certification authority (CA) certificates fall into this category.

- **1.2.3 Certification Authority**

The certification authority (CA) issues signed certificates that affirm the identity of the certificate subject and bind that identity to the public key contained in the certificate. The CA signs the certificate by using its private key. It issues the corresponding public key to all interested parties in a self-signed root CA certificate.

In Secure Boot, Certification Authorities (CAs) include the OEM (or their delegates) and Microsoft. The CAs generate the key pairs that form the root of trust and then use the private keys to sign legitimate operations such as allowed early boot EFI modules and firmware servicing requests. The corresponding public keys are shipped embedded into the UEFI firmware on Secure Boot-enabled PCs and are used to verify these operations.

(More information on usage of CAs and key exchanges is readily available on the internet which relates to the Secure Boot model.)

- **1.2.4 Public Key**

The public Platform Key ships on the PC and is accessible or “public”. In this document we will use the suffix “pub” to denote public key. For example, PKpub denotes the public half of the PK.

- **1.2.5 Private Key**

For PKI to work the private key needs to be securely managed. It should be accessible to a few highly trusted individuals in an organization and located in a physically secure location with strong access policy restrictions in place. In this document we will use the suffix “priv” to denote private key. For example, the PKpriv indicates private half of the PK.

- **1.2.6 Certificates**

The primary use for digital certificates is to verify the origin of signed data, such as binaries etc. A common

use of certificates is for internet message security using Transport Layer Security (TLS) or Secure Sockets Layer (SSL). Verifying the signed data with a certificate lets the recipient know the origin of the data and if it has been altered in transit.

A digital certificate in general contains, at a high level, a distinguished name (DN), a public key, and a signature. The DN identifies an entity -- a company, for example -- that holds the private key that matches the public key of the certificate. Signing the certificate with a private key and placing the signature in the certificate ties the private key to the public key.

Certificates can contain some other types of data. For example, an X.509 certificate includes the format of the certificate, the serial number of the certificate, the algorithm used to sign the certificate, the name of the CA that issued the certificate, the name and public key of the entity requesting the certificate, and the CA's signature.

- **1.2.7 Chaining certificates**

From: [Certificate chains](#):



Figure 2: Three-certificate chain

User certificates are often signed by a different private key, such as a private key of the CA. This constitutes a two-certificate chain. Verifying that a user certificate is genuine involves verifying its signature, which requires the public key of the CA, from its certificate. But before the public key of the CA can be used, the enclosing CA certificate needs to be verified. Because the CA certificate is self-signed, the CA public key is used to verify the certificate.

A user certificate need not be signed by the private key of the root CA. It could be signed by the private key of an intermediary whose certificate is signed by the private key of the CA. This is an instance of a three-certificate chain: user certificate, intermediary certificate, and CA certificate. But more than one intermediary can be part of the chain, so certificate chains can be of any length.

### 1.3 Secure Boot PKI requirements

The UEFI-defined root of trust consists of the Platform Key and any keys an OEM or ODM includes in the firmware core. Pre-UEFI security and a root of trust are not addressed by the UEFI Secure Boot process, but instead by National Institute of Standards and Technology (NIST), and Trusted Computing Group (TCG) publications referenced in this paper.

- **1.3.1 Secure Boot requirements**

You'll need to consider the following parameters for implementing Secure Boot:

- Customer requirements
- Windows Hardware Compatibility requirements
- Key generation and management requirements.

You would need to pick hardware for Secure Boot key management like Hardware Security Modules (HSMs), consider special requirements on PCs to ship to governments and other agencies and finally the process of creating, populating and managing the life cycle of various Secure Boot keys.

- **1.3.2 Secure Boot related keys**

The keys used for Secure Boot are below:

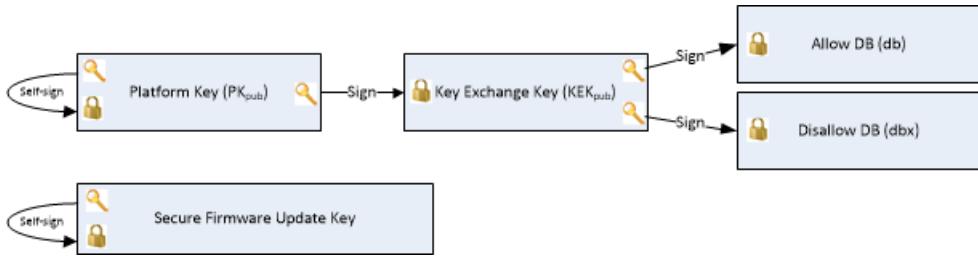


Figure 3: Keys related to Secure Boot

Figure 3 above represents the signatures and keys in a PC with Secure Boot. The platform is secured through a platform key that the OEM installs in firmware during manufacturing. Other keys are used by Secure Boot to protect access to databases that store keys to allow or disallow execution of firmware.

The authorized database (db) contains public keys and certificates that represent trusted firmware components and operating system loaders. The forbidden signature database (dbx) contains hashes of malicious and vulnerable components as well as compromised keys and certificates and blocks execution of those malicious components. The strength of these policies is based on signing firmware using Authenticode and Public Key Infrastructure (PKI). PKI is a well-established process for creating, managing, and revoking certificates that establish trust during information exchange. PKI is at the core of the security model for Secure Boot.

Below are more details on these keys.

- **1.3.3 Platform Key (PK)**

As per section 27.5.1 of the UEFI 2.3.1 Errata C, the platform key establishes a trust relationship between the platform owner and the platform firmware. The platform owner enrolls the public half of the key (PK<sub>pub</sub>) into the platform firmware as specified in **Section 7.2.1 of the UEFI 2.3.1 Errata C**. This step moves the platform into user mode from setup mode. Microsoft recommends that the Platform Key be of type **EFI\_CERT\_X509\_GUID** with public key algorithm RSA, public key length of 2048 bits, and signature algorithm sha256RSA. The platform owner may use type **EFI\_CERT\_RSA2048\_GUID** if storage space is a concern. Public keys are used to check signatures as described earlier in this document. The platform owner can later use the private half of the key (PK<sub>priv</sub>):

- To change platform ownership you must put the firmware into UEFI defined **setup mode** which disables Secure Boot. Revert to setup mode only if there is a need to do this during manufacturing.
- For desktop PC, OEMs manage PK and necessary PKI associated with it. For Servers, OEMs by default manage PK and necessary PKI. Enterprise customers or Server customers can also customize PK, replacing the OEM-trusted PK with a custom-proprietary PK to lock down the trust in UEFI Secure Boot firmware to itself.

#### **1.3.3.1 To enroll or update a Key Exchange Key (KEK) Enrolling the Platform Key**

The platform owner enrolls the public half of the Platform Key (**PK<sub>pub</sub>**) by calling the UEFI Boot Service SetVariable() as specified in Section 7.2.1 of UEFI Spec 2.3.1 errata C, and resetting the platform. If the platform is in setup mode, then the new **PK<sub>pub</sub>** shall be signed with its **PK<sub>priv</sub>** counterpart. If the platform is in user mode, then the new **PK<sub>pub</sub>** must be signed with the current **PK<sub>priv</sub>**. If the PK is of type **EFI\_CERT\_X509\_GUID**, then this must be signed by the immediate **PK<sub>priv</sub>**, not a private key of any certificate issued under the PK.

#### **1.3.3.2 Clearing the Platform Key**

The platform owner clears the public half of the Platform Key (**PK<sub>pub</sub>**) by calling the UEFI Boot Service SetVariable() with a variable size of 0 and resetting the platform. If the platform is in setup mode, then the empty variable does not need to be authenticated. If the platform is in user mode, then the empty variable must be signed with the current **PK<sub>priv</sub>**; see Section 7.2(Variable Services) under [UEFI specification 2.3.1](#)

Errata C for details. It is strongly recommended that the production PKpriv never be used to sign a package to reset the platform since this allows Secure Boot to be disabled programmatically. This is primarily a pre-production test scenario.

The platform key may also be cleared using a secure platform-specific method. In this case, the global variable Setup Mode must also be updated to 1.

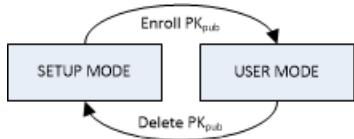


Figure 4: Platform Key State diagram

### 1.3.3.3 PK generation

As per UEFI recommendations, the public key must be stored in non-volatile storage which is tamper and delete resistant on the PC. The Private keys stay secure at Partner or in the OEM's Security Office and only the public key is loaded onto the platform. There are more details under section 2.2.1 and 2.3.

The number of PK generated is at the discretion of the Platform owner (OEM). These keys could be:

1. **One per PC.** Having one unique key for each device. This may be required for government agencies, financial institutions, or other server customers with high-security needs. It may require additional storage and crypto processing power to generate private and public keys for large numbers of PCs. This adds the complexity of mapping devices with their corresponding PK when pushing out firmware updates to the devices in the future. There are a few different HSM solutions available to manage large number of keys based on the HSM vendor. For more info, see [Secure Boot Key Generation Using HSM](#).
2. **One per model.** Having one key per PC model. The tradeoff here is that if a key is compromised all the machines within the same model would be vulnerable. This is recommended by Microsoft for desktop PCs.
3. **One per product line.** If a key is compromised a whole product line would be vulnerable.
4. **One per OEM.** While this may be the simplest to set up, if the key is compromised, every PC you manufacture would be vulnerable. To speed up operation on the factory floor, the PK and potentially other keys could be pre-generated and stored in a safe location. These could be later retrieved and used in the assembly line. Chapters 2 and 3 have more details.

### 1.3.3.4 Rekeying the PK

This may be needed if the PK gets compromised or as a requirement by a customer that for security reasons may decide to enroll their own PK.

Rekeying could be done either for a model or PC based on what method was selected to create PK. All the newer PCs will get signed with the newly created PK.

Updating the PK on a production PC would require either a variable update signed with the existing PK that replaces the PK or a firmware update package. An OEM could also create a SetVariable() package and distribute that with a simple application such as PowerShell that just changes the PK. The firmware update package would be signed by the secure firmware update key and verified by firmware. If doing a firmware update to update the PK, care should be taken to ensure the KEK, db, and dbx are preserved.

On all PCs, it is recommended to not use the PK as the secure firmware update key. If the PKpriv is compromised then so is the secure firmware update key (since they are the same). In this case the update to enroll a new PKpub might not be possible since the process of updating has also been compromised.

On SOCs PCs, there is another reason to not use the PK as the secure firmware update key. This is because the secure firmware update key is permanently burnt into fuses on PCs that meet Windows Hardware Certification requirements.

- **1.3.4 Key Exchange Key (KEK)** Key exchange keys establish a trust relationship between the operating system and the platform firmware. Each operating system (and potentially, each 3rd party application which need to communicate with platform firmware) enrolls a public key (**KEKpub**) into the platform firmware.

#### 1.3.4.1 Enrolling Key Exchange Keys

Key exchange keys are stored in a signature database as described in [1.4 Signature Databases \(Db and Dbx\)](#). The signature database is stored as an authenticated UEFI variable.

The platform owner enrolls the key exchange keys by either calling SetVariable() as specified in Section 7.2(Variable Services) under [UEFI specification](#) 2.3.1 Errata C. with the EFI\_VARIABLE\_APPEND\_WRITE attribute set and the Data parameter containing the new key(s), or by reading the database using GetVariable(), appending the new key exchange key to the existing keys and then writing the database using SetVariable() as specified in Section 7.2(Variable Services) under [UEFI specification](#) 2.3.1 Errata C without the EFI\_VARIABLE\_APPEND\_WRITE attribute set.

If the platform is in setup mode, the signature database variable does not need to be signed but the parameters to the SetVariable() call shall still be prepared as specified for authenticated variables in Section 7.2.1. If the platform is in user mode, the signature database must be signed with the current PKpriv

#### 1.3.4.2 Clearing the KEK

It is possible to "clear" (delete) the KEK. Note that if the PK is not installed on the platform, "clear" requests are not required to be signed. If they are signed, then to clear the KEK requires a PK-signed package, and to clear either db or dbx requires a package signed by any entity present in the KEK.

#### 1.3.4.3 Microsoft KEK

The Microsoft KEK is required to enable revocation of bad images by updating the dbx and potentially for updating db to prepare for newer Windows signed images.

Include the Microsoft Corporation KEK CA 2011 in the KEK database, with the following values:

- SHA-1 cert hash: `31 59 0b fd 89 c9 d7 4e d0 87 df ac 66 33 4b 39 31 25 4b 30`.
- SignatureOwner GUID: `{77fa9abd-0359-4d32-bd60-28f4e78f784b}`.
- Microsoft will provide the certificate to partners and it can be added either as an **EFI\_CERT\_X509\_GUID** or an **EFI\_CERT\_RSA2048\_GUID** type signature.

The Microsoft KEK certificate can be downloaded from: <https://go.microsoft.com/fwlink/?LinkId=321185>.

**1.3.4.4 KEKDefault** The platform vendor may provide a default set of Key Exchange Keys in the KEKDefault variable. Please reference [UEFI specification](#) section 27.3.3 for more information.

#### 1.3.4.5 OEM/3rd party KEK - adding multiple KEK

Customers and Platform Owners don't need to have their own KEK. On non-Windows RT PCs the OEM may have additional KEKs to allow additional OEM or a trusted 3rd party control of the db and dbx.

- **1.3.5 Secure Boot firmware update key** The Secure firmware update key is used to sign the firmware when it needs to be updated. This key has to have a minimum key strength of RSA-2048. All firmware updates must be signed securely by the OEM, their trusted delegate such as the ODM or IBV (Independent BIOS Vendor), or by a secure signing service.

As per [NIST publication 800-147 Field Firmware Update](#) must support all elements of guidelines:

Any update to the firmware flash store must be signed by creator.

Firmware must check signature of the update.

- **1.3.6 Creation of keys for Secure Firmware Update**

The same key will be used to sign all firmware updates since the public half will be residing on the PC. You could also sign the firmware update with a key which chains to Secure Firmware update key.

There could be one key per PC like PK or one per model or one per product line. If there is one key per PC that would mean that millions of unique update packages will need to be generated. Please consider based on resource availability what method would work for you. Having a key per model or product line is a good compromise.

The Secure Firmware Update public key (or its hash to save space) would be stored in some protected storage on the platform – generally protected flash (PC) or one-time-programmable fuses (SOC).

If only the hash of this key is stored (to save space), then the firmware update will include the key, and the first stage of the update process will be verifying that the public key in the update matches the hash stored on the platform.

Capsules are a means by which the OS can pass data to UEFI environment across a reboot. Windows calls the UEFI UpdateCapsule() to deliver system and PC firmware updates. At boot time prior to calling ExitBootServices(), Windows will pass in any new firmware updates found in the Windows Driver Store into UpdateCapsule(). UEFI system firmware can use this process to update system and PC firmware. By leveraging this Windows firmware support an OEM can rely on the same common format and process for updating firmware for both system and PC firmware. Firmware must implement the ACPI ESRT table in order to support UEFI UpdateCapsule() for Windows.

For details on implementing support for the Windows UEFI Firmware Update Platform consult the following documentation: [Windows UEFI Firmware Update Platform](#).

Update capsules can be in memory or on the disk. Windows supports in memory updates.

- **1.3.6.1 Capsule (Capsule-in-Memory)**

Following is the flow of events for an In-memory update capsule to work.

1. A capsule is put in memory by an application in the OS
2. Mailbox event is set to inform BIOS of pending update
3. PC reboots, verifies the capsule image and update is performed by the BIOS

- **1.3.7 Workflow of a typical firmware update**

1. Download and install the firmware driver.
2. Reboot.
3. OS Loader detects and verifies the firmware.
4. OS Loader passes a binary blob to UEFI.
5. UEFI performs the firmware update (This process is owned by the silicon vendor).
6. OS Loader detection completes successfully.
7. OS finishes booting.

## 1.4 Signature Databases (Db and Dbx)

- **1.4.1 Allowed Signature database (db)**

The contents of the EFI\_IMAGE\_SECURITY\_DATABASE db control what images are trusted when verifying loaded images. The database may contain multiple certificates, keys, and hashes in order to identify allowed images.

The Microsoft Windows Production PCA 2011 with a SHA-1 Cert Hash of

`58 0a 6f 4c c4 e4 b6 69 b9 eb dc 1b 2b 3e 08 7b 80 d0 67 8d` must be included in db in order to allow the

Windows OS Loader to load. The Windows CA can be downloaded from here:

<https://go.microsoft.com/fwlink/?LinkId=321192>.

On non-Windows RT PCs the OEM should consider including the **Microsoft Corporation UEFI CA 2011** with a SHA-1 Certificate Hash of `46 de f6 3b 5c e6 1c f8 ba 0d e2 e6 63 9c 10 19 d0 ed 14 f3`. Signing UEFI drivers and applications with this certificate will allow UEFI drivers and applications from 3rd parties to run on the PC without requiring additional steps for the user. The UEFI CA can be downloaded from here: <https://go.microsoft.com/fwlink/?LinkId=321194>.

On non-Windows RT PCs the OEM may also have additional items in the db to allow other operating systems or OEM-approved UEFI drivers or apps, but these images must not compromise the security of the PC in any way.

- **1.4.2 DbDefault:** The platform vendor may provide a default set of entries for the Signature Database in the dbDefault variable. For more information see section 27.5.3 in the UEFI specification.

- **1.4.3 Forbidden Signature Database (dbx)**

The contents of **EFI\_IMAGE\_SIGNATURE\_DATABASE1** dbx must be checked when verifying images before checking db and any matches must prevent the image from executing. The database may contain multiple certificates, keys, and hashes in order to identify forbidden images. The Windows Hardware Certification Requirements state that a dbx must be present, so any dummy value, such as the SHA-256 hash of `0`, may be used as a safe placeholder until such time as Microsoft begins delivering dbx updates. [Click Here](#) to download the latest UEFI revocation list from Microsoft.

- **1.4.4 DbxDefault:** The platform vendor may provide a default set of entries for the Signature Database in the dbxDefault variable. For more information see section 27.5.3 in the UEFI specification.

## 1.5 Keys Required for Secure Boot on all PCs

KEY/DB NAME	VARIABLE	OWNER	NOTES
PKpub	PK	OEM	PK – 1 only. Must be RSA 2048 or stronger.
Microsoft Corporation KEK CA 2011	KEK	Microsoft	Allows updates to db and dbx: <a href="https://go.microsoft.com/fwlink/?LinkId=321195">https://go.microsoft.com/fwlink/?LinkId=321195</a> .
Microsoft Windows Production CA 2011	db	Microsoft	This CA in the Signature Database (db) allows Windows to boot: <a href="https://go.microsoft.com/fwlink/?LinkId=321192">https://go.microsoft.com/fwlink/?LinkId=321192</a> .

KEY/DB NAME	VARIABLE	OWNER	NOTES
Forbidden Signature Database	dbx	Microsoft	List of known bad Keys, CAs or images from Microsoft
Secure firmware update key		OEM	Recommendation is to have this key be different from PK

Table 1: Keys/db needed for Secure Boot

## 2. Key Management Solutions

Below are given some of the metrics we used for comparison.

### 2.1 Metrics used

The following metrics can help you select a HSM PC based on the requirements of [UEFI specification](#) 2.3.1 Errata C and your needs.

#### Public Key Infrastructure (PKI) related

- Does it support RSA 2048 or higher? - The [UEFI specification](#) 2.3.1 Errata C recommends the keys to be RSA-2048 or better.
- Does it have the ability to generate keys and sign?
- How many keys can it store? Does it store keys on HSM or an attached server?
- Authentication method for key retrieval.

Some PCs support multiple authentication entities to be present for key retrieval.

#### Pricing

- What is the price point? HSMs can range in price from \$1,500 to \$70,000 depending on available features.

#### Manufacturing environment

- Speed of operation on factory floor. Crypto processors can speed up key creation and access.
- Ease of setup, deployment, maintenance.
- Skillset and training required?
- Network access for backup and High Availability

#### Standards and Compliance

- What level of FIPS compliance does it have? Is it tamper resistant?
- Support for other standards, for example, MS crypto APIs.
- Does it meet government and other agency requirements?

#### Reliability and disaster recovery

- Does it allow for Key Backup?

Backups can be stored both onsite in a safe location that is a different physical location than the CA

computer and HSM and /or at an offsite location.

- Does it allow for High Availability for disaster recovery?

## 2.2 Key Management Options

- **2.2.1 Hardware Security Module (HSM)**

Based on the above criteria this is probably the most suitable and secure solution. Most HSM have FIPS 140-2 level 3 compliance. FIPS 140-2 level 3 compliance is strict on authentication and requires that keys are not exported or imported from the HSM.

They support multiple ways of key storage. They could be stored either locally on the HSM itself or on the server attached to the HSM. On the server the keys are encrypted and stored and is preferable for solutions which require lots of keys to be stored.

The cryptographic module security policy shall specify a physical security policy, including physical security mechanisms that are implemented in a cryptographic module such as, tamper-evident seals, locks, tamper response and zeroization switches, and alarms. It also allows specifying actions required by the operator(s) to ensure that physical security is maintained such as periodic inspection of tamper-evident seals or testing of tamper response and zeroization switches.

### 2.2.1.1 Network HSM

This solution is the best in its class in terms of security, adherence to standards, key generation, storage and retrieval. Most of these PCs support high availability and have ability to backup keys.

The costs of these products can be in tens of thousands of dollars based on the extra services they offer.

### 2.2.1.2 Standalone HSM

These work great with standalone servers. One can use Microsoft CAPI and CNG or any other secure API supported by HSM. These HSMs come in variety of form factors supporting USB, PCIe and PCMCIA buses.

They optionally support key backup and high availability.

- **2.2.2 Custom solutions providers**

Public Key cryptography can be challenging and require understanding of cryptographic concepts which maybe new. There are custom solution providers who could help with the getting Secure Boot to work in the manufacturing environment.

There are varieties of custom solutions offered by BIOS vendors, HSM companies and PKI consulting companies to get Secure Boot PKI working in the manufacturing environment.

Some of the providers are listed below:

### 2.2.2.1 BIOS vendors

There are some BIOS vendors which may be able to provide custom solutions.

### 2.2.2.2 HSM vendors

Some HSM vendors may be able to provide custom consulting. For more info, see [Secure Boot Key Generation and Signing Using HSM \(Example\)](#).

- **2.2.3 Trusted Platform Module (TPM)**

A Trusted Platform Module (TPM) is a hardware chip on the motherboard that stores cryptographic keys used for encryption. Many computers include a TPM, but if the PC doesn't include it, it is not feasible to add one. Once enabled, the Trusted Platform Module can help secure full disk encryption products such as

Microsoft BitLocker capabilities. It keeps hard drives locked, or sealed, until the PC completes a system verification or authentication process.

The TPM can generate, store, and protect keys used in the encryption and decryption process.

The drawbacks of TPMs are that it may not have fast crypto processors to speed up processing in the manufacturing environment. They also are not suitable for storing large number of keys. Backup and high availability and standards compliance to FIPS 140-2 level 3 may not be available.

- **2.2.4 Smart Cards**

A smart card can generate and store keys. They do share some features which HSM support like authentication and tamper proofing, but they don't include much key storage or backup. They require manual intervention and may not be suitable for automation and use in production environment as the performance maybe low.

The drawbacks of Smart cards are similar to TPMs. They may not have fast crypto processors to speed up processing in the manufacturing environment. They also are not suitable for storing large number of keys. Backup and high availability and standards compliance to FIPS 140-2 level 3 may not be available.

- **2.2.5 Extended Validation Certificate**

EV Certificates are high assurance certificates whose private keys are stored in hardware tokens. This helps establish stronger key management practices. EV certificates have the same drawbacks as Smart cards.

- **2.2.6 Software-centric approaches (NOT RECOMMENDED)**

Use crypto APIs for key management. This may involve storing a key in a key container on an encrypted hard drive and possible for additional sandboxing and security use a Virtual machine.

These solutions are not as secure as using an HSM and expose a higher attack vector.

#### **2.2.6.1 Makecert (NOT RECOMMENDED)**

Makecert is a Microsoft tool and can be used as follows for key generation. To make sure that the attack surface is minimized you may need to "air gap" the PC. The PC that has the PKpriv on should not be connected to the network. It should be in a secure location and ideally should at least use a smart card reader if not a real HSM.

```
makecert -pe -ss MY -$ individual -n "CN=your name here" -len 2048 -r
```

For more info, see [Certificate Creation Tool \(Makecert.exe\)](#).

This solution is not recommended.

### **2.3 HSM Key generation and storage for Secure Boot keys**

- **2.3.1 Storing Private keys**

The space requirement for each RSA-2048 key is 2048 bits. The actual location of the storage of the keys depends on the solution chosen. HSM are a good way of storing keys.

The physical location of the PCs on the factory floor would need to be a protected area with limited user access like a secure cage.

Depending on your requirements these keys could also be stored in a diverse geographical location or backed up in a different location.

The rekeying requirements for these keys could vary based on the customer (see Appendix A for Federal bridge certificate authority rekeying guidelines).

These could be done once per year. You may need to have access to these keys for up to 30 years (depending on the rekeying requirements etc.).

- **2.3.2 Retrieving the private Keys**

The keys may need to be retrieved for many reasons.

1. The PK may need to be retrieved to issue an updated PK due to it being compromised or to adhere to government /other agency regulations.
2. KEKpri will be used to update the db and dbx.
3. Secure firmware update key –pri will be used to sign newer updates.

- **2.3.3 Authentication**

As per FIPS 140-2 authentication is based on level of access.

### **Level 2**

Security Level 2 requires, at a minimum, role-based authentication in which a cryptographic module authenticates the authorization of an operator to assume a specific role and perform a corresponding set of services.

### **Level 3**

Security Level 3 requires identity-based authentication mechanisms, enhancing the security provided by the role-based authentication mechanisms specified for Security Level 2. A cryptographic module authenticates the identity of an operator and verifies that the identified operator is authorized to assume a specific role and perform a corresponding set of services.

PCs like HSM's support Security Level 3, which requires identity-based "k of m authentication". This means k entities are given access to the HSM with a token but at a given point at least k out of the m tokens need to be present for authentication to work to get access to private keys from HSM.

For example, you could have 3 out of 5 tokens should be authenticated to access HSM. Those members could be the security officers, transaction authorizer and/or members from Executive Management.

### **HSM Tokens**

You could have a policy on the HSM which require the token to be present:

- Locally
- Remotely
- Configured to be automated

As a good practice, please use a combination of token and per token password.

## **2.4 Secure Boot and 3rd party signing**

- **2.4.1 UEFI driver signing**

UEFI Drivers must be signed by a CA or key in the db as described elsewhere in the document, or have the hash of the driver image included in db. Microsoft will be providing a UEFI driver signing service similar to the WHQL driver signing service using the **Microsoft Corporation UEFI CA 2011**. Any drivers signed by this will run seamlessly on any PCs that include the Microsoft UEFI CA. It is also possible for an OEM to sign trusted drivers and include the OEM CA in the db, or to include hashes of the drivers in the db. In all cases a UEFI driver (Option ROM) shall not execute if it is not trusted in the db.

Any drivers that are included in the system firmware image do not need to be re-verified. Being part of the

overall system image provides sufficient assurance that the driver is trusted on the PC.

Microsoft has this made available to anyone who wants to sign UEFI drivers. This certificate is part of the Windows HCK Secure Boot tests. Follow [this blog] ([https://blogs.msdn.microsoft.com/windows\\_hardware\\_certification/2013/12/03/microsoft-uefi-ca-signing-policy-updates/](https://blogs.msdn.microsoft.com/windows_hardware_certification/2013/12/03/microsoft-uefi-ca-signing-policy-updates/)) to read more about UEFI CA signing policy and updates.

- **2.4.2 Boot loaders**

The Microsoft UEFI driver signing certificate can be used for signing other OSs. For example, Fedora's Linux boot loader will be signed by it.

This solution doesn't require any more certificates to be added to the key Db. In addition to being cost effective, it can be used for any Linux distribution. This solution would work for any hardware which supports Windows so it is useful for a wide range of hardware.

The UEFI-CA can be downloaded from here: <https://go.microsoft.com/fwlink/?LinkId=321194>. The following links have more information on Windows HCK UEFI signing and submission:

- [Windows Dev Center Hardware Dashboard](#)
- [Windows Certification Dashboard Administration](#)
- [UEFI Firmware Signing](#)
- [Windows Hardware Certification blog: UEFI signing CA update](#)

### 3. Summary and Resources

This section intends to summarize the above sections and show a step by step approach:

1. **Establish a secure CA or identify a partner to securely generate and store keys**

If you are not using a 3rd party solution:

- a. **Install and configure the HSM software on the HSM server.** Check your HSM reference manual for installation instructions. The server will either be connected to a standalone or network HSM.

For info about HSM configuration, see Section 2.2.1, 2.3 and Appendix C.

Most HSMs offer FIPS 140-2 level 2 and 3 compliance. Configure the HSM for either level 2 or level 3 compliance. Level 3 compliance has stricter requirements around authentication and key access and hence is more secure. Level 3 is recommended.

- b. **Configure HSM for High Availability, Backup and Authentication.** Check your HSM reference manual.

Follow HSM provider guidelines on setting up HSM for High Availability and backup.

Also, Network HSMs typically have multiple network ports to segregate traffic; allowing a server to communicate with network HSMs on a network separate from the regular production network.

Once team members who are part of the security team have been identified and tokens assigned to them. You will need to setup HSM hardware for k-of-m authentication.

- c. **Secure Boot Keys and certificate pre-generation.** See Sections 1.3 to 1.5

Use HSM APIs to pre-generate (generate in advance) the PK and Firmware Update Key and certificates.

Required - PK (recommend 1 per model), Firmware Update key (recommend 1 per model), Microsoft KEK, Db, Dbx  
NOTE: The Microsoft KEK, db, and dbx don't have to be generated by the OEM and are mentioned for completeness.  
Optional - OEM/3rd party KEK db, dbx and any other keys which would go into OEM Db.

2. **Apply a Windows image to the PC.**

3. **Install Microsoft db and dbx.** See Section 1.3.6 and [Appendix B – Secure Boot APIs](#).

- a. Install the **Microsoft Windows Production PCA 2011** into db.
- b. Install an empty dbx if Microsoft does not provide one. Windows will automatically update DBX to the latest DBX through Windows Update on first reboot.

**Note**

Use PowerShell cmdlets which are part of the Windows HCK tests or use methods provided by BIOS vendor.

4. **Install Microsoft KEK.** See Section 1.3.3.

Install Microsoft KEK into the UEFI KEK database

**Caution**

Use PowerShell cmdlets which are part of the Windows HCK tests or use methods provided by BIOS vendor.

5. **Optional step - OEM/3rd party secure boot components.** See Section 1.3.4 and 1.4.

- a. Identify if you have need for creating a OEM/3rd party KEK, db and dbx.
- b. Sign OEM/3rd party db and dbx with OEM/3rd party KEK(generated earlier) using HSM API.
- c. Install OEM/3rd party KEK, db and dbx.

6. **UEFI driver signing** – See Section 2.4.

If supporting add-in cards or other UEFI drivers/apps/bootloaders, install **Microsoft Corporation UEFI CA 2011** into UEFI db.

7. **Secure boot firmware update key** - See Section 1.3.5.

- a. Non-Windows RT PCs only: Install the Secure firmware update public key or its hash to save space.
- b. On SoC only, you may need to do something different, for example, burn Secure firmware update key: public or its hash.

8. **Enabling Secure Boot.** See [Appendix B – Secure Boot APIs](#).

- a. Install the OEM/ODM PKpub (certificate preferred, but key is okay) into the UEFI PK.
- b. Enroll the PK using Secure Boot API. The PC should be now enabled for Secure Boot.

**Note**

If you install the PK at the end, the MS KEK, db, dbx don't need to be signed – no SignerInfo must be present. This is a shortcut.

9. **Testing Secure Boot:** Execute any proprietary tests and Windows HCK tests as per instructions. See [Appendix B – Secure Boot APIs](#).

10. **Ship platform:** The PKpriv will likely never be used again, keep it safe.

11. **Servicing:** Future firmware updates are securely signed with the Secure Firmware Update "private" key

using the signing service.

### 3.1 Resources

Security Strategies White Paper - <https://go.microsoft.com/fwlink/?linkid=321288>

Windows HCK Submission -<https://go.microsoft.com/fwlink/?linkid=321287>

## Appendix A – Secure Boot PKI checklist for manufacturing

Below is a high-level checklist summarizing the steps needed for enabling Secure Boot on non-Windows RT PCs.

### Setting up Secure Boot

1. Define security strategy (identify threats, define proactive and reactive strategy) as per the white paper in section 4.
2. Identify security team as per the white paper in section 4.
3. Establish a secure CA or identify a partner (recommended solution) to securely generate and store keys.
4. Identify policy for how frequently you will be rekeying keys. This may depend on if you have any special customer requirements like governments or other agencies.
5. Have a contingency plan in case the Secure Boot Key is compromised.
6. Identify how many PK and other keys will you be generating as per section 1.3.3 and 1.5.

This will be based on customer base, key storage solution and security of PCs.

You can skip steps 7-8 if you are using the recommended solution of using a 3rd party for key management.

7. Procure server and hardware for key management. – network or standalone HSM per section 2.2.1. Consider whether you will need one or several HSMs for high availability and your key back up strategy.
8. Identify at least 3-4 team members who will have an authentication token for authentication on HSM.
9. Use HSM or 3rd party to pre-generate Secure Boot-related keys and certificates. The keys will depend on the PC type: SoC, Windows RT or non-Windows RT. For more info, see Sections 1.3 through 1.5.
10. Populate the firmware with the appropriate keys.
11. Enroll the Secure Boot Platform Key to enable Secure Boot. See Appendix B for more details.
12. Execute any proprietary tests and HCK Secure Boot tests as per instructions. See Appendix B for more details.
13. Ship the PC. The PKpriv will likely never be used again, keep it safe.

### Servicing (Updating firmware)

You may need to update firmware for several reasons such as updating an UEFI component or fixing Secure Boot key compromise or periodic rekeying of Secure Boot keys.

For more info, see Section 1.3.5 and section 1.3.6.

## Appendix B – Secure Boot APIs

### 1. Secure Boot API

The following APIs are related to UEFI/Secure Boot:

- a. [GetFirmwareEnvironmentVariableEx](#): Retrieves the value of the specified firmware environment variable.
- b. [SetFirmwareEnvironmentVariableEx](#): Sets the value of the specified firmware environment variable.
- c. [GetFirmwareType](#): Retrieves the firmware type.

## 2. Setting PK

Use the Set-SecureBootUEFI cmdlet to turn on Secure Boot. After your code sets the PK, system enforcement of Secure Boot does not take effect until the next reboot. Prior to the reboot, your code could call GetFirmwareEnvironmentVariableEx() or the PowerShell cmdlet: Get-SecureBootUEFI to confirm the contents of the Secure Boot databases.

## 3. Verification

You can use Msinfo32.exe or PowerShell cmdlets to check Secure Boot variable state. There is no WMI interface. You could also test by having someone insert an incorrectly-signed bootable USB stick (for example, from the Windows HCK Secure Boot Manual Logo Test) and verify that it fails to boot.

## 4. Secure Boot Powershell Cmdlets

- **Confirm-SecureBootUEFI**: Is UEFI Secure Boot "ON", True or False?  
SetupMode == 0 && SecureBoot == 1
- **Set-SecureBootUEFI**: Set or Append authenticated SecureBoot UEFI variables
- **Get-SecureBootUEFI**: Get authenticated SecureBoot UEFI variable values
- **Format-SecureBootUEFI**: Creates EFI\_SIGNATURE\_LISTs & EFI\_VARIABLE\_AUTHENTICATION\_2 serializations

## 5. Windows HCK and Secure Boot Instructions

The following steps apply to system tests and non-class driver PC tests.

- a. Disable Secure Boot protections.
  - Enter your BIOS configuration and disable Secure Boot.
- b. Install the HCK Client software.
- c. Run all of the Windows HCK tests, except for the following:
  - BitLocker TPM and Recovery password tests with PCR[7]
  - BitLocker TPM and Recovery password tests for ARM PCs with Secure Boot
  - Secure Boot Logo Test
  - Secure Boot Manual Logo Test
- d. Enter your BIOS configuration, enable Secure Boot, and restore Secure Boot to the Default configuration.
- e. Run the following BitLocker and Secure Boot tests:
  - BitLocker TPM and Recovery password tests with PCR[7]
  - BitLocker TPM and Recovery password tests for ARM PCs with Secure Boot
  - Secure Boot Logo Test (automated)

f. Enter the BIOS configuration and clear the Secure Boot configuration. This restores the PC to Setup Mode by deleting PK and other keys.

**Note**

Support for clearing is required for x86/x64 PCs.

g. Run the Secure Boot Manual Logo Test.

**Note**

Secure Boot requires Windows HCK signed or VeriSign drivers on non-Windows RT PCs

## 6. Windows HCK Secure Boot Logo Test (automated)

This test will check for proper out-of-box Secure Boot configuration. This includes:

- Secure Boot is Enabled.
- The PK is not a known, test PK.
- KEK contains the production Microsoft KEK.
- db contains the production Windows CA.
- dbx present.
- Many 1kB variables are created/deleted.
- A 32kB variable is created/deleted.

## 7. Windows HCK Secure Boot manual test folder layout

The Windows HCK Secure Boot Manual Logo test folder layout is described below:

- “\Test” folder has the following:
  - Manufacturing and Servicing Test
  - Programmatically Enable Secure Boot in test configuration
  - Servicing Tests
  - Append a cert to db, verify function
  - Append a hash to dbx, verify function
  - Append a cert to dbx, verify function
  - Append 600+ hashes to dbx, verify size
  - Programmatically change the PK
- “\Generate” folder has scripts which show the following:
  - How test certificates were created
  - The test certificates and private keys are included
  - How all of the tests were created
  - Turning certificates and hashes into signed packages
  - You can run this yourself, substitute your own certificates
- “\certs” folder has all of the certificates you need to boot Windows:

### Note

Please don't use the methodology used in "ManualTests\generate\TestCerts" to generate keys and certificates. This is meant for Windows HCK test purposes only. It uses keys which are stored on disk which is very insecure and not recommended. This is not meant for use in a production environment.

- ``ManualTests\example\OutOfBox`` folder has scripts which you can leverage for installation of Secure Boot on production PCs.

The "ManualTests\generate\tests\subcreate\_outofbox\_example.ps1" demonstrates how these examples were generated and have "TODO" sections when a partner can substitute their PK and other metadata.

## 8. Windows HCK UEFI signing and submission

The following links have more information:

- [Hardware Developer Center Dashboard](#)
- [UEFI Firmware Signing](#)
- [Windows Certification Dashboard Administration](#)
- [Windows Hardware Certification blog: Microsoft UEFI CA Signing policy updates](#)

## Appendix C – Federal Bridge Certification Authority Certificate Policy Assurance Mappings

### 1. Rudimentary

This level provides the lowest degree of assurance concerning identity of the individual. One of the primary functions of this level is to provide data integrity to the information being signed. This level is relevant to environments in which the risk of malicious activity is considered to be low. It is not suitable for transactions requiring authentication, and is generally insufficient for transactions requiring confidentiality, but may be used for the latter where certificates having higher levels of assurance are unavailable.

### 2. Basic

This level provides a basic level of assurance relevant to environments where there are risks and consequences of data compromise, but they are not considered to be of major significance. This may include access to private information where the likelihood of malicious access is not high. It is assumed at this security level that users are not likely to be malicious.

### 3. Medium

This level is relevant to environments where risks and consequences of data compromise are moderate. This may include transactions having substantial monetary value or risk of fraud, or involving access to private information where the likelihood of malicious access is substantial.

### 4. High

This level is appropriate for use where the threats to data are high, or the consequences of the failure of security services are high. This may include very high value transactions or high levels of fraud risk.

## Related topics

[Secure Boot Key Generation and Signing Using HSM \(Example\)](#)

[UEFI Validation Option ROM Validation Guidance](#)

## Secure Boot Overview

# Secure Boot Key Generation and Signing Using HSM (Example)

6/6/2019 • 12 minutes to read

Version 1.3

Here's an example of how to generate Secure Boot keys (PK and others) by using a hardware security module (HSM).

You'll need to know the Secure Boot Public Key Infrastructure (PKI). For more info, see [Windows 8.1 Secure Boot Key Creation and Management Guidance](#).

## Requirements

### Tools Needed

- certreq.exe – Available Inbox
- certutil.exe – Available Inbox
- SignTool.exe – Available in the latest Windows SDK

### Hardware Security Module (HSM)

The whitepaper demonstrates the key generation using examples from the nCipher (now Thales) PCI HSM model nC1003P/nC3023P/nC3033P and the SafeNet Luna HSMs. Most of the concepts apply to other HSM vendors as well.

For other HSMs, contact your manufacturer for additional instructions on how to tailor your approach with the HSM Cryptographic Service Provider (CSP).

## Approach

We use the Microsoft certificate creation tool: **certreq.exe** to generate the Secure Boot Platform Key (PK) and other keys needed for Secure Boot.

The certreq tool can be adapted to use an HSM by providing the Cryptographic Service Provider (CSP) to be the HSM.

### Find the Cryptographic Service Provider (CSP)

You can use either the certutil.exe tool or a tool used by the HSM to list the CSPs.

- **This example uses the certutil tool to show the CSPs on the Thales/nCipher HSM:**

```
C:\secureboot_training\certreq> certutil -csplist
Provider Name: Microsoft Base Cryptographic Provider v1.0
Provider Type: 1 - PROV_RSA_FULL

Provider Name: Microsoft Base DSS and Diffie-Hellman Cryptographic Provider
Provider Type: 13 - PROV_DSS_DH

Provider Name: Microsoft Base DSS Cryptographic Provider
Provider Type: 3 - PROV_DSS

Provider Name: Microsoft Base Smart Card Crypto Provider
Provider Type: 1 - PROV_RSA_FULL

Provider Name: Microsoft DH SChannel Cryptographic Provider
Provider Type: 18 - PROV_DH_SCHANNEL

Provider Name: Microsoft Enhanced Cryptographic Provider v1.0
Provider Type: 1 - PROV_RSA_FULL

Provider Name: Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider
Provider Type: 13 - PROV_DSS_DH

Provider Name: Microsoft Enhanced RSA and AES Cryptographic Provider
Provider Type: 24 - PROV_RSA_AES

Provider Name: Microsoft RSA SChannel Cryptographic Provider
Provider Type: 12 - PROV_RSA_SCHANNEL

Provider Name: Microsoft Strong Cryptographic Provider
Provider Type: 1 - PROV_RSA_FULL

Provider Name: Microsoft Software Key Storage Provider

Provider Name: nCipher Security World Key Storage Provider

Provider Name: Microsoft Smart Card Key Storage Provider
CertUtil: -csplist command completed successfully.
```

For the SHA-256 digesting algorithm, use the **CNG** provider:

"nCipher Security World Key Storage Provider". Legacy providers do not support SHA-256 and are not suitable for use with Secure Boot.

- **This example uses the built-in Thales/nCipher tool to list the CSP:**

```
C:\Program Files\nCipher\nfast\bin> cnglist --list-providers
Microsoft Primitive Provider
Microsoft Smart Card Key Storage Provider
Microsoft Software Key Storage Provider
Microsoft SSL Protocol Provider
nCipher Primitive Provider
nCipher Security World Key Storage Provider
```

For the SHA-256 digesting algorithm, use the **CNG** provider:

"nCipher Security World Key Storage Provider". Legacy providers do not support SHA-256 and are not suitable for use with Secure Boot.

- **This example uses the SafeNet Luna HSMs tool to list the CSP:**

```
C:\>certutil -csplist
-----
Provider Name: Luna Cryptographic Services for Microsoft Windows
Provider Type: 1 - PROV_RSA_FULL

Provider Name: Luna enhanced RSA and AES provider for Microsoft Windows
Provider Type: 24 - PROV_RSA_AES

Provider Name: Luna SChannel Cryptographic Services for Microsoft Windows
Provider Type: 12 - PROV_RSA_SCHANNEL

Provider Name: Microsoft Base Cryptographic Provider v1.0
Provider Type: 1 - PROV_RSA_FULL

Provider Name: Microsoft Base DSS and Diffie-Hellman Cryptographic Provider
Provider Type: 13 - PROV_DSS_DH

Provider Name: Microsoft Base DSS Cryptographic Provider
Provider Type: 3 - PROV_DSS

Provider Name: Microsoft Base Smart Card Crypto Provider
Provider Type: 1 - PROV_RSA_FULL

Provider Name: Microsoft DH SChannel Cryptographic Provider
Provider Type: 18 - PROV_DH_SCHANNEL

Provider Name: Microsoft Enhanced Cryptographic Provider v1.0
Provider Type: 1 - PROV_RSA_FULL

Provider Name: Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider
Provider Type: 13 - PROV_DSS_DH

Provider Name: Microsoft Enhanced RSA and AES Cryptographic Provider
Provider Type: 24 - PROV_RSA_AES

Provider Name: Microsoft RSA SChannel Cryptographic Provider
Provider Type: 12 - PROV_RSA_SCHANNEL

Provider Name: Microsoft Strong Cryptographic Provider
Provider Type: 1 - PROV_RSA_FULL

Provider Name: Microsoft Software Key Storage Provider

Provider Name: Microsoft Smart Card Key Storage Provider

Provider Name: SafeNet Key Storage Provider
CertUtil: -csplist command completed successfully.
```

For SHA-256 digest algorithm you will need to use a CNG provider – “SafeNet Key Storage Provider”.  
Legacy providers do not support SHA-256 and are not suitable for use with Secure Boot.

To generate the key:

```
certreq.exe -new request.inf PK.cer
```

Sample request.inf file:

```

[Version]
Signature= "$Windows NT$"
[NewRequest]
ValidityPeriod = Years
ValidityPeriodUnits = 6
Subject = "CN=Corporation TODO Platform Key,O=TODO Corporation,L=TODO_City,S=TODO_State,C=TODO_Country"
MachineKeySet = true
RequestType=Cert
Exportable = FALSE
HashAlgorithm = SHA256
KeyAlgorithm = RSA
KeyLength = 2048
KeyContainer = "PKContainer"
ProviderName = "nCipher Security World Key Storage Provider"
KeyUsage = 0xf0

```

Update the following values:

- **Subject:** Replace the TODO's with real data

```
"CN=Corporation TODO Platform Key,O=TODO Corporation,L=TODO_City,S=TODO_State,C=TODO_Country".
```

- **ValidityPeriod, ValidityPeriodUnits:** Use the validity period of 6 years. While a PK may only be valid for 2 years, the 6-year period allows for potential future servicing.
- **KeyContainer:** Enter the container id that you used to create the Key with the HSM. You may be asked to provide the tokens that you have used to create the Security World for the Thales HSM.

#### Validating certificate (self-signed)

Verify that the certificate has been generated correctly:

```
certutil -store -v my "<Certificate_serial_number_or_thumbprint>"
```

For example: `certutil -store -v my "7569d364a2e77b814274c81ae6360ffe"`

Sample output:

```

my
=====
X509 Certificate:
Version: 3
Serial Number: 7569d364a2e77b814274c81ae6360ffe
Signature Algorithm:
    Algorithm ObjectId: 1.2.840.113549.1.1.11 sha256RSA
    Algorithm Parameters:
        05 00
Issuer:
    CN=test self-signed

NotBefore: 1/21/2013 7:25 PM
NotAfter: 1/21/2015 7:35 PM

Subject:
    CN=test self-signed

Public Key Algorithm:
    Algorithm ObjectId: 1.2.840.113549.1.1.1 RSA (RSA_SIGN)
    Algorithm Parameters:
        05 00
Public Key Length: 2048 bits
Public Key: UnusedBits = 0
    0000 30 82 01 0a 02 82 01 01 00 cf e3 83 c7 a4 05 dd
    0010 be 05 76 b6 26 16 ae ba 0f a1 c6 3f 4f 58 11 2a
-----
```

```

0020  4c fe fc 44 f5 d2 11 36  75 c8 c9 90 15 d3 06 94
0030  18 ea 10 d8 4c 77 60 1f  45 75 25 6f 21 08 84 d3
0040  8f 6f 70 07 1b 3e eb 26  94 b8 aa 0d fd 0c 13 f1
0050  7f 76 0c 33 a4 ad b4 7a  f3 c1 f1 d8 c9 a0 ba d2
0060  c5 9e 2b ce 36 7e 34 9b  81 26 74 0b 32 47 48 48
0070  08 ab c0 e7 c3 a2 8e e4  1f b8 6f 38 a2 31 84 65
0080  75 67 db 01 fc 41 a8 98  83 ad ba 2f 4e 59 c3 6b
0090  93 84 e0 ab de bd 6f 8f  61 9b b3 42 b3 fb 19 f7
00a0  46 3a ad d7 e9 d1 fa 2b  a7 72 8d 76 ac 9f 6d c3
00b0  79 ba 37 e4 6d 72 b1 6f  22 82 80 77 a7 92 3f b7
00c0  e2 1f e0 c6 90 9a 82 ef  40 47 29 fb c3 83 7e 38
00d0  01 35 1f 66 6c 1b 93 0d  c2 fc 5c e2 4e bd e1 85
00e0  c3 7e a9 51 6f 57 82 86  37 79 92 63 b2 e0 42 4f
00f0  25 5c 1b 03 50 29 2d ee  40 31 c3 a1 c3 cf 62 31
0100  e0 8c 60 2f d4 34 56 f1  bf 02 03 01 00 01

Certificate Extensions: 2
 2.5.29.15: Flags = 1(Critical), Length = 4
    Key Usage
      Digital Signature, Non-Repudiation, Key Encipherment, Data Encipherment (f0)

  2.5.29.14: Flags = 0, Length = 16
    Subject Key Identifier
      5b 3b 53 ed e3 0f a9 48 90 e0 93 09 0f f9 7b 32 3a 8d 89 4f

Signature Algorithm:
  Algorithm ObjectId: 1.2.840.113549.1.1.11 sha256RSA
  Algorithm Parameters:
    05 00

Signature: UnusedBits=0
  0000  3c 08 5f e0 a7 42 2a bc  58 61 64 43 b6 f4 23 99
  0010  ca 58 b1 8c a3 6b eb 9c  31 a0 ce 25 3a d5 b4 74
  0020  c2 0c 9c 00 1e c8 0f d2  05 3d fc 5d 6f 17 cd ac
  0030  4d 14 9e d4 2b 45 1e ad  5f 5b ee 23 a8 29 65 b3
  0040  cd c4 fd 5c e6 6a bd 95  ce f0 f9 be 31 19 87 90
  0050  f8 86 c4 31 a8 b3 d5 b3  14 24 5b de f8 c0 f9 9c
  0060  96 a2 b5 89 39 41 bd 4b  5f 04 16 10 c0 5c b8 fb
  0070  1d 8d 64 b2 87 00 72 46  b9 5e d0 3a 75 8d ea 5a
  0080  f6 5d 9c c5 03 cd c8 54  b7 7a ef c8 3e 3f 4b f6
  0090  d2 c7 70 67 29 92 70 44  fc c6 2e c9 42 dd 6e 01
  00a0  c5 71 27 20 51 ed 34 3c  98 c2 bc 1f 57 16 71 86
  00b0  24 e3 0e 41 57 82 ba 41  df b5 6d f9 4d e4 72 80
  00c0  6f 8d ab 10 06 cd 69 6b  d0 82 ac db 04 da 6b a5
  00d0  83 14 1a a0 6d 90 c4 01  5d 24 68 ac 10 ca db 96
  00e0  44 8b ef f1 13 7f 22 15  32 93 4e 2d 23 ce 7f fb
  00f0  18 9f d0 1c c1 45 2c e6  bb 23 7f 9e 22 ea fc 88

Signature matches Public Key
Root Certificate: Subject matches Issuer
Key Id Hash(rfc-sha1): 5b 3b 53 ed e3 0f a9 48 90 e0 93 09 0f f9 7b 32 3a 8d 89 4f
Key Id Hash(sh1): 1e 07 bb 05 ce d2 db 9c 9f ab d1 46 b8 32 20 e3 41 dc 4c 08
Cert Hash(md5): 45 ab 9b e4 6e 91 53 b5 96 81 10 8e 01 45 6c 54
Cert Hash(sh1): 37 ed 7c 3e ee 76 a2 d0 42 3a e3 1a 16 9f 74 d0 3c 7f 34 2c

CERT_REQUEST_ORIGINATOR_PROP_ID(71):
  VM-DESKTEST.ntdev.corp.microsoft.com

CERT_KEY_PROV_INFO_PROP_ID(2):
  Key Container = PKContainer
  Provider = nCipher Security World Key Storage Provider
  ProviderType = 0
  Flags = 20
 KeySpec = 0

CERT_SHA1_HASH_PROP_ID(3):
  37 ed 7c 3e ee 76 a2 d0 42 3a e3 1a 16 9f 74 d0 3c 7f 34 2c

CERT SUBJECT PUBLIC KEY MD5 HASH PROP_ID(25):
  12 eb 13 79 64 61 08 e9 a6 75 f2 9a 5c 49 b4 f9

CERT_KEY_IDENTIFIER_PROP_ID(20):

```

```

5b 3b 53 ed e3 0f a9 48 90 e0 93 09 0f f9 7b 32 3a 8d 89 4f

CERT_SIGNATURE_HASH_PROP_ID(15):
    0000 38 c4 1b 14 d8 74 95 42 1b fb 7d 72 d2 0b 03 ad
    0010 bd e8 aa 19 14 9e a2 41 30 fe b4 d4 93 b6 9f 3b

CERT_MD5_HASH_PROP_ID(4):
    45 ab 9b e4 6e 91 53 b5 96 81 10 8e 01 45 6c 54
UI Policy = 0
Version: 0

PKContainer

Export Policy = 0
Key Usage = 3
NCRYPT_ALLOW_DECRYPT_FLAG -- 1
NCRYPT_ALLOW_SIGNING_FLAG -- 2

D:AI(A;ID;FA;;SY)(A;ID;FA;;BA)(A;ID;0x1200a9;;;BU)

Allow WriteNT AUTHORITY\SYSTEM
Allow WriteBUILTIN\Administrators
Allow WriteBUILTIN\Users

Private key is NOT exportable
Signature test passed
CertUtil: -store command completed successfully.

```

## Backing up the certificate

Back up your certificates. This way, if either the certificate store or the server goes down, you can add the certificate back to the store. For more info on certreq.exe, see [Advanced Certificate Enrollment and Management: Appendix 3: Certreq.exe Syntax](#)

Note, the PK is a self-signed certificate, and is also used to sign the KEK.

There are 2 parts to PK signing / initial provisioning. Please talk to your Microsoft contact to get these scripts:

- `subcreate_set_PK_example_initial_provisioning_example.ps1`. Used by the signtool to sign the PK comes later in the servicing case.
- `subcreate_set_PK_service_example.ps1`. Since we are dealing with the HSM case, the following line applies in the script applies.

## Signing with PK certificate (servicing scenario)

This section applies to signing with your PK certificate and may not be applicable for initial provisioning of system. However, you can use the method here to test your service scenario.

### Determine the certificate hash (sha1)

Determine the SHA1 hash of the certificate. You can get the SHA1 hash by using either of the following methods:

- In Windows, open the **Certificate** file, select the **Details** tab, and check the value for **Thumbprint**.
- Or use the following command:

```
C:\>certutil -store My PKContainer
```

Sample output:

```
My
=====
Serial Number: 58efcf8f929c5bd41152a8ec413051e
Issuer: CN=test self-signed
NotBefore: 1/30/2013 3:24 PM
NotAfter: 1/30/2019 3:34 PM
Subject: CN=test self-signed
Signature matches Public Key
Root Certificate: Subject matches Issuer
Template:
Cert Hash(sha1): db 31 4d a0 d0 ef 87 d4 2b 42 f7 4b 9c 38 a1 f9 17 3e f7 a2
Key Container = PKContainer
Provider = nCipher Security World Key Storage Provider
Private key is NOT exportable
Signature test passed
CertUtil: -store command completed successfully.
```

### Sign with signtool with the certificate store specified as a reference

Use the SHA1 hash to sign the KEK certificate:

```
C:\> signtool.exe sign /v /fd sha256 /sha1 "db314da0d0ef87d42b42f74b9c38a1f9173ef7a2" /sm /p7 .\ /p7co
1.2.840.113549.1.7.1 /p7ce DetachedSignedData KEK.bin
```

Where KEK.bin is the filename of the binary certificate you want to sign.

Sample output:

```
The following certificate was selected:
Issued to: test self-signed
Issued by: test self-signed
Expires: Fri Jan 30 15:34:32 2019
SHA1 hash: DB314DA0D0EF87D42B42F74B9C38A1F9173EF7A2

Done Adding Additional Store
Successfully signed: KEK.bin

Number of files successfully Signed: 1
Number of warnings: 0
Number of errors: 0
```

**NOTE** For compatibility with the UEFI Specification and maximum compatibility across UEFI implementations, the /p7co and /p7ce parameters **must** be present, the value passed to /p7co **must** be 1.2.840.113549.1.7.1, and the value passed to /p7ce **must** be DetachedSignedData. Also, for improved compatibility with production signing environments, a signtool.exe commandline that fully specifies the hardware key container is as follows:

```
signtool.exe sign /p7 .\ /p7co 1.2.840.113549.1.7.1 /p7ce DetachedSignedData /fd SHA256 /u
1.3.6.1.4.1.311.79.1 /f "{CertificatePath}" /kc "{KeyContainer}" /sha1 "{ThumbPrint}" /csp "{CertificateCSP}"
"{FilePath}"
```

For more info, see [Sign Tool \(SignTool.exe\)](#) and [Windows 8.1 Secure Boot Key Creation and Management Guidance](#).

## Appendix A – Using Thales KeySafe for viewing keys

Thales KeySafe is based on a GUI.

To use KeySafe, you must have installed JRE/JDK 1.4.2, 1.5, or 1.6. Install Java before you install the nCipher software.

Configure the hardserver config file under the `%NFAST_KMDATA%\config\` folder:

Edit settings in the `server_startup` section:

**nonpriv\_port.** This field specifies the port on which the hardserver listens for local non-privileged TCP connections.

- Default to connecting to port 9000.
- If the `NFAST_SERVER_PORT` environment variable is set, it overrides any value set for `nonpriv_port`

**priv\_port.** This field specifies the port on which the hardserver listens for local privileged TCP connections.

- Default to connecting to port 9001.
- If the `NFAST_SERVER_PRIVPORT` environment variable is set, it overrides any value set for `priv_port`

The following are screenshots from the Thales KeySafe GUI:

## Key Operations

The buttons on this panel enable you to create new keys for a wide range of applications, to list details of existing keys, or to import an external application key.

You can discard an existing key by clicking the List Keys button below, highlighting the appropriate entry, and then clicking the Discard Key button.

Before you can use a module to generate keys, you must have either:

- initialized a security world using the module
- reprogrammed the module with an existing security world.

Use the options in the Module Operations panel to either initialize a security world or reprogram a module. Click the Modules menu button on the sidebar in order to go to the Module Operations panel.

<code>Generate Key</code>	create a new application key
<code>List Keys</code>	list all keys in the current security world
<code>Import Key</code>	import an application key from an outside source

The following image is generated by launching the KeySafe utility and then navigating to the KeyList menu.

## Key Listing

Selecting a key from the list below displays that key's parameters.

You can then click the Remove Key button in order to remove the selected key from your security world, or you can make another selection.

Key List			
Key Name	Application	Protection	NVRAM
PK	PKCS#11	Module Protected	No
pk_cert	PKCS#11	Module Protected	No
test_pk11	PKCS#11	Module Protected	No
vmanan	Unknown ('mscapi')	No Key	No

For more info, see the nCipher/Thales Users Guide.

## Appendix B: Using SafeNet CMU Utility to view keys

For more details, please consult the SafeNet Luna HSM documentation.

## Related topics

Windows 8.1 Secure Boot Key Creation and Management Guidance

## Secure Boot Overview

# UEFI Validation Option ROM Guidance

1/18/2019 • 18 minutes to read

Vishal Manan, Architect, OEM Consulting, [vmanan@microsoft.com](mailto:vmanan@microsoft.com)

Jeremiah Cox, Sr. SDET, Windows Security & Identity Team, [jerecox@microsoft.com](mailto:jerecox@microsoft.com)

Tony Lin, Engineering Service Engineer, TW-WIN Plan Ecosystem, [tolin@microsoft.com](mailto:tolin@microsoft.com)

Version 1.3

This document helps OEMs and ODMs validate that their firmware checks the signatures of its option ROM as part of the Secure Boot chain of trust.

This guide assumes you know the fundamentals of UEFI, basic understanding of Secure Boot (Chapters 1, 2, 13, 20 and 27 of the UEFI specification), and PKI security model.

On this page:

- [Introduction](#)
- [1. UEFI and Option ROMs](#)
- [2. Problem statement](#)
- [3. Who is affected?](#)
- [4. How to test for it?](#)
- [5. How to fix it](#)
- [6. Resources](#)
- [Appendix A: Alternate approach to testing using unsigned option ROM drivers](#)
- [Appendix B: Scripts for enabling Secure Boot with NULL db](#)

## Introduction

Option ROMs (or OpROMs) are firmware run by the PC BIOS during platform initialization. They are usually stored on a plug-in card, though they can reside on the system board.

Devices that typically require option ROMs are video cards, network adapters, and storage drivers for RAID modules. These option ROMs also typically provide firmware drivers to the PC.

They include a variety of types of firmware drivers, including legacy PC-AT, Open Firmware, and EFI option ROMs. Examples of firmware drivers include Video BIOS on video cards, PXE boot drivers for Ethernet adapters, and storage drivers on RAID controllers. These devices typically have Option ROMs that provide firmware drivers.

The Unified Extensible Firmware Interface (UEFI) has support for Legacy mode option ROMs.

As per latest UEFI specification (currently at 2.3.1 Errata C – section 2.5.1.2), ISA (legacy) option ROMs are not a part of the UEFI Specification. For the purposes of this discussion, only PCI-based UEFI-compatible option ROMs will be considered.

Option ROMs can be used when it's not be possible to embed a device's firmware in the PC firmware. When the option ROM carries the driver, the IHV can leverage that driver, and keep the driver and device in one place.

This document talks about why you need to validate option ROMs and shows some techniques of doing it.

## Supporting both UEFI BIOS and Legacy BIOS

Many manufacturers create devices that include option ROMs and firmware for many types of PCs. Common combos include:

- Legacy ROM Only
- UEFI Native OpROM
- Legacy ROM + UEFI EBC OpROM
- Legacy ROM + UEFI x64 OpROM
- Legacy ROM + UEFI x64 + UEFI IA32
- Legacy ROM + UEFI x64 + UEFI IA32 + UEFI EBC OpROM

UEFI BIOS can load and execute legacy firmware drivers when a Compatibility Support Module (CSM) is enabled. Note that when Secure Boot is enabled, execution of the Compatibility Support Module and legacy ROMs is prohibited because legacy firmware drivers do not support authentication. If the Option ROM format in the BIOS configuration is set to legacy ROM, it will always use the legacy ROM on the device.

If the Option ROM format is set to **UEFI Compatible**, it will use the newer EFI ROM if one is present and the legacy ROM if one is not.

UEFI drivers are necessary for many of the new firmware level security features as well as to enable UEFI boot sequences. For example, installing Windows from an optical disk which is attached to a non-UEFI compatible storage controller is not possible when a system is booting in UEFI mode when Secure Boot is enabled.

## 1. UEFI and Option ROMs

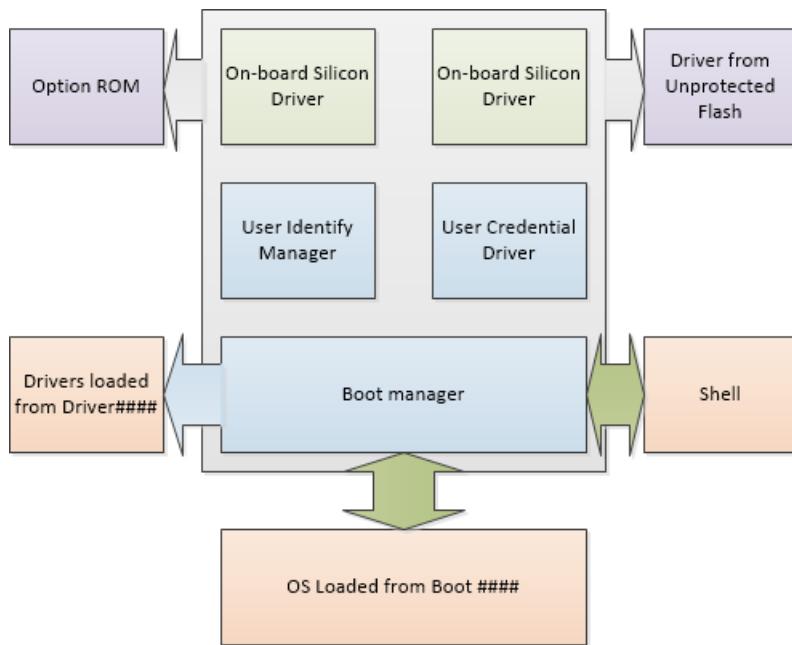


Figure 2: UEFI Driver Security Consideration, Source: UEFI 2.3.1 Errata C

**The following text originated in UEFI 2.3.1 Errata C, but has since been modified with insights from partners:**

Since the UEFI user profile details a number of security-related privileges, it is important that the User Identity Manager and User Credential Providers and the environment in which they execute are trusted.

This includes:

- Protecting the storage area where these drivers are stored.
- Protecting the means by which these drivers are selected.
- Protecting the execution environment of these drivers from unverified drivers.
- The data structures used by these drivers should not be corrupted by unauthorized drivers while they are still being used.

Components like User Identity Manager, the User Credential drivers and on board drivers maybe located in a secure location like write-protected flash drive which is trusted by platform policy.

Some other drivers may reside on an unprotected storage locations like option ROMs or a hard drive partition and may be easily replaced. These drivers must be verified.

For example, either the default platform policy must successfully be able to verify drivers listed in the Driver#### load options, or else the user must be identified prior to processing these drivers. Otherwise, the driver execution should be deferred. If the user profile is changed through a subsequent call to Identify () or through dynamic authentication, the Driver#### options may not be processed again.

The user profile database is closed using different UEFI signal events based on whether it can be protected.

UEFI Drivers & UEFI option ROMs will only be executed for devices in the boot path.

PCI spec allows multiple option ROM images on the same device. These option ROMS could be Legacy x86 & UEFI. UEFI firmware sets platform policy for picking the option ROM. That can make the optional adapter's ROM execute as its own control device.

The firmware verifies signatures during BDS and DXE phases. The sequence of events is as follows:

1. Initialize PCI and derivative Buses
2. Probe PCI Devices for option ROMs
3. Found option ROMs are mapped in memory
4. DXE phase loads any UEFI drivers in ROMs

UEFI option ROMs can be anywhere in memory. The default is to let the ROM on the card manage the device.

UEFI allows platform to control policy around what option ROM controls what device using

`EFI_PLATFORM_DRIVER_OVERRIDE`. UEFI supports option ROMs to register a configuration interface.

On a PC with Secure Boot enabled, option ROM drivers pose a security threat if they are not signed or not validated. Signature validation for option ROMs is a WHCK requirement. The same is true while servicing option ROMs to make sure that the update is validated prior to installation.

From the [Windows Hardware Compatibility Program specifications and policies version 1809](#):

10. **Signed Firmware Code Integrity Check.** Firmware that is installed by the OEM and is either read-only or protected by a secure firmware update process, as defined above, may be considered protected. Systems shall verify that all unprotected firmware components, UEFI drivers, and UEFI applications are signed using minimum RSA-2048 with SHA-256 (MD5 and SHA-1 are prohibited), and verify that UEFI applications and drivers that are not signed as per these requirements will fail to run (this is the default policy for acceptable signature algorithms). If an images signature is not found in the authorized database, or is found in the forbidden database, the image must not be started, and instead, information about it shall be placed in the Image Execution Information Table.

## 2. Problem statement

Some builds of Secure Boot-enabled UEFI BIOS, including Tiano Core, did not by default authenticate UEFI

option ROMs because signed UEFI option ROMs were not available during Secure Boot development. This exposes an attack surface/vulnerability in UEFI Secure Boot.

## 2.1. Vulnerability

This vulnerability was still present in EDK II and UDK2010 as of August 2013. The source maintainers are aware of the issue and a bug is filed. Any firmware derived from EDK II and UDK2010 should verify how Option ROM verification is managed. Option ROM verification behavior is controlled by a PCD value

`PcdOptionRomImageVerificationPolicy` in the EDK II SecurityPkg package.

The source code for the TianoCore vulnerability is SecurityPkg\SecurityPkg.dec file:

```
## Pcd for OptionRom.  
# Image verification policy settings:  
# ALWAYS_EXECUTE 0x00000000  
# NEVER_EXECUTE 0x00000001  
# ALLOW_EXECUTE_ON_SECURITY_VIOLATION 0x00000002  
# DEFER_EXECUTE_ON_SECURITY_VIOLATION 0x00000003  
# DENY_EXECUTE_ON_SECURITY_VIOLATION 0x00000004  
# QUERY_USER_ON_SECURITY_VIOLATION 0x00000005  
gEfiSecurityPkgTokenSpaceGuid.PcdOptionRomImageVerificationPolicy|0x00|UINT32|0x00000001
```

The default value (0x00) is ALWAYS\_EXECUTE, which does not properly perform verification of signed drivers in Option ROMs for add-in peripherals. This is not an ideal value for any system implementing UEFI Secure Boot functionality.

Recommended Value (Best Security): `DENY_EXECUTE_ON_SECURITY_VIOLATION (0x04)`

Recommended Value (Best Flexibility): `QUERY_USER_ON_SECURITY_VIOLATION (0x05)`

In EDK II & UDK2010, proper coding practice uses an override mechanism to modify PCD values for platform firmware. Therefore, the value for `PcdOptionRomImageVerificationPolicy` should not be changed in `SecurityPkg\SecurityPkg.dec`. The override value should be set in the platform's DSC file. An example is shown below using Nt32Pkg\Nt32Pkg.dsc:

```
[PcdsFixedAtBuild]  
gEfiSecurityPkgTokenSpaceGuid.PcdOptionRomImageVerificationPolicy|0x04
```

The PCD override should be placed under the `[PcdsFixedAtBuild]` section of the DSC file. The exact mechanism for overriding parameters may differ depending on BIOS vendor tools.

**Note** This vulnerability may exist in early implementations of UEFI Secure Boot BIOS from independent BIOS vendors. Contact your BIOS vendor to determine if your version may be impacted.

## 3. Who is affected?

A UEFI PC which implements Secure Boot and has a UEFI option ROM driver which is not signed. Furthermore, the firmware for compatibility to get the existing cards working may have a security vulnerability which doesn't verify option ROMs.

**Laptops, netbooks, ultrabooks, & tablets: most are not affected.** Option ROMs are typically present on backplane buses such as PCI/e, ISA, and their derivatives (ExpressCard, miniPCI, CardBus, PCCard, LPC, ThunderBolt etc). If a laptop has none of these exposed, then its attack surface is greatly reduced. Moreover, it is likely UEFI drivers for onboard laptop components are integrated into the core BIOS firmware volume, not located on a separate option ROM. Thus most laptops are not at risk. Also, when Legacy option ROMs are disabled, it looks like UEFI only supports PCI-based option ROMs.

However, if you have a desktop, motherboard or a server which has a UEFI BIOS and implement Secure Boot, you

may be affected. On a server's dedicated RAID controller, or add-in storage controller for SATA, FC etc. or Ethernet PCIe network cards may have option ROMs. Add-in controllers supporting a wide array of functionality on servers are common so this especially applies to the server space.

This can potentially affect 32-bit and 64-bit PCs, both class 2 and class 3.

If a Secure Boot platform supports option ROMs from devices not permanently attached to the platform and it supports the ability to authenticate those option ROMs, then it must support the option ROM validation methods described in Network Protocols — UDP and MTFTP and the authenticated EFI variables described in UEFI specification 2.3.1 Errata C Section 7.2.

## 4. How to test for it?

If you are developing the firmware and it is based on Tiano Core please check for vulnerability mentioned in the section 2.1. If you are using another IBV's firmware please check with them. Or you could do the test it yourself as mentioned below.

You will need the following:

- PC under test with UEFI firmware
- PCI device with Option ROM on the PC under test (like a video card)
- Make sure Secure Boot is enabled

Steps for testing:

1. Insert a UEFI add on PCI card with UEFI Option ROM to the PC under test.

If you are using a PCI video card for testing, hookup an external monitor.

2. Enable Secure Boot with the settings below:

- PK: Your PK or self-signed Test PK
- KEK: MS KEK, PK-signed Fabrikam test KEK or another KEK
- DB: NULL. (This must be NULL.)
- DBX: NULL.
- SecureBoot: The UEFI variable should be set to true

3. Reboot the PC

4. Expect the following result:

- If the UEFI firmware is implemented correctly, the UEFI option ROM driver wouldn't load since the presence of an option ROM will make the firmware check the "Db" for a certificate. Since the "Db" is NULL the UEFI driver will fail to load. For example, if you are using the video card to test, you will see that nothing shows up on display.
- If the firmware isn't implemented correctly, UEFI driver will load from the option ROM since the firmware doesn't check for signatures in "Db". For example, if you are using the video card for test, you will see that the monitor hooked to the option ROM card will have display.

**Note** It doesn't matter if the UEFI option ROM driver is signed or not, the option ROM will not load when DB is null and SB is enabled (PK and KEK are enrolled).

Please refer to sample scripts available in the WHCK for generating the PK and KEK. You can download the scripts from here: <https://go.microsoft.com/fwlink/?LinkId=321292>. Appendix B has sample scripts and more details.

You can also reference Appendix A for another approach to performing the above test. This approach doesn't require setting the DB to Null but needs an unsigned UEFI option ROM driver from the IHV.

## 5. How to fix it

If the above test fails, work with your IBV to acquire the necessary versions and configure them to validate option ROMs. Make sure that the firmware passes the test. For PCs which have shipped you will need to do a secure firmware update. Please refer to [NIST publication 800-147](#) and/or see [Windows 8.1 Secure Boot Key Creation and Management Guidance](#).

You can test the PC and leverage Windows HCK as a test tool (not a certification tool) for testing the secure firmware update.

### 5.1. Signing the driver

In case you find that you may have unsigned drivers on UEFI option ROMs please read below on how to fix that.

Sign each option ROM driver individually. That will break the format of the PCI Option ROM. You only need to sign the UEFI driver before creating the combined Option ROM.

Before inserting the UEFI driver in the OpROM, sign the UEFI image and test it with Secure Boot ON & OFF at the UEFI Shell (load/unload the driver file). Then put the signed driver into the combined option ROM.

You can direct your IHV to Microsoft SysDev center to get their UEFI option ROMs signed through a service available through SysDev center.

### 5.2. Validation of update

Run the test you mentioned above to verify that the vulnerability does not exist. Use the HCK tests to ensure that there are no functional regressions.

## 6. Resources

- UEFI Platform Initialization Specification, Volume 5 Standards, 1.2.1 Errata A:  
<https://go.microsoft.com/fwlink/?linkid=220187>
- Relevant info from UEFI 2.3.1 spec:
  - 2.5.1: Legacy Option ROM Issues
  - 10: Protocols –UEFI Driver Model
  - 13.4.2: PCI Option ROMs
  - 20: EFI Byte Code Virtual Machine
  - 28: HII Overview
  - 29: HII Protocols
  - 30: HII Configuration Processing and Browser Protocol
- [UEFI Forum Learning Center](#)
- [UEFI IHV resources @ intel.com](#)
- Use the [TianoCore edk2-devel mailing list](#) for support from other UEFI developers
- TechNet: [Best Practices for Enterprise Security: Security strategies](#)
- [UEFI specification](#) errata C
- [Trusted Computing Group](#)

- [Tianocore UEFI Development Kit](#)
- [UEFI Firmware](#)
- [Intel Press: Beyond BIOS 2nd Edition](#)
- [Windows 8.1 Secure Boot Key Creation and Management Guidance](#)
- [Validating Windows UEFI Firmware Update Platform Functionality](#)

## Appendix A: Alternate approach to testing using unsigned option ROM drivers

This approach relies on getting tools from IHV to make sure that the UEFI option ROM driver is signed.

You will need the following:

- PC under test with UEFI firmware
- PCI device with an unsigned Option ROM driver attached to the PC under test (like a Video card)
- Make sure Secure Boot is enabled
- Option IHV tools to detect signature on option ROM driver if it isn't apparent that the option ROM driver is signed or not

If the firmware is implemented correctly, and the option ROM is unsigned the card should fail the check by firmware and not load the driver on the card. The PC should report an error code such as

**EFI\_IMAGE\_EXECUTION\_AUTH\_SIG\_FOUND**. In case you are using a video card, you may see that the PC shows just a black screen since the option ROM driver didn't load.

If the firmware is implemented incorrectly, this test would work.

## Appendix B: Scripts for enabling Secure Boot with NULL db

You can either use your current set of Secure Boot variables (PK and KEK) or generate test ones for testing this.

Below are steps used to generate the test PK, KEK and setting Db to NULL. Make sure that Secure Boot is not enabled; otherwise these steps would require signed UEFI bin files.

**Note** We set the Secure Boot variable – Db, KEK and PK in reverse order so we don't have to sign the UEFI bin files.

Prior to this step the PC should be in setup mode.

### 1. Create KEK and PK certificates

This step requires the makecert.exe tool available in the [Windows SDK](#).

```
MakeCert.exe -cy authority -len 2048 -m 60 -a sha256 -pe -ss my -n "CN=DO NOT SHIP - Fabrikam Test KEK CA" Fabrikam_Test_KEK_CA.cer
MakeCert.exe -cy authority -len 2048 -m 60 -a sha256 -pe -ss my -n "CN=DO NOT SHIP - Fabrikam Test PK" TestPK.cer
```

### 2. Script to generate test PK

You can either use your own PK or leverage the scripts from the WHCK for this

<https://go.microsoft.com/fwlink/?LinkId=321292>

A sample is provided below.

```

# this scripts demonstrates how to format the Platform key
# NOTE The PK is actually set in the Enable_OEM_SecureBoot.ps1 script
Import-Module secureboot
$d = (pwd).Path

#####
# Complete the following parameters
#####

$certname = "TestPK"
# TODO change this path to where you have the PK.cer file
# This is where you plugin the certificate generated by the HSM
$certpath = $d + "\\" + $certname + ".cer"

# Each signature has an owner SignatureOwner, which is a GUID identifying the agent which inserted the
signature in the database.
# Agents might include the operating PC or an OEM-supplied driver or application.
# Agents may examine this field to understand whether they should manage the signature or not.
# TODO replace with OEM SignatureOwner GUID.
# You can use tools like Guidgen.exe tool in SDK or a similar tool to generate a GUID
$sigowner = "55555555-5555-5555-5555-555555555555"

$var = "PK"
$efi_guid = "{8BE4DF61-93CA-11d2-AA0D-00E098032B8C}"
$append = $false

#####
# Everything else is calculated
#####

# Workaround relative path bug
# TODO substitute OEM with your OEM name
$siglist = $certname + "_SigList.bin"
$serialization = $certname + "_SigList_Serialization_for_" + $var + ".bin"
$signature = $serialization + ".p7"

$appendstring = "set_"
$attribute = "0x27"
$example = "Example_SetVariable_Data-" + $certname + "_" + $appendstring + $var + ".bin"

Format-SecureBootUEFI -Name $var -SignatureOwner $sigowner -ContentFilePath $siglist -FormatWithCert -
Certificate $certpath -SignableFilePath $serialization -Time 2011-05-21T13:30:00Z -AppendWrite:$append

# OutputFilePath - Specifies the name of the file created that contains the contents of what is set.
# If this parameter is specified, then the content are not actually set, just stored into this file.
# Please note if -OutputFilePath is provided the PK is not set like in this case. The master script
sets it at the end.

# Time - you can change the time below as long as it isn't in the future. Nothing wrong with keeping it
as is.

Set-SecureBootUEFI -Name $var -Time 2011-05-21T13:30:00Z -ContentFilePath $siglist -OutputFilePath
$example -AppendWrite:$append

```

### 3. Generate test KEK or use your own OEM KEK

You can leverage your own OEM KEK or scripts from the WHCK for this. You can also use the Fabrikam\_PK\_SigList.bin from <https://go.microsoft.com/fwlink/?LinkId=321292> instead of generating your own test KEK.

A sample is provided below.

```

# script to add option OEM KEK
Import-Module secureboot
$d = (pwd).Path

#####
# Complete the following parameters
#####

$certname = "Fabrikam_Test_KEK_CA"
# TODO change this path to where you have the PK.cer file
# This is where you plugin the certificate generated by the HSM
$certpath = $d + "\\" + $certname + ".cer"

# TODO change this path to where you have the OEM_KEK.cer file
# Each signature has an owner SignatureOwner, which is a GUID identifying the agent which inserted the
signature in the database.
# Agents might include the operating system or an OEM-supplied driver or application.
# Agents may examine this field to understand whether they should manage the signature or not.
# TODO replace with OEM SignatureOwner GUID.
# You can use tools like Guidgen.exe tool in SDK or a similar tool to generate a GUID

$sigowner = "00000000-0000-0000-0000-000000000000"

$var = "KEK"
$efi_guid = "{8BE4DF61-93CA-11d2-AA0D-00E098032B8C}"
$append = $false

#####
# Everything else is calculated
#####

$siglist = $certname + "_SigList.bin"
$serialization = $certname + "_SigList_Serialization_for_" + $var + ".bin"
$signature = $serialization + ".p7"
if ($append -eq $false)
{
    $appendstring = "set_"
    $attribute = "0x27"
}
else
{
    $appendstring = "append_"
    $attribute = "0x67"
}
$example = "Example_SetVariable_Data-" + $certname + "_" + $appendstring + $var + ".bin"

Format-SecureBootUEFI -Name $var -SignatureOwner $sigowner -ContentFilePath $siglist -FormatWithCert -
CertificateFilePath $certpath -SignableFilePath $serialization -Time 2011-05-21T13:30:00Z -
AppendWrite:$append

# -Time You can change the time below as long as it isn't in the future. Nothing wrong with keeping it
as is.

Set-SecureBootUEFI -Name $var -Time 2011-05-21T13:30:00Z -ContentFilePath $siglist -OutputFilePath
$example -AppendWrite:$append

```

#### 4. Set Db to Null and set KEK and PK

The first thing this script does is set the Db to Null.

**Note** Please keep in mind if the Fabrikam Test KEK CA is the only KEK CA present (meaning there is no Windows KEK CA), the PC may boot into Windows RE.

```
# Prior to script execution, run "Set-ExecutionPolicy Bypass -Force"

Import-Module secureboot
try
{
    Write-Host "Deleting db..."
    Set-SecureBootUEFI -Name db -Time "2011-06-06T13:30:00Z" -Content $null
}
catch
{
}
Write-Host "Setting Fabrikam KEK..."
Set-SecureBootUEFI -Time 2011-05-21T13:30:00Z -ContentFilePath Fabrikam_Test_KEK_CA_SigList.bin -Name KEK

Write-Host "Setting self-signed Test PK..."
Set-SecureBootUEFI -Time 2011-05-21T13:30:00Z -ContentFilePath TestPK_SigList.bin -Name PK

Write-Host "`n... operation complete. `nSetupMode should now be 0 and SecureBoot should also be 0.
Reboot and verify that Windows is correctly authenticated, and that SecureBoot changes to 1."
```

## 5. Plug in the option ROM card and test

The test should either pass or fail based on firmware correctness. For example:

If the option ROM in the firmware is implemented correctly, and you are using a video card for testing, then there should be no display to the attached monitor.

However, if you are using incorrect firmware, the video card should have output on the display.

## Related topics

[Windows Secure Boot Key Creation and Management Guidance](#)

[Secure Boot Overview](#)

[Validating Windows UEFI Firmware Update Platform Functionality](#)

# Disabling Secure Boot

1/18/2019 • 3 minutes to read

You may need to disable Secure Boot to run some PC graphics cards, hardware, or operating systems such as Linux or previous version of Windows.

Secure Boot helps to make sure that your PC boots using only firmware that is trusted by the manufacturer. You can disable Secure Boot through the PC's firmware (BIOS) menus, but the way you disable it varies by PC manufacturer. If you are having trouble disabling Secure Boot after following the steps below, contact your manufacturer for help.

For logo-certified Windows RT 8.1 and Windows RT PCs, Secure Boot is required to be configured so that it cannot be disabled.

## Warning

- After disabling Secure Boot and installing other software and hardware, it may be difficult to re-activate Secure Boot without restoring your PC to the factory state.
- Be careful when changing BIOS settings. The BIOS menu is designed for advanced users, and it's possible to change a setting that could prevent your PC from starting correctly. Be sure to follow the manufacturer's instructions exactly.

## Disable Secure Boot

1. Before disabling Secure Boot, consider whether it is necessary. From time to time, your manufacturer may update the list of trusted hardware, drivers, and operating systems for your PC. To check for updates, go to Windows Update, or check your manufacturer's website.
2. Open the PC BIOS menu. You can often access this menu by pressing a key during the bootup sequence, such as F1, F2, F12, or Esc.

Or, from Windows, hold the Shift key while selecting **Restart**. Go to **Troubleshoot > Advanced Options: UEFI Firmware Settings**.

3. Find the **Secure Boot** setting, and if possible, set it to **Disabled**. This option is usually in either the **Security** tab, the **Boot** tab, or the **Authentication** tab.
4. Save changes and exit. The PC reboots.
5. Install the graphics card, hardware, or operating system that's not compatible with Secure Boot.

In some cases, you may need to change other settings in the firmware, such as enabling a Compatibility Support Module (CSM) to support legacy BIOS operating systems. To use a CSM, you may also need to reformat the hard drive using the Master Boot Record (MBR) format, and then reinstall Windows. For more info, see [Windows Setup: Installing using the MBR or GPT partition style](#).

6. If you're using Windows 8.1, you may see a watermark on the desktop alerting you that Secure Boot is not configured correctly. Get this [update to remove the Secure Boot desktop watermark](#).

## Re-enable Secure Boot

1. Uninstall any graphics cards, hardware, or operating systems that aren't compatible with Secure Boot.
2. Open the PC BIOS menu. You can often access this menu by pressing a key during the bootup sequence,

such as F1, F2, F12, or Esc.

Or, from Windows: go to **Settings charm > Change PC settings > Update and Recovery > Recovery > Advanced Startup: Restart now**. When the PC reboots, go to **Troubleshoot > Advanced Options: UEFI Firmware Settings**.

3. Find the **Secure Boot** setting, and if possible, set it to **Enabled**. This option is usually in either the **Security** tab, the **Boot** tab, or the **Authentication** tab.

On some PCs, select **Custom**, and then load the Secure Boot keys that are built into the PC.

If the PC does not allow you to enable Secure Boot, try resetting the BIOS back to the factory settings.

4. Save changes and exit. The PC reboots.
5. If the PC is not able to boot after enabling Secure Boot, go back into the BIOS menus, disable Secure Boot, and try to boot the PC again.
6. In some cases, you may need to refresh or Remove everything to its original state before you can turn on Secure Boot. For more info, see [How to restore, refresh, or Remove everything](#).
7. If the above steps don't work, and you still want to use the Secure Boot feature, contact your manufacturer for help.

For additional troubleshooting steps for PC manufacturers: see [Secure Boot isn't configured correctly: Determine if the PC is in a manufacturing mode \(info for manufacturers\)](#).

## Related topics

[Secure Boot Overview](#)

[Secure Boot isn't configured correctly: troubleshooting](#)

# Secure Boot isn't configured correctly: troubleshooting

1/24/2019 • 2 minutes to read

The "Secure Boot isn't configured correctly" watermark appears on the Windows desktop when the PC is capable of using the Secure Boot security feature, but the feature is not activated or configured correctly.

This message may appear after updating your PC from Windows 8 to Windows 8.1.

## What is Secure Boot?

Secure Boot helps to make sure that your PC boots using only firmware that is trusted by the manufacturer. For more info on Secure Boot, see [Secure Boot Overview](#).

## Is my PC unsafe?

Your PC may be OK, but it's not as protected as it could be, because Secure Boot isn't running.

You may need to disable Secure Boot to run some hardware, graphics cards, or operating systems such as Linux or previous versions of Windows. For more info, see [Disabling Secure Boot](#).

You check the status of Secure Boot on your PC. click on [Start](#), and type msinfo32 and press enter. Under System Summary, you can see your BIOS mode and Secure Boot State. If Bios Mode is UEFI, and Secure Boot State is Off, then Secure Boot is disabled.

## Can I just dismiss this alert or remove the watermark?

Yes. Go to Windows Update for a patch that gets rid of the watermark.

- Windows 8.1 and Windows Server 2012 R2 users can also download this patch manually: [Update removes the "Windows 8.1 SecureBoot isn't configured correctly" watermark in Windows 8.1 and Windows Server 2012 R2 \(Microsoft Knowledge Base Article ID 2902864\)](#)
- Windows RT 8.1: Get this patch through Windows update.

## I'd like to use this feature. How can I enable it?

Try enabling Secure Boot through using the PC BIOS menus.

**Warning** Be careful when changing BIOS settings. The BIOS menu is designed for advanced users, and it's possible to change a setting that could prevent your PC from starting correctly. Be sure to follow the manufacturer's instructions exactly.

### Enabling Secure Boot

1. Open the PC BIOS menu. You can often access this menu by pressing a key during the bootup sequence, such as F1, F2, F12, or Esc.

Or, from Windows: go to **Settings charm > Change PC settings > Update and Recovery > Recovery > Advanced Startup: Restart now**. When the PC reboots, go to **Troubleshoot > Advanced Options: UEFI Firmware Settings**.

2. Find the **Secure Boot** setting, and if possible, set it to **Enabled**. This option is usually in either the **Security**

tab, the **Boot** tab, or the **Authentication** tab.

On some PCs, select **Custom**, and then load the Secure Boot keys that are built into the PC.

If the PC does not allow you to enable Secure Boot, try resetting the BIOS back to the factory settings.

3. Save changes and exit. The PC reboots.
4. If the PC is not able to boot after enabling Secure Boot, go back into the BIOS menus, disable Secure Boot, and try to boot the PC again.
5. In some cases, you may need to refresh or Remove everything to its original state before you can turn on Secure Boot. For more info, see [How to restore, refresh, or Remove everything](#).
6. If the above steps don't work, and you still want to use the Secure Boot feature, contact your manufacturer for help.

For additional troubleshooting steps for PC manufacturers: see [Secure Boot isn't configured correctly: Determine if the PC is in a manufacturing mode \(info for manufacturers\)](#).

## Related topics

[Why is there a "SecureBoot isn't configured correctly" watermark on my desktop?](#)

[Secure Boot Overview](#)

[Microsoft Support KB article 2902864](#)

[Disabling Secure Boot](#)

# BCD System Store Settings for UEFI

1/18/2019 • 6 minutes to read

For a typical deployment scenario, you do not need to modify the BCD store. This topic discusses the various BCD settings in the BCD store that you can modify. On UEFI systems, this includes settings for the following boot applications:

1. [Windows Boot Manager](#)
2. [Windows Boot Loader](#)
3. [Windows Memory Tester](#)

The following sections describe the available settings for each of these boot applications in detail and how to modify each application for UEFI systems.

For simplicity, the BCDEdit examples in this section modify the BCD system store. To modify another store, such as a copy of the BCD-template, include the store name in the command line.

## Windows Boot Manager Settings for UEFI

Windows Boot Manager (`{bootmgr}`) manages the boot process. UEFI-based systems contain a firmware boot manager, `Bootmgfw.efi`, that loads an EFI application that is based on variables that are stored in NVRAM.

The BCD settings for the `device` and `path` elements in Windows Boot Manager indicate the firmware boot manager. The template that is named BCD-template for Windows includes the following settings for Windows Boot Manager.

```
## Windows Boot Manager

identifier      {bootmgr}
device          partition=\Device\HarddiskVolume1
path            \EFI\Microsoft\Boot\bootmgfw.efi
description    Windows Boot Manager
```

### Device Setting

The `device` element specifies the volume that contains Windows Boot Manager. For UEFI systems, the `device` element for Windows Boot Manager is set to the system partition volume letter. To determine the correct volume letter, use the Diskpart tool to view the disk partitions. The following example assumes that the system has a single hard drive that has multiple partitions, including a system partition that has been assigned a drive letter of S.

The following Diskpart commands select disk 0 and then list the details of the volumes on that disk, including their drive letters. It shows volume 2 as the system partition.

```
DISKPART> select disk 0
DISKPART> list volume

Volume ### Ltr Label Fs Type Size Status Info
----- -- -
Volume 0 D NTFS Partition 103 GB Healthy
Volume 1 C NTFS Partition 49 GB Healthy Boot
Volume 2 S FAT32 Partition 200 MB Healthy System
```

If the system partition does not have an assigned drive letter, assign one by using the **Diskpart assign** command. The following example assumes that the system partition is volume 2 and assigns it S as the drive letter.

```
Diskpart
select disk 0
list volume
select volume 2 // assuming volume 2 is the system partition
assign letter=s
```

After you have determined the system partition volume, set the `device` element for Windows Boot Manager to the corresponding drive letter. The following example sets `device` to drive S.

```
Bcdedit /set {bootmgr} device partition=s:// system partition
```

## Path Setting

The `path` element specifies the location of the Windows Boot Manager application on that volume. For UEFI systems, `path` indicates the firmware boot manager, whose path is \EFI\Microsoft\Boot\Bootmgfw.efi.

You can confirm that BCD-template has the correct path by enumerating the values in the store, as follows:

```
bcdedit /store bcd-template /enum all
```

To explicitly set `path` to \EFI\Microsoft\Boot\Bootmgfw.efi, use the following command.

```
Bcdedit /set {bootmgr} path \efi\microsoft\boot\bootmgfw.efi
```

## Other Settings

You should set Windows Boot Manager to be the first item in the display order of the UEFI firmware, as shown in the following example.

```
Bcdedit /set {fwbootmgr} displayorder {bootmgr} /addfirst
```

You should also specify the topmost Windows boot loader application in the Windows Boot Manager display order. The following example shows how to put a specified Windows boot loader at the top of the display order.

```
Bcdedit /set {bootmgr} displayorder {<GUID>} /addfirst
```

In the preceding example, <GUID> is the identifier for the specified Windows boot loader object. The next section discusses this identifier in greater detail.

**Note** A multiboot system that has multiple installed operating systems has multiple instances of the Windows boot loader. Each instance of the Windows boot loader has its own identifier. You can set the default Windows boot loader (`{default}`) to any of these identifiers.

## Windows Boot Loader Settings

A BCD store has at least one instance, and optionally multiple instances, of the Windows boot loader. A separate BCD object represents each instance. Each instance loads one of the installed versions of Windows that has a configuration that the object's elements have specified. Each Windows boot loader object has its own identifier, and the object's `device` and `path` settings indicate the correct partition and boot application.

`BCD-template` for Windows has a single Windows boot loader object that has the following settings.

```
## Windows Boot Loader

identifier          {9f25ee7a-e7b7-11db-94b5-f7e662935912}
device              partition=C:
path                \Windows\system32\winload.efi
description        Microsoft Windows Server
locale              en-US
inherit             {bootloadersettings}
osdevice            partition=C:
systemroot          \Windows
```

The identifier for this Windows boot loader is `{9f25ee7a-e7b7-11db-94b5-f7e662935912}`. You can use this GUID on your system or let the BCDEdit tool generate a new GUID for you.

To simplify BCDEdit commands, you can specify one of the Windows boot loaders in the BCD system store as the default loader. You can then use the standard identifier (`{default}`) in place of the full GUID. The following example specifies the Windows boot loader for EFI as the default boot loader, assuming that it uses the identifier GUID from `BCD-template`.

```
Bcdedit /default {9f25ee7a-e7b7-11db-94b5-f7e662935912}
```

## Device and OSDevice Settings

The following elements specify key locations:

The `device` element specifies the partition that contains the boot application.

The `osdevice` element specifies the partition that contains the system root.

For the Windows boot loader for EFI, both elements are usually set to the drive letter of the Windows system partition. However, if BitLocker is enabled or a computer has multiple installed versions of Windows, `osdevice` and `device` might be set to different partitions. `BCD-template` sets both elements to drive C, which is the typical value. You can also explicitly set the `osdevice` and `device` values, as shown in the following example. The example also assumes that you have specified the Windows boot loader for EFI as the default boot-loader object.

```
Bcdedit /set {default} device partition=c:
Bcdedit /set {default} osdevice partition=c:
```

## Path Setting

The `path` element of a Windows boot loader specifies the location of the boot loader on that volume. For UEFI systems, `path` indicates the Windows boot loader for EFI, whose path is `\Windows\System32\Winload.efi`.

You can confirm that `BCD-template` has the correct `path` value by enumerating the values in the store. You can also explicitly set the `path` value, as shown in the following example.

```
Bcdedit /set {default} path \windows\system32\winload.efi
```

## Windows Memory Tester Settings

The Windows memory tester (`{memdiag}`) runs memory diagnostics at boot time. The BCD settings for the application's `device` and `path` elements indicate the correct application.

**Note** Note: Intel Itanium computers do not include a Windows memory tester and do not require `{memdiag}`

settings.

BCD-template for Windows has the following settings.

```
## Windows Memory Tester

identifier          {memdiag}
device              partition=\Device\HarddiskVolume1
path                \boot\memtest.exe
description        Windows Memory Diagnostic
```

## Device Setting

For UEFI systems, the `device` element for the Windows memory tester is set to the system partition drive letter. The following example assumes that the system partition is drive S, as used in earlier examples.

```
Bcdedit /set {bootmgr} device partition=s: // system partition
```

## Path Setting

The `path` element specifies the location of Windows Test Manager on the volume that the `device` element has specified. For UEFI systems, `path` indicates the EFI version of the application (\EFI\Microsoft\Boot\Memtest.efi).

You can confirm that BCD-template has the correct `path` value by enumerating the values in the store. You can also use the BCDEdit tool to explicitly set the `path` value, as shown in the following example.

```
Bcdedit /set {memdiag} path \efi\microsoft\boot\memtest.efi
```

# Validating Windows UEFI Firmware Update Platform Functionality

1/18/2019 • 9 minutes to read

This document lists the basic validation scenarios that are required to pass before signing-off on the Windows UEFI Firmware Update Platform functionality. Specification can be downloaded from [here](#).

## Prerequisites

- For each EFI System Resource Table (ESRT) entry, you need a capsule for the latest firmware version. The scenarios will refer to the latest version as X. Each ESRT entry is identified using a unique GUID.
- For each ESRT entry exposed, create a capsule package that its version is incremented above the package created in step 1. These capsules will be referred to as X+1.
- Capsules that aid in simulating failure conditions such as a capsule for which the payload is not signed or signed with an invalid PK.
- Make sure all capsules to be used are signed appropriately from the OS perspective, catalog signed, and firmware signed, PK signed. Unless, you are specifically testing the negative PK signing cases. See "Signing the Firmware driver Package" in the specification for details on how to sign a capsule or firmware driver package.

## How To

### Install a new capsule or reinstall a previously installed capsule

1. Open up device manager.
2. Find the device node that represents your firmware, it is usually under the "Firmware" devices.
3. Right click on the firmware device you wish to update.
4. Select **Update driver software**. You will get a popup that states "Update Driver Software - <Firmware>".
5. Select **Browse my computer for driver software**.
6. On the next window, select **Let me pick from a list of device drivers on my computer**.
7. If the driver has been installed before, select it from the **Show compatible hardware** box. If it does not exist, select **Have disk** and continue on. Otherwise, select **OK** and reboot the system.
8. If you select **Have Disk**, you will get a popup labeled **Install From Disk**.
9. Use **Browse** to go to the directory that has the capsule of the firmware you wish to install.
10. Select the INF file in that directory and hit **OK** to install.
11. During installation, if you get a popup saying the driver is not signed, go ahead and accept this driver.
12. The system asks you to reboot.
13. After you installed the capsule for the firmware, you need to reboot. If you wish to install multiple capsule packages, then wait to reboot until all capsules are installed and then reboot on the final capsule.

### Query the version and status details:

- Run the QueryVersionAndStatus.ps1 PowerShell (PS) script to query the current firmware version, last attempt firmware version and last attempt status.

### To run the script:

1. Run PowerShell as administrator.
2. `Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Force` (This only has to be done once.)

3. Display the version and status details for the given GUID. For example:  

```
.\QueryVersionAndStatus.ps1 6bd2efb9-23ab-4b4c-bc37-016517413e9a
```
4. Check if firmware update was successful: Refer to the section "Validating the status of the firmware update" in the specification document. Make sure that the Last Attempt Status and the Current Version matches the expected version.
5. Recommended: Check to make sure that the devices you are updating are also still functioning.
6. Set the rollback policy: Some of the scenarios might require rolling back firmware. Rollback is not a production scenario. In order to be able to rollback, a registry policy key has to be created. Create a REG\_DWORD key with the name "Policy" under the node HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\FirmwareResources\{<GUID>} and set the value of the Policy key to 1. Note that the GUID should be replaced with the actual GUID from the ESRT.

## Scenarios

### **S1: Each ESRT entry is successfully updatable through capsule**

The following steps should be completed for each ESRT entry that is supported by the platform. Or in other words, for System firmware and each device firmware that supports updating firmware through UpdateCapsule.

#### **Steps**

1. For each ESRT entry, install the capsule for firmware version X.
2. Make sure all the above capsules are installed, prior to rebooting.

#### **Expected Result**

Firmware update should be successful for each ESRT entry that was updated. For all ESRT entries, for which the update was attempted, validate that:

- Current Firmware Version = X
- Last Attempt Version = X
- Last Attempt Status = 0 (STATUS\_SUCCESS)

### **S2: The latest firmware version X is also updatable to X+1**

The following steps should be completed for each ESRT entry that is supported by the platform. Or in other words, for System firmware and each device firmware that supports updating firmware through UpdateCapsule.

#### **Steps**

1. Complete scenario S1 above.
2. For each ESRT entry, install the capsule for firmware version X+1.

#### **Expected Result**

Firmware update should be successful for each ESRT entry that was updated. For all ESRT entries, for which the update was attempted, validate that:

- Current Firmware Version = X+1
- Last Attempt Version = X+1
- Last Attempt Status = 0 (STATUS\_SUCCESS)

### **S3: On failure, firmware update returns the right status code as defined in the specification**

The Status codes are defined in the section named "UEFI System Resource Table Definition", in the table with the title "ESRT Last Attempt Status Field Values".

#### **S3.1 Insufficient Battery and UEFI System Firmware update**

## **Steps**

1. Drain the battery charge to less than 25% and then plug-in the AC power.
2. Install the capsule for UEFI System Firmware version X+1. Let's assume that the current version is X.
3. Before rebooting, make sure that the battery charge is less than 25%

## **Expected Result**

Firmware update should fail. For ESRT entry corresponding to the System Firmware, validate that:

- Current Firmware Version = X
- Last Attempt Version = X+1
- Last Attempt Status = 0xc00002de (STATUS\_INSUFFICIENT\_POWER)

## **S3.2 Insufficient Battery and Device Firmware update**

### **Steps**

1. Drain the battery charge to less than 25% and then plug-in the AC power.
2. Install the capsules for ALL supported devices in the system with firmware version X+1. Let's assume that the current firmware version for the given device is X.
3. Before rebooting, make sure that the battery charge is less than 25% .

## **Expected Result**

Firmware update should fail. For all ESRT entries, for which the update was attempted, validate that:

- Current Firmware Version = X
- Last Attempt Version = X+1
- Last Attempt Status = 0xc00002de (STATUS\_INSUFFICIENT\_POWER)

## **S3.3 Insufficient Battery, UEFI System and Device Firmware update at the same time**

### **Steps**

1. Drain the battery charge to less than 25% and then plug-in the AC power.
2. Install the capsules for UEFI System Firmware and all Device Firmware with version X+1.
3. Before rebooting, make sure that the battery charge is less than 25%.

## **Expected Result**

Firmware update should fail for the System firmware and for all the device firmware for which the update was attempted. For all ESRT entries, for which the update was attempted, validate that:

- Current Firmware Version = X
- Last Attempt Version = X+1
- Last Attempt Status = 0xc00002de (STATUS\_INSUFFICIENT\_POWER)

## **S3.4 Firmware update should fail when the capsule is not PK signed**

The following steps should be completed for each ESRT entry that is supported by the platform. Or in other words, for System firmware and each device firmware that supports updating firmware through UpdateCapsule.

### **Steps**

1. For each ESRT entry, create a capsule X+2, the payload for which is not signed.
2. Install the capsules X+2. Let's assume that the current version is X.

## **Expected Result**

Firmware update should fail for all the ESRT entries for which the update was attempted. For all ESRT entries, for

which the update was attempted, validate that:

- Current Firmware Version = X
- Last Attempt Version = X+2
- Last Attempt Status = 0xC0000022 (STATUS\_ACCESS\_DENIED)

### S3.5 Firmware update should fail when the capsule is signed with the wrong PK certificate

The following steps should be completed for each ESRT entry that is supported by the platform. Or in other words, for System firmware and each device firmware that supports updating firmware through UpdateCapsule.

#### Steps

1. For each ESRT entry, create a capsule X+2, sign the payload with a wrong key or certificate (for example use a debug signed capsule on a production device).
2. Install the capsules X+2. Let's assume that the current version is X.

#### Expected Result

Firmware update should fail for all the ESRT entries for which the update was attempted. For all ESRT entries, for which the update was attempted, validate that:

- Current Firmware Version = X
- Last Attempt Version = X+2
- Last Attempt Status = 0xC0000022 (STATUS\_ACCESS\_DENIED)

### S3.6 Firmware update should fail when the capsule payload is tampered with

The following steps should be completed for each ESRT entry that is supported by the platform. Or in other words, for System firmware and each device firmware that supports updating firmware through UpdateCapsule.

#### Steps

1. For each ESRT entry, create a capsule X+2, sign the payload with the right key or certificate. Then open the firmware bin file and flip 1 or more bits in the file and save the file back.
2. Regenerate the catalog for the bin file and the INF file.
3. Install the capsules X+2. Let's assume that the current version is X.

#### Expected Result

Firmware update should fail for all the ESRT entries for which the update was attempted. For all ESRT entries, for which the update was attempted, validate that:

- Current Firmware Version = X
- Last Attempt Version = X+2
- Last Attempt Status = 0xC0000022 (STATUS\_ACCESS\_DENIED) or 0xC000007B (STATUS\_INVALID\_IMAGE\_FORMAT)

### S3.7: Firmware does not allow rollback beyond the LowestSupportedFirmwareVersion

The following steps should also be carried out for other device firmware (lower priority).

#### Steps

1. For UEFI System Firmware, create a capsule X+1 such that the "LowestSupportedFirmwareVersion" in the ESRT entry for the system firmware is set to X+1.
2. Install the capsule X+1 and make sure that the update succeeds.
3. Create a UEFI System firmware update capsules, such that the version in the INF is X+2 but the actual firmware binary file is of version X.
4. Install the capsule X+2 and reboot the system.

## **Expected Result**

Firmware update should fail. For ESRT entry corresponding to the System Firmware, validate that:

- Current Firmware Version = X+1
- Last Attempt Version = X+2
- Last Attempt Status = 0xC0000059 (STATUS\_REVISION\_MISMATCH)

## **S4: Seamless recovery and firmware update (if implemented)**

This scenario varies from platform to platform depending on the implementation of the seamless recovery. Based on the implementation, the validation might require creating bad capsules that forces the system into recovery or disconnecting the power in the middle of an update or through any other means of exercising the recovery flows.

## **Expected Result**

The system should boot into the OS and the firmware update should be marked as failed. The version reported by the UEFI firmware resource device should not have changed.

## **S5: Firmware Update adheres to the User Experience (UX) requirement**

### **Steps**

- This scenario can be validated while executing any of the above scenarios that lead to a successful firmware update.

## **Expected Result**

The user experience is in accordance to the specification, see section on "User Experience".

- The only text that is displayed on the screen is "Please wait while we install a system update". The text is displayed at the right co-ordinates on the screen as called out in the specification.
- OEM Logo is displayed as described in the specification.

## **Related topics**

[Windows UEFI Firmware Update Platform](#)

[UEFI Validation Option ROM Validation Guidance](#)

# Localize

1/18/2019 • 2 minutes to read

Set up Windows 10 or Windows Server with the right set of languages, settings, and keyboards or other input devices for customers in different regions.

## Localizing Windows

You can add languages and regional support to Windows 10 (except for Windows 10 Home Single Language and Windows 10 Home Country Specific editions), and Windows Server.

### Add languages to Windows images

Windows installations start with at least one language pack and its language components. You can add:

- [Language packs](#): Fully-localized Windows UI text for the dialog boxes, menu items, and help files that you see in Windows. Delivered as .cab files, for example, Microsoft-Windows-Client-Language-Pack\_x64\_es-es.cab.
- [Language Interface Packs \(LIP\)](#): Partially-localized languages. LIPs require a base language pack. For UI that's not localized in the LIP, Windows shows UI from the base language pack. Delivered as .appx files, for example, LanguageExperiencePack.am-et.neutral.appx.
- [Features On Demand](#): Features include language basics (like spell checking), fonts, optical character recognition, handwriting, text-to-speech, and speech recognition. You can save disk space by choosing not to include some language components in your image. While this reduction in image size can be helpful when creating images for lower-cost devices with small storage, it does lead to an incomplete language experience. Delivered as .cab files, for example, Microsoft-Windows-LanguageFeatures-Basic-fr-fr-Package.
- [Recovery languages](#): UI text for the Windows Recovery Environment (WinRE). Delivered as .cab files. Example: lp.cab, WinPE-Rejuv\_fr-fr.cab, and more.
- [International settings](#): Specify the default language, locale, and keyboard values during deployment or after Windows is installed.
  - List of [default input profiles \(input locales\)](#). Input profiles combine the language of the input entered and the keyboard identifier.
  - List of [keyboard identifiers and input method editors \(IME\)](#)
  - List of [default time zones](#)
- [Out of Box Experience \(OOBE\)](#): Create localized versions of end user license agreements (EULA), registration pages, and instructions for pairing keyboards and mice.
- [Retail demo experience](#): Showcase your new devices on the retail sales floor with localized content.

## Localizing the installation process

- [Add languages to Windows Setup](#) to create Windows installation media that works in the user's preferred language and shows the languages available in the Windows image.
- [Add languages to WinPE](#) to help your technicians install Windows in their preferred language.
- [Add languages to a Windows distribution share](#) to help users on a corporate network select Windows in their preferred language

## Related topics

- [Features On Demand](#)

- Add languages to [Windows Recovery Environment \(WinRE\)](#)

# Add languages to Windows images

1/25/2019 • 9 minutes to read

## NOTE

To add a language to your personal PC, go to **Settings > Time & Language > Language**, and choose a language to install. [Learn more](#)

You can add languages and regional support to Windows 10 (except for Windows 10 Home Single Language and Windows 10 Home Country Specific editions), and Windows Server.

Windows installations start with at least one language pack and its language components. You can add:

- **Language packs:** Localization packages for Windows, delivered as a .cab file, for example, Microsoft-Windows-Client-Language-Pack\_x64\_es-es.cab. Includes UI elements like text for dialog boxes, menu items, and help files.
- **Language Interface Packs (LIPs):** Partially-localized language pack. Requires a base language pack. In Windows 10, version 1809, LIPs are delivered as Local Experience Packs (LXPs) .appx files, for example, LanguageExperiencePack.am-et.neutral.appx. For previous versions of Windows 10, LIPs are delivered as .cab files, for example, C:\Languages\es-ES\lp.cab.
- **Features On Demand:** Features include language basics (like spell checking), fonts, optical character recognition, handwriting, text-to-speech, and speech recognition. You can save disk space by choosing not to include some language components in your image. While this reduction in image size can be helpful when creating images for lower-cost devices with small storage, it does lead to an incomplete language experience. Delivered as .cab files, for example, Microsoft-Windows-LanguageFeatures-Basic-fr-fr-Package.
- **Recovery languages:** UI text for the Windows Recovery Environment (WinRE). Delivered as .cab files. Example: lp.cab, WinPE-Rejuv\_fr-fr.cab, and more.

## Get language resources: Language Pack ISO and Feature on Demand ISO

- **OEMs and System Builders** with Microsoft Software License Terms can download the Language Pack ISO and Feature on Demand ISO from the [Microsoft OEM site](#) or the [Device Partner Center](#).
  - For Windows 10, version 1809, LIP .appx files and their associated license files are in the LocalExperiencePack folder on the Language Pack ISO.
  - For previous versions of Windows, Language Interface Packs are available as a separate download.
  - WinRE language packs are distributed on the Language Pack ISO. Don't use the WinPE language packs that ship with the ADK.
- **IT Professionals** can download language packs from the [Microsoft Next Generation Volume Licensing Site](#).
- After Windows is installed, users can download and install more languages by selecting **Settings > Time & language > Language > Add a language**.

Notes:

- Language components must match the version of Windows. For example, you can't add a Windows 10, version 1809 language pack to Windows 10, version 1803.

- Windows Server: Full language packs are not interchangeable with Windows 10, but some LIPs are. For example, you can add Windows 10, version 1809 LIPs to Windows Server 2019.

## Considerations

- **Install languages, then updates and apps.** If you're adding languages to an image that already contains apps or updates (for example, servicing stack updates (SSU) or cumulative updates (CU)), reinstall the apps and updates.
- **Size and performance**
  - You can install multiple languages and components onto the same Windows image. Having too many affects disk space, and can affect performance, especially while updating and servicing Windows.
  - Some languages require more hard-disk space than others.
  - When creating Windows images, you can remove English language components when deploying to non-English regions to save space. You'll need to uninstall them in the reverse order from how you add them.
  - After the Out-Of-Box-Experience (OOBE) is complete, the default language cannot be removed. The default UI language is selected either during the Out-Of-Box-Experience (OOBE), or can be preselected in the image using DISM or an unattended answer file.
  - Some time after OOBE, any preinstalled languages that haven't been used are removed automatically. The language-pack removal task does not remove LIPs.
- **Cross-language upgrades are not supported.** This means that during upgrades or migrations, if you upgrade or migrate an operating system that has multiple language packs installed, you can upgrade or migrate to the system default UI language only. For example, if English is the default language, you can upgrade or migrate only to English.

## Add and remove language packs, LIPs, and features on demand

We'll install the French language, and then add a LIP language (Luxembourgish) that uses French for its base language.

To save space, you can remove English language components when deploying to non-English regions by uninstalling the language components in the reverse order from how you add them.

### Prepare to modify your images: Installation methods

- **Offline: Modify an image (install.wim) that you plan to deploy using Windows PE or Windows Setup:**

Mount the Windows image:

```
md C:\mount\windows
Dism /Mount-Image /ImageFile:install.wim /Index:1 /MountDir:"C:\mount\windows"
```

- **Online: Modify a running Windows installation** (for example, in [audit mode](#))

You can install languages and LIPs by opening the packages from the Language Pack ISO and Feature on Demand ISO and following the instructions to complete the installation.

You can use the **Settings > Time & Language** to modify the settings or download languages from Windows Update.

To run these same updates using a script, use the instructions in this topic, replacing `/Image:"C:\mount\windows"` with `/Online`.

To update the recovery environment, see [Add languages to the recovery environment](#).

## Add and remove language packs, LIPs (LXPs), and Features on Demand

Preinstall languages by adding the language packs and their related Features on Demand for all preinstalled languages, including the base languages if you're adding a LIP language.

1. Mount the Language Pack ISO and the Features on Demand ISO with File Explorer. This will assign them drive letters.

### 2. Add a language pack:

```
Dism /Image:"C:\mount\windows" /Add-Package /PackagePath="D:\x64\langpacks\Microsoft-Windows-Client-Language-Pack_x64_fr-fr.cab"
```

Where D:\ is the Language Pack ISO.

Verify that it's in the image:

```
Dism /Image:"C:\mount\windows" /Get-Packages
```

3. **Add LIP languages.** In this example, we add Luxembourgish, which requires the fr-FR base language:

- For Windows 10, version 1809 and later, LIP languages are delivered as LXPs:

```
DISM /Image:"C:\mount\windows" /Add-ProvisionedAppxPackage  
/PackagePath="D:\LocalExperiencePack\lb-lu\LanguageExperiencePack.lb-LU.Neutral.appx  
/LicensePath:"D:\LocalExperiencePack\lb-lu\License.xml"
```

Where D:\ is the Language Pack ISO

Verify that it's in the image:

```
DISM /Image:"C:\mount\windows" get-provisionedappxpackages
```

- For earlier versions of Windows 10, LIP languages are delivered as .cab files:

```
Dism /Image:C:\mount\windows /add-package /packagepath:C:\LanguagePack\LIPs\ca-ES\LIP_ca-  
ES64bit.cab
```

4. **Add language and region Features on Demand** to support your language packs and LIPs. We recommend including the Basic, Fonts, OCR, Text-to-speech, and Speech recognition FODs if they're available for the languages you're preinstalling. Additionally, include the handwriting language component FOD if you're shipping a device with a pen.

Not all LIP languages have all language components. Luxembourgish (lb-LU), for example, only has basic and handwriting FODs. You can learn which FODs are available for languages [in the LP to FOD mapping spreadsheet](#)

Example: adding Basic, OCR, Text-to-Speech, and Speech Recognition to fr-fr:

```
Dism /Image:"C:\mount\windows" /Add-Package /PackagePath:E:\Microsoft-Windows-LanguageFeatures-Basic-fr-fr-Package~31bf3856ad364e35~amd64~~.cab /PackagePath:E:\Microsoft-Windows-LanguageFeatures-OCR-fr-fr-Package~31bf3856ad364e35~amd64~~.cab /PackagePath:E:\Microsoft-Windows-LanguageFeatures-Handwriting-fr-fr-Package~31bf3856ad364e35~amd64~~.cab /PackagePath:E:\Microsoft-Windows-LanguageFeatures-TextToSpeech-fr-fr-Package~31bf3856ad364e35~amd64~~.cab /PackagePath:E:\Microsoft-Windows-LanguageFeatures-Speech-fr-fr-Package~31bf3856ad364e35~amd64~~.cab
```

Where E:\ is the Feature on Demand ISO.

Example: adding Basic and Handwriting to lb-LU:

```
DISM /Image:"C:\mount\windows" /Add-Package /PackagePath:E:\Microsoft-Windows-LanguageFeatures-Basic-lb-lu-Package~31bf3856ad364e35~amd64~~.cab /PackagePath:E:\Microsoft-Windows-LanguageFeatures-Handwriting-lb-lu-Package~31bf3856ad364e35~amd64~~.cab
```

Example: adding Thai fonts (requires adding th-TH language pack and Basic first).

```
Dism /Image:"C:\mount\windows" Add-Capability /CapabilityName:Language.Fonts.Thai~~und-THAI~0.0.1.0 /Source:E:  
Dism /Get-Capabilities /Image:"C:\mount\windows"
```

Verify that they're in the image:

```
Dism /Image:"C:\mount\windows" /Get-Packages  
Dism /Image:"C:\mount\windows" /Get-Capabilities
```

## 5. Remove languages, LIPs, LXPs, and Features on Demand.

### IMPORTANT

You cannot remove a language pack from an offline Windows image if there are pending online actions. The Windows image should be a recently installed and captured image. This will guarantee that the Windows image does not have any pending online actions that require a reboot.

Get a list of packages and features installed in your image:

```
Dism /Image:"C:\mount\windows" /Get-Packages  
Dism /Image:"C:\mount\windows" /Get-Capabilities
```

Remove languages, .cab-based LIPs, and Features On Demand:

```
Dism /Image:"C:\mount\windows" /Remove-Package /PackageName:<language pack name> /PackageName:<feature on demand name> ...
```

Remove LIPs added through LXPs:

```
Dism /remove-provisionedappxpackage  
/packagename:Microsoft.LanguageExperiencePack<lang_version>_neutral__8wekyb3d8bbwe
```

Verify that they're no longer in your image:

```
Dism /Image:"C:\mount\windows" /Get-Packages  
Dism /Image:"C:\mount\windows" /Get-Capabilities
```

## 6. Change your default language, locale, and other international settings:

- For offline images, you can only specify language packs and .cab-based LIPs, not languages delivered as .appx-based LXPs.
- If you install a LIP by using a .cab file and specify it as the default system UI language, the LIP language will be set as the default UI language (or system locale) and the parent language will be set as the default system UI language (or install language).

```
Dism /Image:"C:\mount\windows" /Set-AllIntl:fr-fr
```

See which language settings are available:

```
Dism /Image:"C:\mount\windows" /Get-Intl
```

- For online images, use Control Panel, or the [International Settings PowerShell cmdlets](#): `Get-WinSystemLocale` and `Set-WinSystemLocale <language>`. This works with languages delivered as LXPs.

For more options, see

- [Default input profiles \(input locales\)](#)
- [Keyboard identifiers and input method editors \(IME\)](#)
- [Time zones](#)
- [DISM Languages and International Servicing Command-Line Options](#).

## Add languages to the recovery environment (Windows RE)

For any language you add to Windows, add the language to the [Windows Recovery Environment \(WinRE\)](#), whenever possible.

Not all languages or LIPs have localized recovery images.

Use languages from the Language Pack ISO, not from the Windows 10 ADK, to localize WinRE.

### NOTE

**Known issue:** Windows RE included with Windows 10 October 2018 Update Thai language image (en-US base language + th-th language pack) shipped with an issue in which an end user will only see UI components (or English) without Thai language text in the Windows Recovery Environment, the text displayed during recovery (Push-button reset), and/or certain screens during the reboot process of feature updates. To fix this issue pre-install the 2018.11D Quality Update.

### 1. Mount the recovery image.

- Offline:** the recovery image is inside the Windows image:

```
md C:\mount\winre  
Dism /Mount-Image /ImageFile:"C:\mount\windows\Windows\System32\Recovery\winre.wim" /index:1  
/MountDir:"C:\mount\winre"
```

- Online:**

```
md C:\mount\winre
Dism /Mount-Image /ImageFile:"C:\Windows\System32\Recovery\winre.wim" /index:1
/MountDir:"C:\mount\winre"
```

Note, Windows RE is inside the Windows image until the Out of Box Experience, after which it is moved to a separate recovery partition.

## 2. Add languages:

```
D:
cd D:\Windows Preinstallation Environment\x64\WinPE_OCs\fr-fr\
Dism /Image:"C:\mount\winre" /Add-Package /PackagePath:"D:\Windows Preinstallation
Environment\x64\WinPE_OCs\fr-fr\lp.cab"
Dism /Image:"C:\mount\winre" /Add-Package /PackagePath:"D:\Windows Preinstallation
Environment\x64\WinPE_OCs\fr-fr\WinPE-Rejuv_fr-fr.cab"
Dism /Image:"C:\mount\winre" /Add-Package /PackagePath:"D:\Windows Preinstallation
Environment\x64\WinPE_OCs\fr-fr\WinPE-EnhancedStorage_fr-fr.cab"
Dism /Image:"C:\mount\winre" /Add-Package /PackagePath:"D:\Windows Preinstallation
Environment\x64\WinPE_OCs\fr-fr\WinPE-Scripting_fr-fr.cab"
Dism /Image:"C:\mount\winre" /Add-Package /PackagePath:"D:\Windows Preinstallation
Environment\x64\WinPE_OCs\fr-fr\WinPE-SecureStartup_fr-fr.cab"
Dism /Image:"C:\mount\winre" /Add-Package /PackagePath:"D:\Windows Preinstallation
Environment\x64\WinPE_OCs\fr-fr\WinPE-SRT_fr-fr.cab"
Dism /Image:"C:\mount\winre" /Add-Package /PackagePath:"D:\Windows Preinstallation
Environment\x64\WinPE_OCs\fr-fr\WinPE-WDS-Tools_fr-fr.cab"
Dism /Image:"C:\mount\winre" /Add-Package /PackagePath:"D:\Windows Preinstallation
Environment\x64\WinPE_OCs\fr-fr\WinPE-WMI_fr-fr.cab"
Dism /Image:"C:\mount\winre" /Add-Package /PackagePath:"D:\Windows Preinstallation
Environment\x64\WinPE_OCs\fr-fr\WinPE-StorageWMI_fr-fr.cab"
Dism /Image:"C:\mount\winre" /Add-Package /PackagePath:"D:\Windows Preinstallation
Environment\x64\WinPE_OCs\fr-fr\WinPE-HTA_fr-fr.cab"
```

Where D:\ is the Language Pack ISO.

## 3. Verify that they're in the image:

```
Dism /Get-Packages /Image:"C:\mount\winre"
```

Example output from /Get-Packages:

```
Package Identity : Microsoft-Windows-WinPE-Rejuv_fr-fr ... fr-FR~10.0.9926.0 State : Installed
```

## 4. Commit changes to the recovery image:

```
Dism /Commit-Image /MountDir:"C:\mount\winre"
```

For online images, also remove the mount folder:

```
rmdir /s "C:\mount\winre"
```

## Capture the changes

- **Offline: For Windows images (install.wim)**

Commit changes to the Windows image:

```
Dism /Commit-Image /MountDir:"C:\mount\windows"
```

- **Online: Modify a running Windows installation**

Generalize and capture the image:

```
%WINDIR%\system32\sysprep\sysprep.exe /generalize /shutdown /oobe
```

To learn more, see [generalize](#).

## Next steps:

### Localize more of Windows:

- [Out of Box Experience \(OOBE\)](#): Create localized end user license agreements (EULA), registration pages, and instructions for pairing keyboards and mice.
- [Retail demo experience](#): Showcase your new devices on the retail sales floor with localized content.

### Localize the installation process

- [Add languages to Windows Setup](#): Now that you've added languages to your Windows image, make sure they appear in Windows Setup. You can also create Windows installation media that works in the user's preferred language.
- [Add languages to WinPE](#) to help your technicians install Windows in their preferred language.
- [Add languages to a Windows distribution share](#) to help users on a corporate network select Windows in their preferred language

## Related topics

- [Features On Demand](#)
- Add languages to [Windows Recovery Environment \(WinRE\)](#)

## Related topics

[Localize Windows](#)

[Service a Windows Image Using DISM](#)

[DISM - Deployment Image Servicing and Management Technical Reference for Windows](#)

[DISM Languages and International Servicing Command-Line Options](#)

[DISM Unattended Servicing Command-Line Options](#)

[Windows System Image Manager Technical Reference](#)

# Available languages for Windows

1/18/2019 • 6 minutes to read

The following tables show the supported language packs for Windows 10 and Windows Server, and supported language interface packs (LIPs) for Windows 10. LIPs are available for Windows 10, but are not available for Windows Server. For more information, see [Language packs](#).

**Windows 10 and Windows Server:** Full language packs are not interchangeable, but some LIPs are. For example, you can add Windows 10, version 1809 LIPs to Windows Server 2019.

The version of the language, LIP, or Feature on Demand must match the version number. For example, you can neither add a Windows 10 version 1809 LIP to Windows 10 version 1803 image, nor to Windows Server 2012.

You cannot add languages to Windows 10 Home Single Language and Windows 10 Home Country Specific editions.

To learn how to add languages to Windows, or where to get them, see [Where to get language packs and LIPs](#).

For a complete list of supported languages and locales, see [Locale Identifier Constants and Strings](#).

## Supported Language Packs and Language Interface Packs

The following tables include these settings:

- Language/region-** The name of the language that will be displayed in the UI. All 38 language packs are available for Windows 10 and Windows Server 2016. In Windows Server 2012 the user interface (UI) is localized only for the 18 languages listed in bold.
- Language/region tag-** The language identifier based on the language tagging conventions of RFC 3066. This setting is used with the Deployment Image Servicing and Management (DISM) tool, or in an unattended answer file.
- Language/region ID-** The hexadecimal representation of the language identifier. This setting is used with the keyboard identifier when specifying an input method using DISM.
- Language/region decimal identifier-** The decimal representation of the language identifier. This setting is used in Oobe.xml.

## Language Packs

LANGUAGE/REGION	LANGUAGE/REGION TAG	LANGUAGE/REGION ID	LANGUAGE/REGION DECIMAL ID
Arabic (Saudi Arabia)	ar-SA	0x0401	1025
Bulgarian (Bulgaria)	bg-BG	0x0402	1026
Chinese (Hong Kong SAR)	zh-HK <small>Note: No longer used. See zh-TW.</small>	0x0c04	3076
Chinese (PRC)	zh-CN	0x0804	2052
Chinese (Taiwan)	zh-TW	0x0404	1028

LANGUAGE/REGION	LANGUAGE/REGION TAG	LANGUAGE/REGION ID	LANGUAGE/REGION DECIMAL ID
Croatian (Croatia)	hr-HR	0x041a	1050
<b>Czech (Czech Republic)</b>	cs-CZ	0x0405	1029
Danish (Denmark)	da-DK	0x0406	1030
<b>Dutch (Netherlands)</b>	nl-NL	0x0413	1043
<b>English (United States)</b>	en-US	0x0409	1033
English (United Kingdom)	en-GB	0x0809	2057
Estonian (Estonia)	et-EE	0x0425	1061
Finnish (Finland)	fi-FI	0x040b	1035
French (Canada)	fr-CA	0x0c0c	3084
<b>French (France)</b>	fr-FR	0x040c	1036
<b>German (Germany)</b>	de-DE	0x0407	1031
Greek (Greece)	el-GR	0x0408	1032
Hebrew (Israel)	he-IL	0x040d	1037
<b>Hungarian (Hungary)</b>	hu-HU	0x040e	1038
<b>Italian (Italy)</b>	it-IT	0x0410	1040
<b>Japanese (Japan)</b>	ja-JP	0x0411	1041
<b>Korean (Korea)</b>	ko-KR	0x0412	1042
Latvian (Latvia)	lv-LV	0x0426	1062
Lithuanian (Lithuania)	lt-LT	0x0427	1063
Norwegian, Bokmål (Norway)	nb-NO	0x0414	1044
<b>Polish (Poland)</b>	pl-PL	0x0415	1045
<b>Portuguese (Brazil)</b>	pt-BR	0x0416	1046
<b>Portuguese (Portugal)</b>	pt-PT	0x0816	2070
Romanian (Romania)	ro-RO	0x0418	1048
<b>Russian (Russia)</b>	ru-RU	0x0419	1049

LANGUAGE/REGION	LANGUAGE/REGION TAG	LANGUAGE/REGION ID	LANGUAGE/REGION DECIMAL ID
Serbian (Latin, Serbia)	sr-Latn-CS <i>Note:</i> No longer used. See sr-Latn-RS.	0x081a	2074
Serbian (Latin, Serbia)	sr-Latn-RS	0x241A	9242
Slovak (Slovakia)	sk-SK	0x041b	1051
Slovenian (Slovenia)	sl-SI	0x0424	1060
Spanish (Mexico)	es-MX	0x080a	2058
<b>Spanish (Spain)</b>	es-ES	0x0c0a	3082
<b>Swedish (Sweden)</b>	sv-SE	0x041d	1053
Thai (Thailand)	th-TH	0x041e	1054
<b>Turkish (Turkey)</b>	tr-TR	0x041f	1055
Ukrainian (Ukraine)	uk-UA	0x0422	1058

## Language interface packs (LIPs)

Except where noted, the following LIPs are available for Windows 10. For Windows Server, options to change keyboard and regional settings such as currency, time zones, and time/date format are available but LIPs are not available. For more information, see [Language packs](#).

### NOTE

As of Windows 10, version 1809, LIPs are distributed as .appx files. To learn how to add an .appx, see [Add languages to Windows](#).

LANGUAGE/REGION	LANGUAGE/REGION TAG	BASE LANGUAGE/REGION	LANGUAGE/REGION ID	LANGUAGE/REGION DECIMAL ID
Afrikaans (South Africa)	af-ZA	Primary: en-US Secondary: en-GB	0x0436	1078
Albanian (Albania)	sq-AL	Primary: en-US Secondary: en-GB	0x041c	1052
Amharic (Ethiopia)	am-ET	Primary: en-US Secondary: en-GB	0x045e	1118

LANGUAGE/REGION	LANGUAGE/REGION TAG	BASE LANGUAGE/REGION	LANGUAGE/REGION ID	LANGUAGE/REGION DECIMAL ID
Armenian (Armenia)	hy-AM	Primary: en-US Secondary: en-GB, ru-RU	0x042b	1067
Assamese (India)	as-IN	Primary: en-US Secondary: en-GB	0x044d	1101
Azerbaijan	az-Latn-AZ	Primary: en-US Secondary: en-GB, ru-RU	0x042c	1068
Bangla (Bangladesh)	bn-BD	Primary: en-US Secondary: en-GB	0x0845	2117
Basque (Basque)	eu-ES	Primary: es-ES Secondary: en-GB, en-US, fr-FR	0x042d	1069
Belarusian	be-BY	Primary: ru-RU Secondary: en-GB, en-US	0x0423	1059
Bangla (India)	bn-IN	Primary: en-US Secondary: en-GB	0x0445	1093
Bosnian (Latin)	bs-Latn-BA	Primary: en-US Secondary: en-GB, hr-HR, sr-Latn-RS	0x141a	5146
Catalan	ca-ES	Primary: es-ES Secondary: en-GB, en-US, fr-FR	0x0403	1027
Central Kurdish	ku-ARAB-IQ	Primary: en-US Secondary: ar-SA, en-GB	0x0492	1170
Cherokee	chr-CHER-US	Primary: en-US Secondary: en-GB	0x045c	1116
Dari	prs-AF	Primary: en-US Secondary: en-GB	0x048c	1164

LANGUAGE/REGION	LANGUAGE/REGION TAG	BASE LANGUAGE/REGION	LANGUAGE/REGION ID	LANGUAGE/REGION DECIMAL ID
Filipino	fil-PH	Primary: en-US Secondary: en-GB	0x0464	1124
Galician	gl-ES	Primary: es-ES Secondary: en-GB, en-US	0x0456	1110
Georgian (Georgia)	ka-GE	Primary: en-US Secondary: en-GB, ru-RU	0x0437	1079
Gujarati (India)	gu-IN	Primary: en-US Secondary: en-GB	0x0447	1095
Hausa (Latin, Nigeria)	ha-Latn-NG	Primary: en-US Secondary: en-GB, fr-FR	0x0468	1128
Hindi (India)	hi-IN	Primary: en-US Secondary: en-GB	0x0439	1081
Icelandic (Iceland)	is-IS	Primary: en-US Secondary: en-GB	0x040f	1039
Igbo (Nigeria)	ig-NG	Primary: en-US Secondary: en-GB	0x0470	1136
Indonesian (Indonesia)	id-ID	Primary: en-US Secondary: en-GB	0x0421	1057
Inuktitut (Latin, Canada)	iu-Latn-CA Not available in Windows 10.	Primary: en-US Secondary: en-GB	0x085d	2141
Irish (Ireland)	ga-IE	Primary: en-US Secondary: en-GB	0x083c	2108
isiXhosa (South Africa)	xh-ZA	Primary: en-US Secondary: en-GB	0x0434	1076
isiZulu (South Africa)	zu-ZA	Primary: en-US Secondary: en-GB	0x0435	1077

LANGUAGE/REGION	LANGUAGE/REGION TAG	BASE LANGUAGE/REGION	LANGUAGE/REGION ID	LANGUAGE/REGION DECIMAL ID
Kannada (India)	kn-IN	Primary: en-US Secondary: en-GB	0x044b	1099
Kazakh (Kazakhstan)	kk-KZ	Primary: en-US Secondary: en-GB, ru-RU	0x043f	1087
Khmer (Cambodia)	km-KH	Primary: en-US Secondary: en-GB	0x0453	1107
K'iche' (Guatemala)	quc-Latn-GT	Primary: es-MX Secondary: es-ES, en-US, en-GB	0x0486	1158
K'iche' (Guatemala)	qut-GT No longer used.	Primary: es-MX Secondary: es-ES, en-US, en-GB	0x0486	1158
Kinyarwanda	rw-RW	Primary: en-US Secondary: en-GB	0x0487	1159
Kiswahili (Kenya)	sw-KE	Primary: en-US Secondary: en-GB	0x0441	1089
Konkani (India)	kok-IN	Primary: en-US Secondary: en-GB	0x0457	1111
Kyrgyz (Kyrgyzstan)	ky-KG	Primary: ru-RU Secondary: en-GB, en-US	0x0440	1088
Lao (Laos)	lo-LA	Primary: en-US Secondary: en-GB	0x0454	1108
Luxembourgish (Luxembourg)	lb-LU	Primary: fr-FR Secondary: de-DE, en-GB, en-US	0x046e	1134
Macedonian (FYROM)	mk-MK	Primary: en-US Secondary: en-GB	0x042f	1071
Malay (Malaysia, Brunei, and Singapore)	ms-MY	Primary: en-US Secondary: en-GB	0x043e	1086

LANGUAGE/REGION	LANGUAGE/REGION TAG	BASE LANGUAGE/REGION	LANGUAGE/REGION ID	LANGUAGE/REGION DECIMAL ID
Malayalam (India)	ml-IN	Primary: en-US Secondary: en-GB	0x044c	1100
Maltese (Malta)	mt-MT	Primary: en-US Secondary: en-GB	0x043a	1082
Maori (New Zealand)	mi-NZ	Primary: en-US Secondary: en-GB	0x0481	1153
Marathi (India)	mr-IN	Primary: en-US Secondary: en-GB	0x044e	1102
Mongolian (Cyrillic)	mn-MN	Primary: en-US Secondary: en-GB, ru-RU	0x0450	1104
Nepali (Federal Democratic Republic of Nepal)	ne-NP	Primary: en-US Secondary: en-GB	0x0461	1121
Norwegian, Nynorsk (Norway)	nn-NO	Primary: nb-NO Secondary: en-GB, en-US	0x0814	2068
Odia (India)	or-IN	Primary: en-US Secondary: en-GB	0x0448	1096
Persian	fa-IR	Primary: en-US Secondary: en-GB	0x0429	1065
Punjabi (India)	pa-IN	Primary: en-US Secondary: en-GB	0x0446	1094
Punjabi (Arabic)	pa-Arab-PK	Primary: en-US Secondary: en-GB	0x0846	2118
Quechua (Peru)	quz-PE	Primary: es-MX Secondary: es-ES, en-GB, en-US	0x0c6b	3179
Scottish Gaelic	gd-GB	Primary: en-US Secondary: en-GB	0x0491	1169

LANGUAGE/REGION	LANGUAGE/REGION TAG	BASE LANGUAGE/REGION	LANGUAGE/REGION ID	LANGUAGE/REGION DECIMAL ID
Serbian (Cyrillic, Bosnia and Herzegovina)	sr-Cyrl-BA	Primary: en-US Secondary: en-GB, sr-Latn-RS	0x1C1A	7194
Serbian (Cyrillic, Serbia)	sr-Cyrl-CS <i>Note:</i> No longer used. See sr-Latn-RS.	Primary: sr-Latn-CS Secondary: en-GB, en-US	0x0c1a	3098
Serbian (Cyrillic, Serbia)	sr-Cyrl-RS	Primary: sr-Latn-RS Secondary: en-GB, en-US	0x281A	10266
Sesotho sa Leboa (South Africa)	nso-ZA	Primary: en-US Secondary: en-GB	0x046c	1132
Setswana (South Africa)	tn-ZA	Primary: en-US Secondary: en-GB	0x0432	1074
Sindhi (Arabic)	sd-Arab-PK	Primary: en-US Secondary: en-GB	0x0859	2137
Sinhala (Sri Lanka)	si-LK	Primary: en-US Secondary: en-GB	0x045b	1115
Tajik (Cyrillic)	tg-Cyrl-TJ	Primary: ru-RU Secondary: en-GB, en-US	0x0428	1064
Tamil (India)	ta-IN	Primary: en-US Secondary: en-GB	0x0449	1097
Tatar (Russia)	tt-RU	Primary: ru-RU Secondary: en-GB, en-US	0x0444	1092
Telugu (India)	te-IN	Primary: en-US Secondary: en-GB	0x044a	1098
Tigrinya	ti-ET	Primary: en-US Secondary: en-GB	0x0473	1139
Turkmen	tk-TM	Primary: ru-RU Secondary: en-GB, en-US	0x0442	1090

LANGUAGE/REGION	LANGUAGE/REGION TAG	BASE LANGUAGE/REGION	LANGUAGE/REGION ID	LANGUAGE/REGION DECIMAL ID
Urdu	ur-PK	Primary: en-US Secondary: en-GB	0x0420	1056
Uyghur	ug-CN	Primary: zh-CN Secondary: en-GB, en-US	0x0480	1152
Uzbek (Latin)	uz-Latn-UZ	Primary: en-US Secondary: en-GB, ru-RU	0x0443	1091
Valencian	ca-ES-valencia	Primary: es-ES Secondary: en-GB, en-US	0x0803	2051
Vietnamese	vi-VN	Primary: en-US Secondary: en-GB	0x042a	1066
Welsh (Great Britain)	cy-GB	Primary: en-US Secondary: en-GB	0x0452	1106
Wolof	wo-SN	Primary: fr-FR Secondary: en-GB, en-US	0x0488	1160
Yoruba (Nigeria)	yo-NG	Primary: en-US Secondary: en-GB	0x046a	1130

## Related topics

[Add Language Packs to Windows](#)

[Windows Language Pack Default Values](#)

[Default Input Locales for Windows Language Packs](#)

# Language and region Features on Demand (FOD)

5/9/2019 • 4 minutes to read

When you add a language pack to a Windows 10 or Windows Server image, you can also add [Features on Demand](#) (FODs) to enable additional functionality. To view available non-language or region-related FODs, see [Available Features on Demand](#).

To see how to add FODs and language packs to your Windows image, see [Add languages to Windows](#).

## Language FODs

**Recommendation:** Add FODs after you add a language pack. Preinstall the Basic, Fonts, Text-to-speech, Optical character recognition (OCR), and speech recognition languages you expect users in the device's target market to need. Additionally, add the handwriting recognition FODs to devices that ship with a pen.

Language FODs are available in six types:

### NOTE

Not all Features on Demand are available for every language. You can learn which FODs are available for languages [in the LP to FOD mapping spreadsheet](#).

COMPONENT	SAMPLE PACKAGE NAME	SAMPLE CAPABILITY NAME	DESCRIPTION
Basic	Microsoft-Windows-LanguageFeatures-Basic-fr-fr-Package.cab	Language.Basic~~~fr-FR~0.0.1.0	<p>Spell checking, text prediction, word breaking, and hyphenation if available for the language.</p> <p>You must add this component before adding any of the other language FODs.</p> <p><b>Recommendation:</b> Preinstall this package for each language that you've preinstalled in an image.</p>
Fonts	Microsoft-Windows-LanguageFeatures-Fonts-Thai-Package.cab	Language.Fonts.Thai~~~und-THAI~0.0.1.0	<p>Fonts.</p> <p><b>Note:</b> Some languages require a font pack. For example, th-TH requires the Thai font pack.</p> <p>Check the <a href="#">list of fonts</a>.</p>

Component	Sample Package Name	Sample Capability Name	Description
OCR	Microsoft-Windows-LanguageFeatures-OCR-fr-fr-Package.cab	Language.OCR~~~fr-FR~0.0.1.0	<p>Recognizes and outputs text in an image.</p> <p><b>Recommendation:</b> Preinstall this package for each language that you've preinstalled in an image.</p> <p><b>Dependencies:</b> The basic component of the same language.</p>
Handwriting recognition	Microsoft-Windows-LanguageFeatures-Handwriting-fr-fr-Package.cab	Language.Handwriting~~~fr-FR~0.0.1.0	<p>Enables handwriting recognition for devices with pen input.</p> <p><b>Recommendation:</b> Preinstall for the device's target language on any device with a touch- or pen-capable screen.</p> <p><b>Dependencies:</b> The basic component of the same language.</p>
Text-to-speech	Microsoft-Windows-LanguageFeatures-TextToSpeech-fr-fr-Package.cab	Language.TextToSpeech~~~fr-FR~0.0.1.0	<p>Enables text to speech, used by Cortana and Narrator.</p> <p><b>Recommendation:</b> Preinstall this package for each language that you've preinstalled in an image.</p> <p><b>Dependencies:</b> The basic component of the same language.</p>
Speech recognition	Microsoft-Windows-LanguageFeatures-Speech-fr-fr-Package.cab	Language.Speech~~~fr-FR~0.0.1.0	<p>Recognizes voice input, used by Cortana and Windows Speech Recognition.</p> <p><b>Recommendation:</b> Preinstall this package for each language that you've preinstalled in an image.</p> <p><b>Dependencies:</b> The basic and text-to-speech components of the same language.</p>

COMPONENT	SAMPLE PACKAGE NAME	SAMPLE CAPABILITY NAME	DESCRIPTION
Retail Demo experience	Microsoft-Windows-RetailDemo-OfflineContent-Content-fr-fr-Package.cab	Language.RetailDemo~~~fr-FR~0.0.1.0	<p><a href="#">Retail Demo experience.</a></p> <p><b>Recommendation:</b> Preinstall this package for each language where you'll be selling devices on a retail showroom. Consider also preloading retail demo mode apps and content.</p> <p><b>Dependencies:</b> The basic component of the same languageBasic, plus the base retail demo pack: Microsoft-Windows-RetailDemo-OfflineContent-Content-Package, and the English retail demo pack: Microsoft-Windows-RetailDemo-OfflineContent-Content-en-us-Package.</p>

## Fonts

When adding languages for some regions, you'll need to add fonts.

Font FODs provide additional fonts used for particular writing systems and languages. These additional fonts are not required for the Windows to display a language. However, prior to Windows 10, most of these fonts were installed by default on every system, and some were used for user interface display. Customers using particular languages are likely to require the additional fonts related to that language for documents or for user interface strings in applications that run in those languages. The capability names use script identifiers; for example, "Deva" indicates Devanagari script, which is used for Hindi, Konkani and other languages.

**Recommendation:** If a PC will be sold to a certain region, install that region's font (For example, th-TH should be preinstalled on devices shipping to regions with Thai language).

REGION	DESCRIPTION	FONT CAPABILITY REQUIRED
am-ET	Amharic	Microsoft-Windows-LanguageFeatures-Fonts-Ethi-Package
ar-SA	Arabic (Saudi Arabia)	Microsoft-Windows-LanguageFeatures-Fonts-Arab-Package
ar-SY	Arabic (Syria)	Microsoft-Windows-LanguageFeatures-Fonts-Syrc-Package
as-IN	Assamese	Microsoft-Windows-LanguageFeatures-Fonts-Beng-Package

REGION	DESCRIPTION	FONT CAPABILITY REQUIRED
bn-BD	Bangla (Bangladesh)	Microsoft-Windows-LanguageFeatures-Fonts-Beng-Package
bn-IN	Bangla (India)	Microsoft-Windows-LanguageFeatures-Fonts-Beng-Package
chr-Cher-US	Cherokee (Cherokee)	Microsoft-Windows-LanguageFeatures-Fonts-Cher-Package
fa-IR	Persian	Microsoft-Windows-LanguageFeatures-Fonts-Arab-Package
gu-IN	Gujarati	Microsoft-Windows-LanguageFeatures-Fonts-Gujr-Package
he-IL	Hebrew	Microsoft-Windows-LanguageFeatures-Fonts-Hebr-Package
hi-IN	Hindi	Microsoft-Windows-LanguageFeatures-Fonts-Deva-Package
ja-JP	Japanese	Microsoft-Windows-LanguageFeatures-Fonts-Jpan-Package
km-KH	Khmer	Microsoft-Windows-LanguageFeatures-Fonts-Khmr-Package
kn-IN	Kannada	Microsoft-Windows-LanguageFeatures-Fonts-Knda-Package
kok-IN	Konkani	Microsoft-Windows-LanguageFeatures-Fonts-Deva-Package
ko-KR	Korean	Microsoft-Windows-LanguageFeatures-Fonts-Kore-Package
ku-Arab-IQ	Central Kurdish (Arabic)	Microsoft-Windows-LanguageFeatures-Fonts-Arab-Package
lo-LA	Lao	Microsoft-Windows-LanguageFeatures-Fonts-Laoo-Package

REGION	DESCRIPTION	FONT CAPABILITY REQUIRED
ml-IN	Malayalam	Microsoft-Windows-LanguageFeatures-Fonts-Mlym-Package
mr-IN	Marathi	Microsoft-Windows-LanguageFeatures-Fonts-Deva-Package
ne-NP	Nepali	Microsoft-Windows-LanguageFeatures-Fonts-Deva-Package
or-IN	Odia	Microsoft-Windows-LanguageFeatures-Fonts-Orya-Package
pa-Arab-PK	Punjabi (Arabic)	Microsoft-Windows-LanguageFeatures-Fonts-Arab-Package
pa-IN	Punjabi	Microsoft-Windows-LanguageFeatures-Fonts-Guru-Package
prs-AF	Dari	Microsoft-Windows-LanguageFeatures-Fonts-Arab-Package
sd-Arab-PK	Sindhi (Arabic)	Microsoft-Windows-LanguageFeatures-Fonts-Arab-Package
si-LK	Sinhala	Microsoft-Windows-LanguageFeatures-Fonts-Sinh-Package
syr-SY	Syriac	Microsoft-Windows-LanguageFeatures-Fonts-Syrc-Package
ta-IN	Tamil	Microsoft-Windows-LanguageFeatures-Fonts-Taml-Package
te-IN	Telugu	Microsoft-Windows-LanguageFeatures-Fonts-Telu-Package
th-TH	Thai	Microsoft-Windows-LanguageFeatures-Fonts-Thai-Package
ti-ET	Tigrinya	Microsoft-Windows-LanguageFeatures-Fonts-Ethi-Package
ug-CN	Uyghur	Microsoft-Windows-LanguageFeatures-Fonts-Arab-Package

Region	Description	Font Capability Required
ur-PK	Urdu	Microsoft-Windows-LanguageFeatures-Fonts-Arab-Package
zh-CN	Chinese (Simplified)	Microsoft-Windows-LanguageFeatures-Fonts-Hans-Package
zh-TW	Chinese Traditional (Hong Kong, Macau and Taiwan)	Microsoft-Windows-LanguageFeatures-Fonts-Hant-Package

## Additional fonts available

These fonts are optional and not required for any region.

Name	Description
Microsoft-Windows-LanguageFeatures-Fonts-PanEuropeanSupplementalFonts-Package	Pan-European Supplemental Fonts. Includes additional fonts: Arial Nova, Georgia Pro, Gill Sans Nova, Neue Haas Grotesk, Rockwell Nova, Verdana Pro.

## Other region-specific requirements

Region	Package	Description	Recommendation
zh-TW	Microsoft-Windows-InternationalFeatures-Taiwan-Package~31bf3856ad364e35~amd64~~.cab	Supplemental support for Taiwan date formatting requirements. Package will be provided to customers located in Taiwan.	Preinstall only on devices shipping to the Taiwan market. Not installing this capability on devices causes any API calls to that use the Taiwan calendar to fail.

Note that this feature is distributed as a .cab file on the Feature on Demand ISO. Use `DISM /add-package` to add it to your image. See [Add or remove packages offline with DISM](#) for more information.

## List of all language-related Features on Demand

[Download the list of all available language FODs](#)

## Related topics

[Available Features on Demand](#)

[Add Language Packs to Windows](#)

[DISM Capabilities Package Servicing Command-Line Options](#)

# International Settings

1/18/2019 • 8 minutes to read

You can specify the default language, locale, and keyboard values during deployment or after Windows is installed.

You can configure international settings by using the International module for Windows PowerShell, by using an answer file with Windows Setup, or by using Deployment Imaging Servicing and Management (DISM).

For information about using DISM to configure international settings in an offline Windows image, see [DISM Languages and International Servicing Command-Line Options](#).

## IMPORTANT

In Windows 10, the intl.cpl command line tools do not support the new settings available in the Region and Language section of Control Panel. For Windows 10, use the International Windows PowerShell cmdlet settings to automate customizing international settings.

In addition, Deployment Imaging Servicing and Management (DISM) should also only be used against an offline Windows image. In Windows 10, language settings are dynamically configured based on the user's language list. Individual settings, such as the display language, default input method, and user locale may be reset dynamically based on user preferences on a running Windows installation. Use the International PowerShell cmdlet settings to change the international settings of a running Windows installation.

## Prepare keyboards, time zones, and other regional settings

You can specify the default keyboard layout, language, or locale, either during deployment or after Windows is installed.

- [Configure International Settings in Windows](#)
- [Default Input Profiles \(Input Locales\) in Windows](#): Lists the default input profiles (language and keyboard pairs) used for each region.
- [Default Time Zones](#): Lists the default time zone used for each region.
- [Keyboard identifiers for Windows](#): Lists the keyboard hexadecimal values used when configuring input profiles.

## Configure international settings by using Windows PowerShell

In Windows 10, you can use the [International Settings PowerShell cmdlets](#) to change the language on a running Windows installation.

1. Open a Windows PowerShell prompt.

2. Display the locale information:

```
Get-WinSystemLocale
```

3. Set the locale for the region and language, for example, to Japanese:

```
Set-WinSystemLocale ja-JP
```

## Configure international settings by using Control Panel

On a running Windows installation, you can use Control Panel to select language packs and configure additional international settings.

1. On the Start page, type **language**, and select **Add a language**.
2. Browse or search for the language that you want to install. For example, select **Catalan**, and then select **Add**.

Catalan is now added as one of your languages.
3. In the **Change your language preferences** pane, select **Options** next to the language that you added.
4. If a language pack is available for your language, select **Download and install language pack**.
5. When the language pack is installed, the language is displayed as available to use for the Windows display language.
6. To make this language your display language, move it to the top of your language list.
7. Log off and then log back on to Windows for the change to take effect.

Installing many additional language packs affects disk space and system performance. In particular, disk space and system performance are affected during servicing operations, such as service pack installations. Therefore, we recommend that you add a language pack to your computer only if you plan to use the language pack.

Language packs also let multiple users who share a computer select distinct display languages. For example, one user can select to see the dialog boxes, menus, and other text in Japanese, whereas another user can select to see the same content in French.

## Configure international settings by using DISM

You can use Deployment Imaging Servicing and Management (DISM) to change the international settings against an offline Windows image

1. Mount a Windows image. For example,

```
Dism /Mount-Image /ImageFile:C:\my_distribution\sources\install.wim /Index:1 /MountDir:C:\mount\windows
```

2. Get the language settings that are configured in the Windows image by using the **/Get-Intl** parameter. For example

```
Dism /image:C:\mount\windows /Get-Intl
```

3. Change the default language, locale, and other international settings by using the **/set-allIntl** parameter.

```
Dism /image:C:\mount\windows /set-allIntl:fr-fr
```

For additional parameters and other options, see [DISM Languages and International Servicing Command-Line Options](#).

## Configure international settings by using an answer file

You can configure international settings in an answer file in the following ways:

- Language Packs are installed from a distribution share and settings are configured installed during **WindowsPE** configuration pass.

Corporations that deploy a multilingual edition of Windows typically create an answer file that configures international settings during the **WindowsPE** configuration pass. For multilingual deployments, language packs can exist in both a distribution share and in the image. You can add and configure Language packs from the distribution share during the **WindowsPE** configuration pass, or you can add these Language packs during the **WindowsPE** configuration pass and configure the settings in another configuration pass.

The Microsoft-Windows-International-Core-WinPE component includes the settings that you can use to modify the language and locale settings during the **windowsPE** configuration pass. Additionally, you can change the Setup UI language for Windows Setup by specifying values in this component.

- Language packs are installed to the Windows image and settings are configured during **specialize** and **oobesystem** configuration passes.

OEMs and corporations that deploy a single-language edition of Windows to various regions typically create an answer file for each region and set the locale and keyboard settings in the **specialize** configuration pass. In this scenario, the language pack is added to the Windows image before international settings are configured.

The Microsoft-Windows-International-Core component includes the settings that you can use to modify the language and locale settings during the **specialize** and **oobeSystem** configuration passes.

You can pre-select a language and skip the Windows Welcome language selection UI page for users by specifying language and locale settings in the **oobeSystem** configuration pass in the Microsoft-Windows-International-Core component. In general, a user can choose between the default Setup language and any additional languages that are installed in the image. The selection of the language will update the other regional settings to the default values that are associated with that language. The user can then individually change the default settings.

### To configure international settings during the Windows PE configuration pass

1. Verify that the necessary language packs are available in the image or in a Windows distribution share. For more information about multilingual distribution shares, see [Add Multilingual Support to a Windows Distribution](#).
2. Open Windows System Image Manager (Windows SIM) and create an answer file. For more information, see [Create or Open an Answer File](#).
3. Add the Microsoft-Windows-International-Core-WinPE component to the answer file to apply settings during the **windowsPE** configuration pass.
4. Configure international settings in the Microsoft-Windows-International-Core-WinPE component. For example, if the Spanish language pack is available in the distribution share, you can add es-ES values to the component settings in the **windowsPE** configuration pass.

Most system locales require a restart. When you configure your locale settings during the **windowsPE** configuration pass, the computer will automatically restart. Additional restarts are not required.

For more information about these settings, see the Microsoft-Windows-International-Core-WinPE components in the Windows® Unattended Setup Reference.

5. Save the answer file and close Windows SIM. The language pack in the distribution share will be automatically added and the international settings will be applied when you run Windows Setup and specify this answer file.

### To configure international settings during the specialize configuration pass

1. Verify that the necessary language packs are available in the image. For more information about how to add a language pack offline, see [Add and Remove Language Packs Offline Using DISM](#). For more information about how to add a language pack using an answer file, see [Add a Package to an Answer File](#).
2. Open Windows SIM and create a new answer file. For more information, see [Create or Open an Answer File](#).
3. Add the Microsoft-Windows-International-Core component to apply settings during the **specialize** and **oobeSystem** configuration passes.

Most system locales require a restart. When you process language settings during the **specialize** or **oobeSystem** configuration passes, the computer might require an additional restart.

4. Edit the settings for the Microsoft-Windows-International-Core component to configure international settings for a specific region. For example, you can add *EN-US* values to the Microsoft-Windows-International-Core settings in the **specialize** configuration pass.

You can also pre-select a language and specify language and locale settings in the **oobeSystem** configuration pass in the Microsoft-Windows-International-Core component. When you do this, the Windows Welcome language selection UI page will be skipped when the users boot to Windows Welcome. In general the user can select between the default Setup language and any additional languages that are installed in the image. The selection of the language will update the other regional settings to the default values associated with that language. The user can then change these default settings individually.

For more information about these settings, see the Microsoft-Windows-International-Core component in the Windows® Unattended Setup Reference.

5. Save the answer file and close Windows SIM. When you run Windows Setup specifying this answer file, the regional settings that you specified in the answer file will be applied.

#### To change international settings in separate configuration passes in the same answer file:

- Create multiple sections in an answer file that will process different language settings during different phases of Windows installation. This enables you to configure multiple language settings in an answer file by specifying different settings to be processed in different configuration passes. For more information, see [How Configuration Passes Work](#).

For example, you can create language and locale settings in the **windowsPE** configuration pass with the Microsoft-Windows-International-Core-WinPE component.

You can then change the default settings in either the **oobeSystem** or the **specialize** configuration pass by adding settings to the Microsoft-Windows-International-Core component.

For example, you can specify *EN-US* as the default language to use on the computer in the **windowsPE** configuration pass. Then, if you intend to send the computer to a different region, you can add more language and locale settings to the **oobeSystem** configuration pass.

If language settings are processed during the **oobeSystem** configuration pass, a restart might be required. Also, the time that is required for the computer to process the language settings might prevent the end user from starting Windows Welcome quickly.

## Related topics

[Windows Setup Technical Reference](#)

[Windows System Image Manager Technical Reference](#)

[Add Language Packs to Windows](#)

## Add and Remove Language Packs Offline Using DISM

# Default Input Profiles (Input Locales) in Windows

1/18/2019 • 13 minutes to read

Input profiles (or input locales) describe the language of the input entered, and the keyboard on which it is being entered. When the first user logs into Windows and identifies their region, Windows sets the input profiles.

The input profiles are made up of a [language identifier](#) and a [keyboard identifier](#). For example, the Arabic (Algerian) input profile is 1401:00020401, where 1401 is the hexadecimal identifier of the language: Arabic (Algeria) and 00020401 is the hexadecimal identifier of the keyboard: Arabic 101.

When the user first identifies the time and date format (User Locale) as Algeria, Windows sets up both the primary input profile, and a secondary input profile: French (France) with French keyboard. The secondary input profile can help the user by providing a keyboard with a Latin character set for tasks that require it, such as filling out email addresses. Some character sets (like CHS IME) have a Latin character set built in.

Windows uses the language component of the input profile for tasks like spelling, hyphenation, and text prediction of the intended key press when using the touch-screen keyboard.

When setting up new devices for your users, you can use the DISM commands: /Set-InputLocale or /Set-AllIntl to identify a default input profile. You can either select the input profile by its language and keyboard pair (1401:00020401) or you can use a language/region tag to receive the default settings for that language/region.

Examples:

```
Dism /Image:"C:\mount\windows" /Set-InputLocale:042d:0000040a
Dism /Image:"C:\mount\windows" /Set-InputLocale:0411:{03B5835F-F03C-411B-9CE2-AA23E1171E36}{A76C93D9-5523-4E90-AAFA-4DB112F9AC76}
Dism /Image:"C:\mount\windows" /Set-InputLocale:id-ID
Dism /Image:"C:\mount\windows" /Set-AllIntl:fr-FR
```

For a list of language/region names, see [Available Language Packs for Windows](#).

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
Afrikaans - South Africa	af-ZA: United States - English (0436:00000409)	
Albanian - Albania	sq-AL: Albanian (041c:0000041c)	
Alsatian - France	gsw-FR: French (0484:0000040c)	
Amharic - Ethiopia	am-ET: Amharic Input Method (045e:{E429B25A-E5D3-4D1F-9BE3-0C608477E3A1}{8F96574E-C86C-4bd6-9666-3F7327D4CBE8})	en-US: United States - English (0409:00000409)
Arabic - Algeria	ar-DZ: Arabic (102) AZERTY (1401:00020401)	fr-FR: French (040c:0000040c)
Arabic - Bahrain	ar-BH: Arabic (101) (3c01:00000401)	en-US: United States - English (0409:00000409)

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
Arabic - Egypt	ar-EG: Arabic (101) (0c01:00000401)	en-US: United States - English (0409:00000409)
Arabic - Iraq	ar-IQ: Arabic (101) (0801:00000401)	en-US: United States - English (0409:00000409)
Arabic - Jordan	ar-JO: Arabic (101) (2c01:00000401)	en-US: United States - English (0409:00000409)
Arabic - Kuwait	ar-KW: Arabic (101) (3401:00000401)	en-US: United States - English (0409:00000409)
Arabic - Lebanon	ar-LB: Arabic (101) (3001:00000401)	en-US: United States - English (0409:00000409)
Arabic - Libya	ar-LY: Arabic (101) (1001:00000401)	en-US: United States - English (0409:00000409)
Arabic - Morocco	ar-MA: Arabic (102) AZERTY (1801:00020401)	fr-FR: French (040c:0000040c)
Arabic - Oman	ar-OM: Arabic (101) (2001:00000401)	en-US: United States - English (0409:00000409)
Arabic - Qatar	ar-QA: Arabic (101) (4001:00000401)	en-US: United States - English (0409:00000409)
Arabic - Saudi Arabia	ar-SA: Arabic (101) (0401:00000401)	en-US: United States - English (0409:00000409)
Arabic - Syria	ar-SY: Arabic (101) (2801:00000401)	en-US: United States - English (0409:00000409)
Arabic - Tunisia	ar-TN: Arabic (102) AZERTY (1c01:00020401)	fr-FR: French (040c:0000040c)
Arabic - U.A.E.	ar-AE: Arabic (101) (3801:00000401)	en-US: United States - English (0409:00000409)
Arabic - Yemen	ar-YE: Arabic (101) (2401:00000401)	en-US: United States - English (0409:00000409)
Armenian - Armenia	hy-AM: Armenian Phonetic (042b:0002042b)	hy-AM: Armenian Typewriter (042b:0003042b) ru-RU: Russian (0419:00000419)
Assamese - India	as-IN: Assamese - Inscript (044d:0000044d)	en-US: United States - English (0409:00000409)

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
Azerbaijani - Azerbaijan (Cyrillic)	az-Cyrl-AZ: Azerbaijani Cyrillic (082c:0000082c)	en-US: United States - English (0409:00000409) az-Latn-AZ: Azeri Latin (042c:0000042c)
Azerbaijani - Azerbaijan (Latin)	az-Latn-AZ: Azerbaijani Latin (042c:0000042c)	en-US: United States - English (0409:00000409) az-Cyrl-AZ: Azeri Cyrillic (082c:0000082c)
Bangla (Bangladesh)	bn-BD: Bangla - Bangladesh (0845:00000445)	en-US: United States - English (0409:00000409)
Bangla - India (Bengali Script)	bn-IN: Bangla India-INSCRIPT (0445:00020445)	en-US: United States - English (0409:00000409)
Bashkir - Russia	ba-RU: Bashkir (046d:0000046d)	ru-RU Russian (0419:00000419) en-US: United States - English (0409:00000409)
Basque - Basque	eu-ES: Spanish (042d:0000040a)	
Belarusian - Belarus	be-BY: Belarusian (0423:00000423)	ru-RU Russian (0419:00000419) en-US: United States - English (0409:00000409)
Bosnian - Bosnia and Herzegovina (Cyrillic)	bs-Cyrl-BA: Bosnian (Cyrillic) (201a:0000201a)	bs-Latn-BA: Croatian (141a:0000041a)
Bosnian - Bosnia and Herzegovina (Latin)	bs-Latn-BA: Croatian (141a:0000041a)	
Breton - France	br-FR: French (047e:0000040c)	
Bulgarian - Bulgaria	bg-BG: Bulgarian (0402:00030402)	en-US: United States - International (0409:00020409)
Burmese - Myanmar	my-MM: Myanmar (0455:00010c00)	en-US: United States - English (0409:00000409)
Catalan - Catalan	ca-ES: Spanish (0403:0000040a)	
Central Atlas Tamazight (Latin) - Algeria	fr-FR: French (040c:0000040c)	en-US: United States - English (0409:00000409)
Central Atlas Tamazight (Latin) - Algeria	tzm-Latn-DZ: Central Atlas Tamazight (085f:0000085f)	

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
Central Atlas Tamazight (Tifinagh) - Morocco	tzm-Tfng-MA: (105f:0000105f)	fr-FR: French (040c:0000040c)
Central Kurdish (Iraq)	ku-Arab-IQ: (0492:00000492)	en-US: United States - English (0409:00000409)
Cherokee (Cherokee, United States)	chr-Cher-US: Cherokee Nation (045c:0000045c)	Cherokee Nation Phonetic (045c:0001045c) en-US: United States - English (0409:00000409)
Chinese - PRC	zh-CN: Microsoft Pinyin - Simple Fast (0804:{81D4E9C9-1D3B-41BC-9E6C-4B40BF79E35E}{FA550B04-5AD7-411f-A5AC-CA038EC515D7})	
Chinese - Taiwan	zh-TW: Chinese (Traditional) - New Phonetic (0404:{B115690A-EA02-48D5-A231-E3578D2FDF80}{B2F9C502-1742-11D4-9790-0080C882687E})	
Corsican - France	co-FR: French (0483:0000040c)	
Croatian - Bosnia and Herzegovina	hr-BA: Croatian (101a:0000041a)	
Croatian - Croatia	hr-HR: Croatian (041a:0000041a)	
Czech - Czech Republic	cs-CZ: Czech (0405:00000405)	
Danish - Denmark	da-DK: Danish (0406:00000406)	en-US: Danish (0409:00000406)
Dari - Afghanistan	prs-AF: Persian (Standard) (048c:00050429)	en-US: United States - English (0409:00000409)
Divehi - Maldives	dv-MV: Divehi Phonetic (0465:00000465)	en-US: United States - English (0409:00000409)
Dutch - Belgium	nl-BE: Belgian (Period) (0813:00000813)	
Dutch - Netherlands	nl-NL: United States - International (0413:00020409)	
Dzongkha	dz-BT: 0C51:00000C51;0409:00000409	en-US: United States - English (0409:00000409)
English - Australia	en-AU: United States - English (0c09:00000409)	
English - Belize	en-BZ: United States - English (2809:00000409)	

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
English - Canada	en-CA: United States - English (1009:00000409)	en-CA: Canadian Multilingual Standard (1009:00011009)
English - Caribbean	en-029: United States - English (2409:00000409)	
English - India	en-IN: India (4009:00004009)	
English - Ireland	en-IE: Irish (1809:00001809)	
English - Jamaica	en-JM: United States - English (2009:00000409)	
English - Malaysia	en-MY: United States - English (4409:00000409)	
English - New Zealand	en-NZ: United States - English (1409:00000409)	
English - Philippines	en-PH: United States - English (3409:00000409)	
English - Singapore	en-SG: United States - English (4809:00000409)	
English - South Africa	en-ZA: United States - English (1c09:00000409)	
English - Trinidad	en-TT: United States - English (2c09:00000409)	
English - Great Britain	en-GB: Great Britain (0809:00000809)	
English - United States	en-US: United States - English (0409:00000409)	
English - Zimbabwe	en-ZW: United States - English (3009:00000409)	
Estonian - Estonia	et-EE: Estonian (0425:00000425)	
Faroese - Faroe Islands	fo-FO: Danish (0438:00000406)	
Filipino - Philippines	fil-PH: United States - English (0464:00000409)	
Finnish - Finland	fi-FI: Finnish (040b:0000040b)	
French - Belgium	fr-BE: Belgian French (080c:0000080c)	
French - Canada	fr-CA: Canadian Multilingual Standard (0c0c:00011009)	en-CA: Canadian Multilingual Standard (1009:00011009)

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
French - France	fr-FR: French (040c:0000040c)	
French - Luxembourg	fr-LU: Swiss French (140c:0000100C)	fr-LU: French (140c:0000040d)
French - Monaco	fr-MC: French (180c:0000040c)	
French - Switzerland	fr-CH: Swiss French (100c:0000100c)	de-CH: Swiss German (0807:00000807)
Frisian - Netherlands		
Fulah (Latin, Senegal)	ff-Latn-SN: Wolof (0867:00000488)	
Galician - Galician	gl-ES: Spanish (0456:0000040a)	
Georgian - Georgia	ka-GE: Georgian (QWERTY) (0437:00010437)	en-US: United States - English (0409:00000409)
German - Austria	de-AT: German (0c07:00000407)	de-AT: German ()
German - Germany	de-DE: German (0407:00000407)	de-DE: German ()
German - Liechtenstein	de-LI: Swiss German (1407:00000807)	
German - Luxembourg	de-LU: German (1007:00000407)	
German - Switzerland	de-CH: Swiss German (0807:00000807)	fr-CH: Swiss French (100C:0000100C)
Greek - Greece	el-GR: Greek (0408:00000408)	en-US: United States - English (0409:00000409)
Greenlandic - Greenland	kl-GL: Danish (046f:00000406)	
Guarani - Paraguay	gn-PY: Guarani (0474:00000474)	
Gujarati - India (Gujarati Script)	gu-IN: Gujarati (0447:00000447)	en-US: United States - English (0409:00000409)
Hausa (Latin) - Nigeria	ha-Latn-NG: Hausa (0468:00000468)	
Hawaiian - United States	haw-US: (0475:00000475)	en-US: United States - English (0409:00000409)
Hebrew - Israel	he-IL: (040d:0002040d)	en-US: United States - English (0409:00000409)
Hindi - India	hi-IN: Hindi Traditional (0439:00010439)	en-US: United States - English (0409:00000409)

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
Hungarian - Hungary	hu-HU: Hungarian (040e:0000040e)	
Icelandic - Iceland	is-IS: Icelandic (040f:0000040f)	
Igbo - Nigeria	ig-NG: Igbo (0470:00000470)	
Inari Sami - Finland	smn-FI: Finnish with Sami (243b:0001083b)	
Indonesian - Indonesia	id-ID: United States - English (0421:00000409)	
Inuktitut (Latin) - Canada	iu-Latn-CA: Inuktitut - Latin (085d:0000085d)	en-CA: United States - English (1009:00000409)
Inuktitut (Syllabics) - Canada	iu-Cans-CA: Inuktitut - Naqittaut (045d:0001045d)	en-CA: United States - English (1009:00000409)
Irish - Ireland	ga-IE: Irish (083c:00001809)	
isiXhosa / Xhosa - South Africa	xh-ZA: United States - English (0434:00000409)	
isiZulu / Zulu - South Africa	zu-ZA: United States - English (0435:00000409)	
Italian - Italy	it-IT: Italian (0410:00000410)	
Italian - Switzerland	it-CH: Swiss French (0810:0000100c)	it-CH: Italian (0810:00000410)
Japanese - Japan	ja-JP: Microsoft IME (0411:{03B5835F-F03C-411B-9CE2-AA23E1171E36}{A76C93D9-5523-4E90-AAFA-4DB112F9AC76})	
Javanese (Latin) - Indonesia	jv-Latn-ID: US (0c00:00000409)	
Kannada - India (Kannada Script)	kn-IN: Kannada (044b:0000044b)	en-US: United States - English (0409:00000409)
Kazakh - Kazakhstan	kk-KZ: Kazakh (043f:0000043f)	en-US: United States - English (0409:00000409)
Khmer - Cambodia	km-KH: Khmer (0453:00000453)	en-US: United States - English (0409:00000409)
K'iche - Guatemala	qut-GT: Latin American (0486:0000080a)	
Kinyarwanda - Rwanda	rw-RW: United States - English (0487:00000409)	

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
Konkani - India	kok-IN: Devanagari-INSCRIPT (0457:00000439)	en-US: United States - English (0409:00000409)
Korean(Extended Wansung) - Korea	ko-KR: Microsoft IME (0412:{A028AE76-01B1-46C2-99C4-ACD9858AE02F}{B5FE1F02-D5F2-4445-9C03-C568F23C99A1})	
Kyrgyz - Kyrgyzstan	ky-KG: Kyrgyz Cyrillic (0440:00000440)	en-US: United States - English (0409:00000409)
Lao - Lao PDR	lo-LA: Lao (0454:00000454)	en-US: United States - English (0409:00000409)
Latvian - Legacy	lv-LV: Latvian (QWERTY) (0426:00010426)	
Latvian - Standard	lv-LV: Latvian (Standard) (0426:00020426)	
Lithuanian - Lithuania	lt-LT: Lithuanian (0427:00010427)	
Lower Sorbian - Germany	dsb-DE: Sorbian Standard (082e:0002042e)	
Lule Sami - Norway	smj-NO: Norwegian with Sami (103b:0000043b)	
Lule Sami - Sweden	smj-SE: Swedish with Sami (143b:0000083b)	
Luxembourgish - Luxembourg	lb-LU: Luxembourgish (046e:0000046e)	
Macedonian - F.Y.R.O.M	mk-MK: Macedonia (FYROM) - Standard (042f:0001042f)	en-US: United States - English (0409:00000409)
Malay - Brunei	ms-BN: United States - English (083e:00000409)	
Malay - Malaysia	ms-MY: United States - English (043e:00000409)	
Malayalam - India (Malayalam Script)	ml-IN: Malayalam (044c:0000044c)	en-US: United States - English (0409:00000409)
Maltese - Malta	mt-MT: Maltese 47-Key (043a:0000043a)	
Maori - New Zealand	mi-NZ: Maori (0481:00000481)	en-NZ: United States - English (1409:00000409)

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
Mapudungun - Chile	arn-CL: Latin American (047a:0000080a)	
Marathi - India	mr-IN: Marathi (044e:0000044e)	en-US: United States - English (0409:00000409)
Mohawk - Mohawk	moh-CA: United States - English (047c:00000409)	
Mongolian (Cyrillic) - Mongolia	mn-MN: Mongolian Cyrillic (0450:00000450)	en-US: United States - English (0409:00000409)
Mongolian (Mongolian) - Mongolia	mn-Mong-MN: Traditional Mongolian (Standard) (0c50:00010850)	en-US: United States - English (0409:00000409)
Mongolian (Mongolian – PRC – Legacy)	mn-Mong-CN: Mongolian (Mongolian Script) (0850:00000850)	en-US: United States - English (0409:00000409)
Mongolian (Mongolian– PRC – Standard)	mn-Mong-CN: Mongolian (Mongolian Script) (0850:00010850)	en-US: United States - English (0409:00000409)
N'ko – Guinea	nqo-GN: N'Ko (0c00:00090C00)	en-US: United States - English (0409:00000409)
Nepali - Federal Democratic Republic of Nepal	ne-NP: Nepali (0461:00000461)	en-US: United States - English (0409:00000409)
Northern Sami - Finland	se-FI: Finnish with Sami (0c3b:0001083b)	
Northern Sami - Norway	se-NO: Norwegian with Sami (043b:0000043b)	
Northern Sami - Sweden	se-SE: Swedish with Sami (083b:0000083b)	
Norwegian - Norway (Bokmål)	nb-NO: Norwegian (0414:00000414)	
Norwegian - Norway (Nynorsk)	nn-NO: Norwegian (0814:00000414)	
Occitan - France	oc-FR: French (0482:0000040c)	
Odia - India (Odia Script)	or-IN: Odia (0448:00000448)	en-US: United States - English (0409:00000409)
Pashto - Afghanistan	ps-AF: Pashto (Afghanistan) (0463:00000463)	en-US: United States - English (0409:00000409)

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
Persian	fa-IR: Central Kurdish (0429:00000429)	Fa-IR: Persian (Standard) (0429:00050429) en-US: United States - English (0409:00000409)
Polish - Poland	pl-PL: Polish (Programmers) (0415:00000415)	
Portuguese - Brazil	pt-BR: Portuguese (Brazilian ABNT) (0416:00000416)	
Portuguese - Portugal	pt-PT: Portuguese (0816:00000816)	
Punjabi - India (Gurmukhi Script)	pa-IN: Punjabi (0446:00000446)	en-US: United States - English (0409:00000409)
Punjabi (Islamic Republic of Pakistan)	pa-Arab-PK: Urdu (0846:00000420)	en-US: United States - English (0409:00000409)
Quechua - Bolivia	quz-BO: Latin American (046b:0000080a)	
Quechua - Ecuador	quz-EC: Latin American (086b:0000080a)	
Quechua - Peru	quz-PE: Latin American (0c6b:0000080a)	
Romanian - Romania	ro-RO: Romanian (Standard) (0418:00010418)	
Romansh - Switzerland	rm-CH: Swiss German (0417:00000807)	
Russian - Russia	ru-RU: Russian (0419:00000419)	en-US: United States - English (0409:00000409)
Sakha - Russia	sah-RU: Sakha (0485:00000485)	ru-RU Russian (0419:00000419) en-US: United States - English (0409:00000409)
Sanskrit - India	sa-IN: Devanagari-INSCRIPT (044f:00000439)	en-US: United States - English (0409:00000409)
Scottish Gaelic - Great Britain	gd-GB: Gaelic (0491:00011809)	
Serbian - Bosnia and Herzegovina (Cyrillic)	sr-Cyrl-BA: Serbian (Cyrillic) (1c1a:00000c1a)	en-US: United States - English (0409:00000409)

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
Serbian - Bosnia and Herzegovina (Latin)	sr-Latn-BA: Serbian (Latin) (181a:0000081a)	
Serbian - Montenegro (Cyrillic)	sr-Cyrl-ME: Serbian (Cyrillic) (301a:00000c1a)	en-US: United States - International (0409:00020409)
Serbian - Montenegro (Latin)	sr-Latn-ME: Serbian (Latin) (2c1a:0000081a)	
Serbian - Serbia (Cyrillic)	sr-Cyrl-RS: Serbian (Cyrillic) (281a:00000c1a)	en-US: United States - International (0409:00020409)
Serbian - Serbia (Latin)	sr-Latn-RS: Serbian (Latin) (241a:0000081a)	
Serbian - Serbia and Montenegro (Former) (Cyrillic)	sr-Cyrl-CS: Serbian (Cyrillic) (0c1a:00000c1a)	en-US: United States - English (0409:00000409)
Serbian - Serbia and Montenegro (Former) (Latin)	sr-Latn-CS: Serbian (Latin) (081a:0000081a)	
Sesotho sa Leboa / Northern Sotho - South Africa	nso-ZA: Sesotho sa Leboa (046c:0000046c)	
Setswana / Tswana - Botswana	tn-BW: Setswana (0832:00000432)	
Setswana / Tswana - South Africa	tn-ZA: Setswana (0432:00000432)	
Shona – Zimbabwe	sn-Latn-ZW: US (0c00:00000409)	
Sindhi (Islamic Republic of Pakistan)	sd-Arab-PK: Urdu (0859:00000420)	en-US: United States - English (0409:00000409)
Sinhala - Sri Lanka	si-LK: Sinhala (045b:0000045b)	en-US: United States - English (0409:00000409)
Skolt Sami - Finland	sms-FI: Finnish with Sami (203b:0001083b)	
Slovak - Slovakia	sk-SK: Slovak (041b:0000041b)	
Slovenian - Slovenia	sl-SI: Slovenian (0424:00000424)	
Southern Sami - Norway	sma-NO: Norwegian with Sami (183b:0000043b)	
Southern Sami - Sweden	sma-SE: Swedish with Sami (1c3b:0000083b)	
Spanish - Argentina	es-AR: Latin American (2c0a:0000080a)	

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
Spanish - Bolivarian Republic of Venezuela	es-VE: Latin American (200a:0000080a)	
Spanish - Bolivia	es-BO: Latin American (400a:0000080a)	
Spanish - Chile	es-CL: Latin American (340a:0000080a)	
Spanish - Colombia	es-CO: Latin American (240a:0000080a)	
Spanish - Costa Rica	es-CR: Latin American (140a:0000080a)	
Spanish - Dominican Republic	es-DO: Latin American (1c0a:0000080a)	
Spanish - Ecuador	es-EC: Latin American (300a:0000080a)	
Spanish - El Salvador	es-SV: Latin American (440a:0000080a)	
Spanish - Guatemala	es-GT: Latin American (100a:0000080a)	
Spanish - Honduras	es-HN: Latin American (480a:0000080a)	
Spanish - Latin America	es-419: Latin American (580a:0000080a)	
Spanish - Mexico	es-MX: Latin American (080a:0000080a)	
Spanish - Nicaragua	es-NI: Latin American (4c0a:0000080a)	
Spanish - Panama	es-PA: Latin American (180a:0000080a)	
Spanish - Paraguay	es-PY: Latin American (3c0a:0000080a)	
Spanish - Peru	es-PE: Latin American (280a:0000080a)	
Spanish - Commonwealth of Puerto Rico	es-PR: Latin American (500a:0000080a)	
Spanish - Spain (International Sort)	es-ES: Spanish (0c0a:0000040a)	en-US: United States - English (0409:00000409)
Spanish - Spain (Traditional Sort)	es-ES_tradnl: Spanish (040a:0000040a)	
Spanish - United States	es-US: Latin American (540a:0000080a)	en-US: United States - English (0409:00000409)

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
Spanish - Uruguay	es-UY: Latin American (380a:0000080a)	
Standard Moroccan Tamazight - Morocco	zgh-Tfng-MA: Tifinagh (Basic) (0c00:0000105F)	en-US: United States - English (0409:00000409)
Swahili - Kenya	sw-KE: United States - English (0441:00000409)	
Swedish - Finland	sv-FI: Swedish (081d:0000041d)	
Swedish - Sweden	sv-SE: Swedish (041d:0000041d)	
Syriac - Syria	syr-SY: Syriac (045a:0000045a)	en-US: United States - English (0409:00000409)
Tajik - Tajikistan	tg-Cyrl-TJ: Tajik (0428:00000428)	en-US: United States - English (0409:00000409)
Tamil - India	ta-IN: Tamil (0449:00000449)	en-US: United States - English (0409:00000409)
Tamil - Sri Lanka	ta-LK: Tamil (0849:00000449)	en-US: United States - English (0409:00000409)
Tatar – Russia (Legacy)	tt-RU: Tatar (0444:00000444)	ru-RU: Russian (0419:00000419) en-US: United States - English (0409:00000409)
Tatar – Russia (Standard)	tt-RU: Tatar (0444:00010444)	ru-RU: Russian (0419:00000419) en-US: United States - English (0409:00000409)
Telugu - India (Telugu Script)	te-IN: Telugu (044a:0000044a)	en-US: United States - English (0409:00000409)
Thai - Thailand	th-TH: Thai Kedmanee (041e:0000041e)	en-US: United States - English (0409:00000409)
Tibetan - PRC	bo-CN: Tibetan (PRC) (0451:00010451)	en-US: United States - English (0409:00000409)
Tigrinya (Eritrea)	ti-ET: Tigrinya Input Method (0473:{E429B25A-E5D3-4D1F-9BE3-0C608477E3A1}{3CAB88B7-CC3E-46A6-9765-B772AD7761FF})	en-US: United States - English (0409:00000409)
Tigrinya (Ethiopia)	ti-ET: Tigrinya Input Method (0473:{E429B25A-E5D3-4D1F-9BE3-0C608477E3A1}{3CAB88B7-CC3E-46A6-9765-B772AD7761FF})	en-US: United States - English (0409:00000409)

LANGUAGE/REGION	PRIMARY INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)	SECONDARY INPUT PROFILE
Turkish - Turkey	tr-TR: Turkish Q (041f:0000041f)	
Turkmen - Turkmenistan	tk-TM: Turkmen (0442:00000442)	en-US: United States - English (0409:00000409)
Ukrainian - Ukraine	uk-UA: Ukrainian (Enhanced) (0422:00020422)	en-US: United States - English (0409:00000409)
Upper Sorbian - Germany	hsb-DE: Sorbian Standard (042e:0002042e)	
Urdu – India	ur-IN: Urdu (0820:00000420)	en-US: United States - English (0409:00000409)
Urdu (Islamic Republic of Pakistan)	ur-PK: Urdu (0420:00000420)	en-US: United States - English (0409:00000409)
Uyghur - PRC	ug-CN: Uyghur (0480:00010480)	en-US: United States - English (0409:00000409)
Uzbek - Uzbekistan (Cyrillic)	uz-Cyrl-UZ: Uzbek Cyrillic (0843:00000843)	uz-Latn-UZ: United States - English (0443:00000409)
Uzbek - Uzbekistan (Latin)	uz-Latn-UZ: United States - English (0443:00000409)	
Valencian - Valencia	ca-ES-valencia: Spanish (0803:0000040a)	
Vietnamese - Vietnam	vi-VN: Vietnamese (042a:0000042a)	en-US: United States - English (0409:00000409)
Welsh - Great Britain	cy-GB: Great Britain Extended (0452:00000452)	en-GB: Great Britain (0809:00000809)
Wolof - Senegal	wo-SN: Wolof (0488:00000488)	
Yi - PRC	ii-CN: Yi Input Method(0478:{E429B25A-E5D3-4D1F-9BE3-0C608477E3A1}{409C8376-007B-4357-AE8E-26316EE3FB0D})	zh-CN: Microsoft Pinyin - Simple Fast (0804:{81D4E9C9-1D3B-41BC-9E6C-4B40BF79E35E}{FA550B04-5AD7-411f-A5AC-CA038EC515D7})
Yoruba - Nigeria	yo-NG: Yoruba (046a:0000046a)	

## Related topics

[Default Time Zones](#)

[Add Language Packs to Windows](#)

[Available Language Packs for Windows](#)

[Keyboard identifiers for Windows](#)



# Default Time Zones

1/18/2019 • 13 minutes to read

When the first user logs into Windows 10 or Windows Server and identifies their region, Windows sets the time zone. The user can change the time zone at any time.

Windows updates the time zones in the registry when time zones are available and updates are downloaded.

**To get the most current list of time zones, use [tzutil from Windows](#):**

```
tzutil /l
```

To change the default time zone in an offline image:

```
Dism /Image:"C:\mount\windows" /Set-TimeZone:"W. Europe Standard Time"
```

Country	ISO3166	Timezone	UTC	Timezone description
Afghanistan	AF	Afghanistan Standard Time	(UTC+04:30)	Kabul
Åland Islands	AX	FLE Standard Time	(UTC+02:00)	Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius
Albania	AL	Central Europe Standard Time	(UTC+01:00)	Belgrade, Bratislava, Budapest, Ljubljana, Prague
Algeria	DZ	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
American Samoa	AS	UTC-11	(UTC-11:00)	Coordinated Universal Time-11
Andorra	AD	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Angola	AO	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
Anguilla	AI	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Antarctica	AQ	Pacific SA Standard Time	(UTC-03:00)	Santiago
Antigua and Barbuda	AG	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan

Argentina	AR	Argentina Standard Time	(UTC-03:00)	City of Buenos Aires
Armenia	AM	Caucasus Standard Time	(UTC+04:00)	Yerevan
Aruba	AW	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Australia	AU	AUS Eastern Standard Time	(UTC+10:00)	Canberra, Melbourne, Sydney
Austria	AT	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Azerbaijan	AZ	Azerbaijan Standard Time	(UTC+04:00)	Baku
Bahamas, The	BS	Eastern Standard Time	(UTC-05:00)	Eastern Time (US & Canada)
Bahrain	BH	Arab Standard Time	(UTC+03:00)	Kuwait, Riyadh
Bangladesh	BD	Bangladesh Standard Time	(UTC+06:00)	Dhaka
Barbados	BB	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Belarus	BY	Belarus Standard Time	(UTC+03:00)	Minsk
Belgium	BE	Romance Standard Time	(UTC+01:00)	Brussels, Copenhagen, Madrid, Paris
Belize	BZ	Central America Standard Time	(UTC-06:00)	Central America
Benin	BJ	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
Bermuda	BM	Atlantic Standard Time	(UTC-04:00)	Atlantic Time (Canada)
Bhutan	BT	Bangladesh Standard Time	(UTC+06:00)	Dhaka
Bolivarian Republic of Venezuela	VE	Venezuela Standard Time	(UTC-04:30)	Caracas
Bolivia	BO	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan

Bonaire, Sint Eustatius and Saba	BQ	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Bosnia and Herzegovina	BA	Central European Standard Time	(UTC+01:00)	Sarajevo, Skopje, Warsaw, Zagreb
Botswana	BW	South Africa Standard Time	(UTC+02:00)	Harare, Pretoria
Bouvet Island	BV	UTC	(UTC)	Coordinated Universal Time
Brazil	BR	E. South America Standard Time	(UTC-03:00)	Brasilia
British Indian Ocean Territory	IO	Central Asia Standard Time	(UTC+06:00)	Astana
Brunei	BN	Singapore Standard Time	(UTC+08:00)	Kuala Lumpur, Singapore
Bulgaria	BG	FLE Standard Time	(UTC+02:00)	Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius
Burkina Faso	BF	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Burundi	BI	South Africa Standard Time	(UTC+02:00)	Harare, Pretoria
Cabo Verde	CV	Cape Verde Standard Time	(UTC-01:00)	Cabo Verde Is.
Cambodia	KH	SE Asia Standard Time	(UTC+07:00)	Bangkok, Hanoi, Jakarta
Cameroon	CM	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
Canada	CA	Eastern Standard Time	(UTC-05:00)	Eastern Time (US & Canada)
Cayman Islands	KY	SA Pacific Standard Time	(UTC-05:00)	Bogota, Lima, Quito, Rio Branco
Central African Republic	CF	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
Chad	TD	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
Chile	CL	Pacific SA Standard Time	(UTC-03:00)	Santiago

China	CN	China Standard Time	(UTC+08:00)	Beijing, Chongqing, Hong Kong, Urumqi
Christmas Island	CX	SE Asia Standard Time	(UTC+07:00)	Bangkok, Hanoi, Jakarta
Cocos (Keeling) Islands	CC	Myanmar Standard Time	(UTC+06:30)	Yangon (Rangoon)
Colombia	CO	SA Pacific Standard Time	(UTC-05:00)	Bogota, Lima, Quito, Rio Branco
Comoros	KM	E. Africa Standard Time	(UTC+03:00)	Nairobi
Congo	CG	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
Congo (DRC)	CD	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
Cook Islands	CK	Hawaiian Standard Time	(UTC-10:00)	Hawaii
Costa Rica	CR	Central America Standard Time	(UTC-06:00)	Central America
Côte d'Ivoire	CI	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Croatia	HR	Central European Standard Time	(UTC+01:00)	Sarajevo, Skopje, Warsaw, Zagreb
Cuba	CU	Eastern Standard Time	(UTC-05:00)	Eastern Time (US & Canada)
Curaçao	CW	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Cyprus	CY	E. Europe Standard Time	(UTC+02:00)	E. Europe
Czech Republic	CZ	Central Europe Standard Time	(UTC+01:00)	Belgrade, Bratislava, Budapest, Ljubljana, Prague
Democratic Republic of Timor-Leste	TL	Tokyo Standard Time	(UTC+09:00)	Osaka, Sapporo, Tokyo
Denmark	DK	Romance Standard Time	(UTC+01:00)	Brussels, Copenhagen, Madrid, Paris

Djibouti	DJ	E. Africa Standard Time	(UTC+03:00)	Nairobi
Dominica	DM	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Dominican Republic	DO	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Ecuador	EC	SA Pacific Standard Time	(UTC-05:00)	Bogota, Lima, Quito, Rio Branco
Egypt	EG	Egypt Standard Time	(UTC+02:00)	Cairo
El Salvador	SV	Central America Standard Time	(UTC-06:00)	Central America
Equatorial Guinea	GQ	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
Eritrea	ER	E. Africa Standard Time	(UTC+03:00)	Nairobi
Estonia	EE	FLE Standard Time	(UTC+02:00)	Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius
Ethiopia	ET	E. Africa Standard Time	(UTC+03:00)	Nairobi
Falkland Islands (Islas Malvinas)	FK	SA Eastern Standard Time	(UTC-03:00)	Cayenne, Fortaleza
Faroe Islands	FO	GMT Standard Time	(UTC)	Dublin, Edinburgh, Lisbon, London
Fiji Islands	FJ	Fiji Standard Time	(UTC+12:00)	Fiji
Finland	FI	FLE Standard Time	(UTC+02:00)	Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius
France	FR	Romance Standard Time	(UTC+01:00)	Brussels, Copenhagen, Madrid, Paris
French Guiana	GF	SA Eastern Standard Time	(UTC-03:00)	Cayenne, Fortaleza
French Polynesia	PF	Hawaiian Standard Time	(UTC-10:00)	Hawaii
French Southern and Antarctic Lands	TF	West Asia Standard Time	(UTC+05:00)	Ashgabat, Tashkent

Gabon	GA	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
Gambia, The	GM	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Georgia	GE	Georgian Standard Time	(UTC+04:00)	Tbilisi
Germany	DE	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Ghana	GH	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Gibraltar	GI	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Greece	GR	GTB Standard Time	(UTC+02:00)	Athens, Bucharest
Greenland	GL	Greenland Standard Time	(UTC-03:00)	Greenland
Grenada	GD	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Guadeloupe	GP	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Guam	GU	West Pacific Standard Time	(UTC+10:00)	Guam, Port Moresby
Guatemala	GT	Central America Standard Time	(UTC-06:00)	Central America
Guernsey	GG	GMT Standard Time	(UTC)	Dublin, Edinburgh, Lisbon, London
Guinea	GN	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Guinea-Bissau	GW	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Guyana	GY	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Haiti	HT	Eastern Standard Time	(UTC-05:00)	Eastern Time (US & Canada)
Heard Island and McDonald Islands	HM	Mauritius Standard Time	(UTC+04:00)	Port Louis

Honduras	HN	Central America Standard Time	(UTC-06:00)	Central America
Hong Kong SAR	HK	China Standard Time	(UTC+08:00)	Beijing, Chongqing, Hong Kong, Urumqi
Hungary	HU	Central Europe Standard Time	(UTC+01:00)	Belgrade, Bratislava, Budapest, Ljubljana, Prague
Iceland	IS	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
India	IN	India Standard Time	(UTC+05:30)	Chennai, Kolkata, Mumbai, New Delhi
Indonesia	ID	SE Asia Standard Time	(UTC+07:00)	Bangkok, Hanoi, Jakarta
Iran	IR	Iran Standard Time	(UTC+03:30)	Tehran
Iraq	IQ	Arabic Standard Time	(UTC+03:00)	Baghdad
Ireland	IE	GMT Standard Time	(UTC)	Dublin, Edinburgh, Lisbon, London
Israel	IL	Israel Standard Time	(UTC+02:00)	Jerusalem
Italy	IT	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Jamaica	JM	SA Pacific Standard Time	(UTC-05:00)	Bogota, Lima, Quito, Rio Branco
Jan Mayen	SJ	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Japan	JP	Tokyo Standard Time	(UTC+09:00)	Osaka, Sapporo, Tokyo
Jersey	JE	GMT Standard Time	(UTC)	Dublin, Edinburgh, Lisbon, London
Jordan	JO	Jordan Standard Time	(UTC+02:00)	Amman
Kazakhstan	KZ	Central Asia Standard Time	(UTC+06:00)	Astana
Kenya	KE	E. Africa Standard Time	(UTC+03:00)	Nairobi

Kiribati	KI	UTC+12	(UTC+12:00)	Coordinated Universal Time+12
Korea	KR	Korea Standard Time	(UTC+09:00)	Seoul
Kosovo	XK	Central European Standard Time	(UTC+01:00)	Sarajevo, Skopje, Warsaw, Zagreb
Kuwait	KW	Arab Standard Time	(UTC+03:00)	Kuwait, Riyadh
Kyrgyzstan	KG	Central Asia Standard Time	(UTC+06:00)	Astana
Laos	LA	SE Asia Standard Time	(UTC+07:00)	Bangkok, Hanoi, Jakarta
Latvia	LV	FLE Standard Time	(UTC+02:00)	Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius
Lebanon	LB	Middle East Standard Time	(UTC+02:00)	Beirut
Lesotho	LS	South Africa Standard Time	(UTC+02:00)	Harare, Pretoria
Liberia	LR	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Libya	LY	E. Europe Standard Time	(UTC+02:00)	E. Europe
Liechtenstein	LI	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Lithuania	LT	FLE Standard Time	(UTC+02:00)	Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius
Luxembourg	LU	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Macao SAR	MO	China Standard Time	(UTC+08:00)	Beijing, Chongqing, Hong Kong, Urumqi
Macedonia, Former Yugoslav Republic of	MK	Central European Standard Time	(UTC+01:00)	Sarajevo, Skopje, Warsaw, Zagreb
Madagascar	MG	E. Africa Standard Time	(UTC+03:00)	Nairobi
Malawi	MW	South Africa Standard Time	(UTC+02:00)	Harare, Pretoria

Malaysia	MY	Singapore Standard Time	(UTC+08:00)	Kuala Lumpur, Singapore
Maldives	MV	West Asia Standard Time	(UTC+05:00)	Ashgabat, Tashkent
Mali	ML	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Malta	MT	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Man, Isle of	IM	GMT Standard Time	(UTC)	Dublin, Edinburgh, Lisbon, London
Marshall Islands	MH	UTC+12	(UTC+12:00)	Coordinated Universal Time+12
Martinique	MQ	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Mauritania	MR	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Mauritius	MU	Mauritius Standard Time	(UTC+04:00)	Port Louis
Mayotte	YT	E. Africa Standard Time	(UTC+03:00)	Nairobi
Mexico	MX	Central Standard Time (Mexico)	(UTC-06:00)	Guadalajara, Mexico City, Monterrey
Micronesia	FM	West Pacific Standard Time	(UTC+10:00)	Guam, Port Moresby
Moldova	MD	GTB Standard Time	(UTC+02:00)	Athens, Bucharest
Monaco	MC	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Mongolia	MN	Ulaanbaatar Standard Time	(UTC+08:00)	Ulaanbaatar
Montenegro	ME	Central European Standard Time	(UTC+01:00)	Sarajevo, Skopje, Warsaw, Zagreb
Montserrat	MS	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Morocco	MA	Morocco Standard Time	(UTC)	Casablanca

Mozambique	MZ	South Africa Standard Time	(UTC+02:00)	Harare, Pretoria
Myanmar	MM	Myanmar Standard Time	(UTC+06:30)	Yangon (Rangoon)
Namibia	NA	Namibia Standard Time	(UTC+01:00)	Windhoek
Nauru	NR	UTC+12	(UTC+12:00)	Coordinated Universal Time+12
Nepal	NP	Nepal Standard Time	(UTC+05:45)	Kathmandu
Netherlands	NL	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
New Caledonia	NC	Central Pacific Standard Time	(UTC+11:00)	Solomon Is., New Caledonia
New Zealand	NZ	New Zealand Standard Time	(UTC+12:00)	Auckland, Wellington
Nicaragua	NI	Central America Standard Time	(UTC-06:00)	Central America
Niger	NE	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
Nigeria	NG	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
Niue	NU	UTC-11	(UTC-11:00)	Coordinated Universal Time-11
Norfolk Island	NF	Central Pacific Standard Time	(UTC+11:00)	Solomon Is., New Caledonia
North Korea	KP	Korea Standard Time	(UTC+09:00)	Seoul
Northern Mariana Islands	MP	West Pacific Standard Time	(UTC+10:00)	Guam, Port Moresby
Norway	NO	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Oman	OM	Arabian Standard Time	(UTC+04:00)	Abu Dhabi, Muscat
Pakistan	PK	Pakistan Standard Time	(UTC+05:00)	Islamabad, Karachi

Palau	PW	Tokyo Standard Time	(UTC+09:00)	Osaka, Sapporo, Tokyo
Palestinian Authority	PS	Egypt Standard Time	(UTC+02:00)	Cairo
Panama	PA	SA Pacific Standard Time	(UTC-05:00)	Bogota, Lima, Quito, Rio Branco
Papua New Guinea	PG	West Pacific Standard Time	(UTC+10:00)	Guam, Port Moresby
Paraguay	PY	Paraguay Standard Time	(UTC-04:00)	Asuncion
Peru	PE	SA Pacific Standard Time	(UTC-05:00)	Bogota, Lima, Quito, Rio Branco
Philippines	PH	Singapore Standard Time	(UTC+08:00)	Kuala Lumpur, Singapore
Pitcairn Islands	PN	Pacific Standard Time	(UTC-08:00)	Pacific Time (US & Canada)
Poland	PL	Central European Standard Time	(UTC+01:00)	Sarajevo, Skopje, Warsaw, Zagreb
Portugal	PT	GMT Standard Time	(UTC)	Dublin, Edinburgh, Lisbon, London
Puerto Rico	PR	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Qatar	QA	Arab Standard Time	(UTC+03:00)	Kuwait, Riyadh
Reunion	RE	Mauritius Standard Time	(UTC+04:00)	Port Louis
Romania	RO	GTB Standard Time	(UTC+02:00)	Athens, Bucharest
Russia	RU	Russian Standard Time	(UTC+03:00)	Moscow, St. Petersburg, Volgograd (RTZ 2)
Rwanda	RW	South Africa Standard Time	(UTC+02:00)	Harare, Pretoria
Saint Barthélemy	BL	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Saint Helena, Ascension and Tristan da Cunha	SH	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik

Saint Kitts and Nevis	KN	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Saint Lucia	LC	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Saint Martin (French part)	MF	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Saint Pierre and Miquelon	PM	Greenland Standard Time	(UTC-03:00)	Greenland
Saint Vincent and the Grenadines	VC	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Samoa	WS	Samoa Standard Time	(UTC+13:00)	Samoa
San Marino	SM	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
São Tomé and Príncipe	ST	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Saudi Arabia	SA	Arab Standard Time	(UTC+03:00)	Kuwait, Riyadh
Senegal	SN	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Serbia	RS	Central Europe Standard Time	(UTC+01:00)	Belgrade, Bratislava, Budapest, Ljubljana, Prague
Seychelles	SC	Mauritius Standard Time	(UTC+04:00)	Port Louis
Sierra Leone	SL	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Singapore	SG	Singapore Standard Time	(UTC+08:00)	Kuala Lumpur, Singapore
Sint Maarten (Dutch part)	SX	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Slovakia	SK	Central Europe Standard Time	(UTC+01:00)	Belgrade, Bratislava, Budapest, Ljubljana, Prague
Slovenia	SI	Central Europe Standard Time	(UTC+01:00)	Belgrade, Bratislava, Budapest, Ljubljana, Prague

Solomon Islands	SB	Central Pacific Standard Time	(UTC+11:00)	Solomon Is., New Caledonia
Somalia	SO	E. Africa Standard Time	(UTC+03:00)	Nairobi
South Africa	ZA	South Africa Standard Time	(UTC+02:00)	Harare, Pretoria
South Georgia and the South Sandwich Islands	GS	UTC-02	(UTC-02:00)	Coordinated Universal Time-02
South Sudan	SS	E. Africa Standard Time	(UTC+03:00)	Nairobi
Spain	ES	Romance Standard Time	(UTC+01:00)	Brussels, Copenhagen, Madrid, Paris
Sri Lanka	LK	Sri Lanka Standard Time	(UTC+05:30)	Sri Jayawardenepura
Sudan	SD	E. Africa Standard Time	(UTC+03:00)	Nairobi
Suriname	SR	SA Eastern Standard Time	(UTC-03:00)	Cayenne, Fortaleza
Svalbard	SJ	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Swaziland	SZ	South Africa Standard Time	(UTC+02:00)	Harare, Pretoria
Sweden	SE	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Switzerland	CH	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Syria	SY	Syria Standard Time	(UTC+02:00)	Damascus
Taiwan	TW	Taipei Standard Time	(UTC+08:00)	Taipei
Tajikistan	TJ	West Asia Standard Time	(UTC+05:00)	Ashgabat, Tashkent
Tanzania	TZ	E. Africa Standard Time	(UTC+03:00)	Nairobi

Thailand	TH	SE Asia Standard Time	(UTC+07:00)	Bangkok, Hanoi, Jakarta
Togo	TG	Greenwich Standard Time	(UTC)	Monrovia, Reykjavik
Tokelau	TK	Tonga Standard Time	(UTC+13:00)	Nuku'alofa
Tonga	TO	Tonga Standard Time	(UTC+13:00)	Nuku'alofa
Trinidad and Tobago	TT	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Tunisia	TN	W. Central Africa Standard Time	(UTC+01:00)	West Central Africa
Turkey	TR	Turkey Standard Time	(UTC+02:00)	Istanbul
Turkmenistan	TM	West Asia Standard Time	(UTC+05:00)	Ashgabat, Tashkent
Turks and Caicos Islands	TC	Eastern Standard Time	(UTC-05:00)	Eastern Time (US & Canada)
Tuvalu	TV	UTC+12	(UTC+12:00)	Coordinated Universal Time+12
U.S. Minor Outlying Islands	UM	UTC-11	(UTC-11:00)	Coordinated Universal Time-11
Uganda	UG	E. Africa Standard Time	(UTC+03:00)	Nairobi
Ukraine	UA	FLE Standard Time	(UTC+02:00)	Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius
United Arab Emirates	AE	Arabian Standard Time	(UTC+04:00)	Abu Dhabi, Muscat
United Kingdom	GB	GMT Standard Time	(UTC)	Dublin, Edinburgh, Lisbon, London
United States	US	Pacific Standard Time	(UTC-08:00)	Pacific Time (US & Canada)
Uruguay	UY	Montevideo Standard Time	(UTC-03:00)	Montevideo
Uzbekistan	UZ	West Asia Standard Time	(UTC+05:00)	Ashgabat, Tashkent
Vanuatu	VU	Central Pacific Standard Time	(UTC+11:00)	Solomon Is., New Caledonia

Vatican City	VA	W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Vietnam	VN	SE Asia Standard Time	(UTC+07:00)	Bangkok, Hanoi, Jakarta
Virgin Islands, U.S.	VI	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Virgin Islands, British	VG	SA Western Standard Time	(UTC-04:00)	Georgetown, La Paz, Manaus, San Juan
Wallis and Futuna	WF	UTC+12	(UTC+12:00)	Coordinated Universal Time+12
Yemen	YE	Arab Standard Time	(UTC+03:00)	Kuwait, Riyadh
Zambia	ZM	South Africa Standard Time	(UTC+02:00)	Harare, Pretoria
Zimbabwe	ZW	South Africa Standard Time	(UTC+02:00)	Harare, Pretoria

# Keyboard Identifiers and Input Method Editors for Windows

1/18/2019 • 3 minutes to read

Use keyboard identifiers and Input Method Editors (IMEs) identify the keyboard type.

## Keyboard identifiers

The following table lists keyboard identifiers that are available for Windows. You can also install support for additional keyboard types. The valid keyboards that can be configured for your device are listed in the registry key: **HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet001\Control\Keyboard Layouts**

KEYBOARD	KEYBOARD IDENTIFIER (HEXADECIMAL)
Albanian	0x00000041c
Arabic (101)	0x000000401
Arabic (102)	0x00010401
Arabic (102) AZERTY	0x00020401
Armenian Eastern	0x00000042b
Armenian Phonetic	0x0002042b
Armenian Typewriter	0x0003042b
Armenian Western	0x0001042b
Assamese - Inscript	0x00000044d
Azerbaijani (Standard)	0x0001042c
Azerbaijani Cyrillic	0x00000082c
Azerbaijani Latin	0x00000042c
Bashkir	0x00000046d
Belarusian	0x000000423
Belgian (Comma)	0x0001080c
Belgian (Period)	0x000000813
Belgian French	0x00000080c

KEYBOARD	KEYBOARD IDENTIFIER (HEXADECIMAL)
Bangla (Bangladesh)	0x00000445
Bangla (India)	0x00020445
Bangla (India - Legacy)	0x00010445
Bosnian (Cyrillic)	0x0000201a
Buginese	0x000b0c00
Bulgarian	0x0030402
Bulgarian (Latin)	0x00010402
Bulgarian (phonetic layout)	0x00020402
Bulgarian (phonetic traditional)	0x00040402
Bulgarian (Typewriter)	0x00000402
Canadian French	0x00001009
Canadian French (Legacy)	0x00000c0c
Canadian Multilingual Standard	0x00011009
Central Atlas Tamazight	0x0000085f
Central Kurdish	0x00000429
Cherokee Nation	0x0000045c
Cherokee Nation Phonetic	0x0001045c
Chinese (Simplified) - US Keyboard	0x00000804
Chinese (Traditional) - US Keyboard	0x00000404
Chinese (Traditional, Hong Kong S.A.R.)	0x00000c04
Chinese (Traditional Macao S.A.R.) US Keyboard	0x00001404
Chinese (Simplified, Singapore) - US keyboard	0x00001004
Croatian	0x0000041a
Czech	0x00000405
Czech (QWERTY)	0x00010405

KEYBOARD	KEYBOARD IDENTIFIER (HEXADECIMAL)
Czech Programmers	0x00020405
Danish	0x00000406
Devanagari-INSCRIPT	0x00000439
Divehi Phonetic	0x00000465
Divehi Typewriter	0x00010465
Dutch	0x00000413
Dzongkha	0x00000C51
Estonian	0x00000425
Faeroese	0x00000438
Finnish	0x0000040b
Finnish with Sami	0x0001083b
French	0x0000040c
Futhark	0x00120c00
Georgian	0x00000437
Georgian (Ergonomic)	0x00020437
Georgian (QWERTY)	0x00010437
Georgian Ministry of Education and Science Schools	0x00030437
Georgian (Old Alphabets)	0x00040437
German	0x00000407
German (IBM)	0x00010407
Gothic	0x000c0c00
Greek	0x00000408
Greek (220)	0x00010408
Greek (220) Latin	0x00030408
Greek (319)	0x00020408

KEYBOARD	KEYBOARD IDENTIFIER (HEXADECIMAL)
Greek (319) Latin	0x00040408
Greek Latin	0x00050408
Greek Polytonic	0x00060408
Greenlandic	0x0000046f
Guarani	0x00000474
Gujarati	0x00000447
Hausa	0x00000468
Hebrew	0x0000040d
Hindi Traditional	0x00010439
Hungarian	0x0000040e
Hungarian 101-key	0x0001040e
Icelandic	0x0000040f
Igbo	0x00000470
India	0x000004009
Inuktitut - Latin	0x0000085d
Inuktitut - Naqittaut	0x0001045d
Irish	0x00001809
Italian	0x00000410
Italian (142)	0x00010410
Japanese	0x00000411
Javanese	0x00110c00
Kannada	0x0000044b
Kazakh	0x0000043f
Khmer	0x00000453
Khmer (NIDA)	0x00010453

KEYBOARD	KEYBOARD IDENTIFIER (HEXADECIMAL)
Korean	0x00000412
Kyrgyz Cyrillic	0x00000440
Lao	0x00000454
Latin American	0x0000080a
Latvian (Standard)	0x00020426
Latvian (Legacy)	0x00010426
Lisu (Basic)	0x00070c00
Lisu (Standard)	0x00080c00
Lithuanian	0x00010427
Lithuanian IBM	0x00000427
Lithuanian Standard	0x00020427
Luxembourgish	0x0000046e
Macedonia (FYROM)	0x0000042f
Macedonia (FYROM) - Standard	0x0001042f
Malayalam	0x0000044c
Maltese 47-Key	0x0000043a
Maltese 48-key	0x0001043a
Maori	0x00000481
Marathi	0x0000044e
Mongolian (Mongolian Script - Legacy)	0x00000850
Mongolian (Mongolian Script - Standard)	0x00020850
Mongolian Cyrillic	0x00000450
Myanmar	0x00010c00
N'ko	0x00090c00
Nepali	0x00000461

KEYBOARD	KEYBOARD IDENTIFIER (HEXADECIMAL)
New Tai Lue	0x00020c00
Norwegian	0x00000414
Norwegian with Sami	0x0000043b
Odia	0x00000448
Ol Chiki	0x000d0c00
Old Italic	0x000f0c00
Osmanya	0x000e0c00
Pashto (Afghanistan)	0x00000463
Persian	0x00000429
Persian (Standard)	0x00050429
Phags-pa	0x000a0c00
Polish (214)	0x00010415
Polish (Programmers)	0x00000415
Portuguese	0x00000816
Portuguese (Brazilian ABNT)	0x00000416
Portuguese (Brazilian ABNT2)	0x00010416
Punjabi	0x00000446
Romanian (Legacy)	0x00000418
Romanian (Programmers)	0x00020418
Romanian (Standard)	0x00010418
Russian	0x00000419
Russian - Mnemonic	0x00020419
Russian (Typewriter)	0x00010419
Sakha	0x00000485
Sami Extended Finland-Sweden	0x0002083b

KEYBOARD	KEYBOARD IDENTIFIER (HEXADECIMAL)
Sami Extended Norway	0x0001043b
Scottish Gaelic	0x00011809
Serbian (Cyrillic)	0x00000c1a
Serbian (Latin)	0x0000081a
Sesotho sa Leboa	0x0000046c
Setswana	0x00000432
Sinhala	0x0000045b
Sinhala - wij 9	0x0001045b
Slovak	0x0000041b
Slovak (QWERTY)	0x0001041b
Slovenian	0x00000424
Sora	0x00100c00
Sorbian Extended	0x0001042e
Sorbian Standard	0x0002042e
Sorbian Standard (Legacy)	0x0000042e
Spanish	0x0000040a
Spanish Variation	0x0001040a
Swedish	0x0000041d
Swedish with Sami	0x0000083b
Swiss French	0x0000100c
Swiss German	0x00000807
Syriac	0x0000045a
Syriac Phonetic	0x0001045a
Tai Le	0x00030c00
Tajik	0x00000428

KEYBOARD	KEYBOARD IDENTIFIER (HEXADECIMAL)
Tamil	0x00000449
Tatar	0x00010444
Tatar (Legacy)	0x00000444
Telugu	0x0000044a
Thai Kedmanee	0x0000041e
Thai Kedmanee (non-ShiftLock)	0x0002041e
Thai Pattachote	0x0001041e
Thai Pattachote (non-ShiftLock)	0x0003041e
Tibetan (PRC - Standard)	0x00010451
Tibetan (PRC - Legacy)	0x00000451
Tifinagh (Basic)	0x00050c00
Tifinagh (Full)	0x00060c00
Turkish F	0x0001041f
Turkish Q	0x0000041f
Turkmen	0x00000442
Uyghur	0x00010408
Uyghur (Legacy)	0x00000480
Ukrainian	0x00000422
Ukrainian (Enhanced)	0x00020422
United Kingdom	0x00000809
United Kingdom Extended	0x00000452
United States - Dvorak	0x00010409
United States - International	0x00020409
United States-Dvorak for left hand	0x00030409
United States-Dvorak for right hand	0x00040409

KEYBOARD	KEYBOARD IDENTIFIER (HEXADECIMAL)
United States - English	0x00000409
Urdu	0x00000420
Uyghur	0x00010480
Uzbek Cyrillic	0x00000843
Vietnamese	0x0000042a
Wolof	0x00000488
Yakut	0x00000485
Yoruba	0x0000046a

## Input Method Editors

LANGUAGE/REGION	INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)
Amharic - (Ethiopia)	am-ET: Amharic Input Method (045e:{E429B25A-E5D3-4D1F-9BE3-0C608477E3A1}{8F96574E-C86C-4bd6-9666-3F7327D4CBE8})
Chinese (PRC)	zh-CN: Microsoft Pinyin - Simple Fast (0804:{81D4E9C9-1D3B-41BC-9E6C-4B40BF79E35E}{FA550B04-5AD7-411f-A5AC-CA038EC515D7})
Chinese (Taiwan)	zh-TW: Chinese (Traditional) - New Phonetic (0404:{B115690A-EA02-48D5-A231-E3578D2FDF80}{B2F9C502-1742-11D4-9790-0080C882687E})
Chinese (Traditional DaYi)	0404:{E429B25A-E5D3-4D1F-9BE3-0C608477E3A1}{037B2C25-480C-4D7F-B027-D6CA6B69788A}
Chinese (Wubi)	0804:{6a498709-e00b-4c45-a018-8f9e4081ae40}{82590C13-F4DD-44f4-BA1D-8667246FDF8E}
Chinese (Yi)	ii-CN: Yi Input Method(0478:{E429B25A-E5D3-4D1F-9BE3-0C608477E3A1}{409C8376-007B-4357-AE8E-26316EE3FB0D})
Japanese (Japan)	ja-JP: Microsoft IME (0411:{03B5835F-F03C-411B-9CE2-AA23E1171E36}{A76C93D9-5523-4E90-AAFA-4DB112F9AC76})
Korean (Hangul)	ko-KR: Microsoft IME (0412:{A028AE76-01B1-46C2-99C4-ACD9858AE02F}{B5FE1F02-D5F2-4445-9C03-C568F23C99A1})
Korean (Old Hangul)	0412:{a1e2b86b-924a-4d43-80f6-8a820df7190f}{b60af051-257a-46bc-b9d3-84dad819bafb}

LANGUAGE/REGION	INPUT PROFILE (LANGUAGE AND KEYBOARD PAIR)
Tigrinya (Ethiopia)	ti-ET: Tigrinya Input Method (0473:{E429B25A-E5D3-4D1F-9BE3-0C608477E3A1}{3CAB88B7-CC3E-46A6-9765-B772AD7761FF})

## Related topics

[Available Language Packs for Windows](#)

[Default Input Profiles \(Input Locales\) in Windows](#)

# Optimize

1/18/2019 • 2 minutes to read

You can optimize a Windows installation in several ways. This section covers how to reduce the size of a Windows image, a Windows installation, and how to create power plans.

## In this section

TOPIC	DESCRIPTION
<a href="#">Compact OS, single-instancing, and image optimization</a>	Learn how to reduce the size of your Windows image
<a href="#">Manage the Component Store</a>	Reduce the size of a Windows installation
<a href="#">Battery Life</a>	Configure power plans to improve battery life or performance

# Compact OS, single-instancing, and image optimization

1/18/2019 • 8 minutes to read

Windows 10 includes tools to help you use less drive space. You can now compress the files for the entire operating system, including your preloaded desktop applications. Compact OS lets you run the operating system from compressed files (similar to WIMBoot in Windows 8.1 Update 1), and single-instancing helps you run your pre-loaded Windows desktop applications in compressed files. The new processes helps maintain a small footprint over time by using individual files, rather than combining them in a WIM file.

Here's some ways to shrink the image, optimize the image, and some considerations when deploying to low-cost devices.

## Deployment tools that help save space

### Compact OS

Compact OS installs the operating system files as compressed files. Compact OS is supported on both UEFI-based and BIOS-based devices. See the [size comparison table](#) below.

Unlike WIMBoot, because the files are no longer combined into a single WIM file, Windows update can replace or remove individual files as needed to help maintain the drive footprint size over time.

#### To deploy Compact OS using a WIM file

1. Boot your destination device with the Windows 10 version of Windows PE. (To use a previous version of Windows PE, make sure you use the Windows 10 version of DISM. To learn more, see [Copy DISM to Another Computer](#).)
2. Create a pagefile equal to 256 MB.

```
Wpeutil createpagefile C:\pagefile /size=256
```

Where "C" is the Windows partition.

3. Format and prepare the partitions, and then apply the image to a partition using the DISM /Apply-Image /Compact option:

```
DISM /Apply-Image /ImageFile:install.wim /Index:1 /ApplyDir:D:\ /compact
```

This is usually done by running a deployment script. To learn more, see [Apply Images Using DISM](#).

**Note:** If you're applying an image in compact mode and using the /ScratchDir option, make sure your ScratchDir folder is not on a FAT32-formatted partition. Using a FAT32 partition could result in unexpected reboots during OOBE.

#### To deploy Compact OS from Windows Setup

- Use an unattend.xml file with the setting: Microsoft-Windows-Setup\ImageInstall\OSImage\Compact.

#### To deploy Compact OS with a USB bootable drive

For Windows 10, Version 1607 and earlier

1. On your technician PC, open Windows ICD and create your project.

2. Plug in a USB flash drive and note the drive letter (example: D:).
3. Click **Create > Production Media > WIM > Enable OS File Compression: Yes > Next > USB Bootable drive** > drive letter (D:) > **Next > Build**.
4. Boot the destination PC using the USB flash drive. Windows installs automatically.

**Note** When running Windows Imaging and Configuration Designer (ICD) on a PC running a previous version of Windows, such as Windows 8.1, you'll need to install the [Windows Assessment and Deployment Kit \(ADK\)](#) with both the Windows ICD and **Deployment Tools** features. This installs the latest versions of the drivers required by DISM (wimmount.sys and adkwof.sys) used to create Compact OS images.

#### To deploy Compact OS from an FFU image

For Windows 10, Version 1607 and earlier

1. To deploy an FFU image as compressed, the original FFU image must be created as a compressed image.

From Windows ICD, click **Create > Production Media > FFU > Enable OS File Compression: Yes > name the file, for example, D:\flash.ffu > Build**.

2. You can deploy the FFU image directly to a drive from Windows ICD or from Windows Preinstallation Environment (WinPE). To learn more, see [Deploy Windows using Full Flash Update \(FFU\)](#).

**Note** When running Windows Imaging and Configuration Designer (ICD) on a PC running a previous version of Windows, such as Windows 8.1, you'll need to install the [Windows Assessment and Deployment Kit \(ADK\)](#) with both the Windows ICD and **Deployment Tools** features. This installs the latest versions of the drivers required by DISM (wimmount.sys and adkwof.sys) used to create Compact OS images.

#### Command-line support

You can query whether the operating system is running Compact OS, and change it at any time, using the [Compact.exe](#) command.

#### From Windows PE, determine if the OS is compacted:

```
Compact.exe /CompactOS:Query /WinDir:E:\Windows
```

Where E:\Windows is the folder where Windows has been installed.

#### From an online installation, change from non-compacted to compacted OS:

```
Compact.exe /CompactOS:always
```

#### Single-instancing of provisioning packages

For Windows 10, when you add new Windows desktop applications to a device, you'll capture these changes into a compressed provisioning package for use by the automatic recovery tools. Rather than maintaining both the original files and the provisioning package, you can use DISM to remove the original files, and run from directly from the compressed provisioning package instead. This is known as single-instancing the image. See the [size comparison table](#) below.

While single-instancing is supported on both solid-state drives and rotational drives, for performance reasons, we recommend that single-instancing is only used on devices with solid-state drives.

Example:

```
DISM /Apply-CustomDataImage /CustomDataImage:C:\Recovery\Customizations\USMT.ppkg /ImagePath:C:\SingleInstance
```

where C is the drive letter of the Windows partition.

**Warning** Do not put quotes with the /ImagePath:C:\ option.

You can determine whether a provisioning package (.ppkg) is single-instanced by using fsutil.exe:

```
fsutil.exe wim enumwims C:
```

where C is the drive that contains the provisioning package. Any single-instanced provisioning package on the drive will be listed in the command output. If there are none, the command will return "Error: The system cannot find the file specified."

## Image optimization

After applying updates to a Windows image, cleanup the image and then export it to a new file:

```
md c:\mount\Windows
md C:\mount\temp

Dism /Mount-Image /ImageFile:"C:\Images\install.wim" /Index:1 /MountDir:C:\mount\Windows

Dism /Cleanup-Image /Image=C:\mount\Windows /StartComponentCleanup /ResetBase /ScratchDir:C:\mount\temp

Dism /Unmount-Image /MountDir:C:\mount\Windows /Commit

Dism /Export-Image /SourceImageFile:C:\Images\install.wim /SourceIndex:1
/DestinationImageFile:C:\Images\install_cleaned.wim
```

where C:\Images\install.wim is a Windows image file that you want to update. Beginning with Windows 10, version 1607, you can optionally specify the /Defer parameter with /ResetBase to defer any long-running cleanup operations to the next automatic maintenance, but we highly recommend that **only** use /Defer as an option in the factory where DISM /ResetBase requires more than 30 minutes to complete.

## Size requirements and considerations

You'll still need to meet minimum size requirements for the hard drive, RAM, application resource usage, and data storage.

### Hard Drive

Windows 10 requires a minimum of 16 gigabytes (GB) of space on 32-bit devices, and 20 GB on 64-bit devices.

Although some configurations of Windows may appear to fit on smaller drives when Windows is first installed, 8 GB SSDs are not large enough. Even if a user pairs an 8 GB hard drive with a second drive that is 4 GB or larger for application and data file storage, 8 GB hard drives do not allow for the increase in the Windows memory footprint that is expected to occur as users work on their computer.

Some of the primary reasons for the increase over time in the memory footprint include the following:

- **Servicing.** Hard disk space must be reserved for software patches to the operating system and for service pack releases.
- **System Restore Points.** Windows automatically generate restore points. The amount of space that is required by default is relative to the size of the hard drive. For more information about restore points, see the [Restore Points](#) topic on MSDN. **Note** Users can adjust the amount of space used on the computer for System Restore by using the **System Protection** user interface in the **System Properties** dialog box (Sysdm.cpl). Users can also use system image backups that are stored on an external hard disk to restore a system.

- **Logs and Caches.** The operating system stores files such as event logs and error logs on the drive.

## RAM, Pagefile.sys, and Hiberfil.sys

The Pagefile.sys and Hiberfil.sys files increase in size in direct proportion to the amount of RAM on the computer. Windows installations on 16 GB drives have a smaller memory footprint when the computer is limited to 1 GB of RAM. An increase of RAM to a size that is greater than 1 GB will result in increased size of the system files and less space on the hard drive for other applications and files. Increasing the size of the hard drive, however, does not affect the size of these system files. Learn more about [On/Off Transition Performance](#)

To save space on the drive, you can remove or reduce the size of the hiberfil.sys. See the [size comparison table](#) below. To learn more, see [Lab 7: Change settings, enter product keys, and run scripts with an answer file \(unattend.xml\)](#).

- `powercfg /h /type reduced` : Reduces the file by 30%
- `powercfg /h off` : Removes the file.

## Language packs and features on demand

Installed language packs (LPs) can take more space than just the size of the LP itself. When you preinstall FODs and UWP apps on a Windows installation that contains multiple LPs, resource files based on preinstalled LPs are also installed. When unused languages are automatically removed after OOBE, corresponding UWP and feature on demand (FOD) resource files are not removed. Preinstalling fewer LPs saves disk space by limiting the number of resource files that remain on a system after removing unused language packs.

Features on demand are distributed in compressed CAB files so the size of an installed FOD is larger than the size of the original CAB. You can use `/Get-CapabilityInfo` in DISM to view an FOD's download and install sizes. See [Features on demand](#) for how to get information about FODs.

## Applications

Software applications that are installed on the computer may require additional space for caches, logs, and updates. Disk space must also be available on the drive to account for temporary increases in resource usage during installation of applications, patches, and updates.

## User Data

On computers that support removable media such as an SD card or USB flash drive, users can easily expand personal data file storage for user documents by using this removable media. However, we recommend that users reserve some space on the hard drive for these types of files.

## Size comparisons

The table below shows the additional space saved by using compact OS, Single instancing, and reducing or turning Off Hiberfile on 2GB (x86 processor architecture) and 4GB (x64 processor architecture), on Windows 10, version 1607:

IMAGE	WINDOWS 10 HOME X86, 2GB MEMORY	WINDOWS 10 HOME X64, 4GB MEMORY
Base Footprint	11.68GB	15.06GB
Compact OS, with no single instancing	8.85GB (>2.75GB savings)	11.3GB (>3.7GB)
Compact OS, single instanced	7.66GB (>4GB)	10.09GB (>4.75GB)
Hiberfile off, no compact OS	10.87GB (>825MB)	13.48GB (>1.5GB)
Hiberfile reduced, no compact OS	11.27GB (>400MB)	14.15GB (>930MB)

## Related topics

[Windows Imaging and Configuration Designer](#)

[Capture and Apply Windows, System, and Recovery Partitions](#)

[DISM Image Management Command-Line Options](#)

# Manage the Component Store

1/24/2019 • 3 minutes to read

"Why is WinSxS so large?" has been asked by many Windows users. While this question has been discussed in blog posts, this topic goes into a little more details about the concepts behind the component store (specifically the WinSxS folder) and then provides links to topics that highlight ways to better manage the size of the WinSxS folder.

The short answer is that the WinSxS folder isn't as large as it may appear at first glance because size calculations can include Windows binaries located elsewhere which makes the WinSxS folder seem larger than it really is.

## The Windows component store and WinSxS folder

The WinSxS folder is located in the Windows folder, for example `c:\Windows\WinSxS`. It's the location for Windows Component Store files. The Windows Component Store is used to support the functions needed for the customization and updating of Windows. Here are some examples of how the Windows Component Store files are used:

- Using Windows Update to install new component versions. This keeps systems secure and up-to-date.
- Enabling or disabling Windows features.
- Adding roles or features using Server Manager.
- Moving systems between different Windows Editions.
- System recovery from corruption or boot failures
- Uninstalling problematic updates
- Running programs using side-by-side assemblies

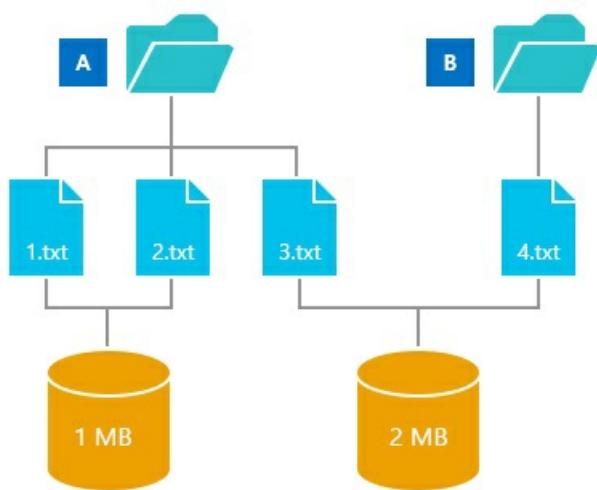
The Windows Component Store was first introduced in Windows XP to support side by side assemblies. Beginning in Windows Vista, the component store was enhanced to track and service all of the components that make up the operating system. Those different operating system components track objects such as files, directories, registry keys, and services. Specific versions of components are then collected together into packages. Packages are used by Windows Update and DISM to update Windows. The components and packages used in a Windows installation are processed by the Windows Component Store. Determining the size of the Windows Component Store is complicated by the fact that many of the files are used by Windows from directories outside the Windows Component Store using a technique known as *hard linking*. In such cases, the files from a component version appear both inside and outside the Windows Component Store. By using *hard linking* Windows is able to appear to keep multiple copies of the same file without actually taking the added space for multiple copies.

## Hard links

A hard link is a file system object which allows two files to refer to the same location on disk. This means that more than one file can refer to the same data and changes to that data in one file are reflected in the other files. This complicates notions of directory size as can be seen using the following example:

1. Directory A has three files: 1.txt, 2.txt, and 3.txt
2. Directory B has one file: 4.txt

3. Files 1.txt and 2.txt are hard linked together and contain 1MB of data.
4. Files 3.txt and 4.txt are also hard linked together and contain 2MB of data.



In this example, you can see that the hard links enable multiple files to refer to the same set of data.

Now what is the size of directory A?

The answer depends on what you plan to do with directory A:

1. If you read the files in the directory A then the size of all the files that are read is the sum of each file size. In this example, that would be 4 MB.
2. If you copy all the files from directory A to a new location, then the amount of data copied is the sum of all data hard linked from the files. In this example, that would be 3 MB.
3. If you are trying to free up space by deleting the directory A, you will only see a reduction in size for the files that are hard linked only by directory A. In this example, this amounts to a savings of 1 MB.

Back to the question of how much space is used by the Windows Component Store, and specifically the WinSxS folder. The third answer in the directory A example, most closely matches how much extra space is used. Files hard linked to the rest of the system are required for system operations, so they should not be counted, and files hard linked to multiple locations within the component store should only have the size stored on disk counted.

## Managing the Windows Component Store

You can use new features in Windows 8.1 and Windows Server 2012 R2 to manage the Windows Component Store:

[Determine the Actual Size of the WinSxS Folder](#)

[Clean Up the WinSxS Folder](#)

[Reduce the Size of the Component Store in an Offline Windows Image](#)

## Related topics

[Where Did My Space Go? \(blog post\)](#)

[More on hard links](#)

[NTFS Metafiles blog post](#)

[How to create and manipulate NTFS junction points](#)

# Determine the Actual Size of the WinSxS Folder

1/24/2019 • 4 minutes to read

Why is the WinSxS folder so big? The short answer to this commonly asked question is that the component store (WinSxS folder) contains all the components that make-up Windows to allow you operate your system. These components are kept to rollback any problematic change or to repair a file that becomes corrupted. For more information about the component store, see [Manage the Component Store](#). For information on how to delete files in the WinSxS folder, see [Clean Up the WinSxS Folder](#).

For operating system files, it can appear that more than one copy of the same version of a file is stored in more than one place on the operating system, but there's usually only one real copy of the file. The rest of the copies are just "projected" by hard linking from the component store. A *hard link* is a file system object that lets two files refer to the same location on disk. Some tools, such as the File Explorer, determine the size of directories without taking into account that the contained files might be hard linked. This might lead you to think that the WinSxS folder takes up more disk space than it really does.

## WARNING

Some important system files are located only in the WinSxS folder. Deleting files from the WinSxS folder or deleting the entire WinSxS folder might severely damage your system, so that your PC might not boot, and make it impossible to update.

A new option has been added to the DISM tool for Windows 8.1 to help determine how much disk space the WinSxS folder really uses.

## Analyze the size of the component store (WinSxS folder)

- Open an elevated command window and type:

```
Dism.exe /Online /Cleanup-Image /AnalyzeComponentStore
```

## NOTE

The **/AnalyzeComponentStore** option isn't recognized on Windows 8 and earlier.

The information returned is:

TITLE	DESCRIPTION
Windows Explorer Reported Size of Component Store	This value the size of the WinSxS folder if computed by Windows Explorer. This value doesn't factor in the use of hard links within the WinSxS folder.
Actual Size of Component Store	This value factors in hard links within the WinSxS folder. It doesn't exclude files that are shared with Windows by using hard links.

TITLE	DESCRIPTION
Shared with Windows	This value provides the size of files that are hard linked so that they appear both in the component store and in other locations (for the normal operation of Windows). This is included in the actual size, but shouldn't be considered part of the component store overhead.
Backups and Disabled Features	<p>This is the size of the components that are being kept to respond to failures in newer components or to provide the option of enabling more functionality. It also includes the size of component store metadata and side-by-side components.</p> <p>This is included in the actual size and is part of the component store overhead.</p>
Cache and Temporary Data	This is the size of files that are used internally by the component store to make component servicing operations faster. This is included in the actual size and is part of the component store overhead.
Date of Last Cleanup	This is the date of the most recently completed component store cleanup.
Number of Reclaimable Packages	This is the number of superseded packages on the system that component cleanup can remove.
Component Store Cleanup Recommended	This is a component store cleanup recommendation. Cleanup is recommended when performing a cleanup process may reduce the size of the component store overhead.

Based on this analysis you can determine the overhead of the WinSxS folder by taking the sum of the backups and disabled features size with the cache and temporary data size.

Example output:

```
C:\>dism /online /cleanup-image /analyzecomponentstore

Deployment Image Servicing and Management tool
Version: 6.3.XXXX.0

Image Version: 6.3.XXXX.0

[=====100.0%=====]

Component Store (WinSxS) information:

Windows Explorer Reported Size of Component Store : 4.98 GB

Actual Size of Component Store : 4.88 GB

Shared with Windows : 4.38 GB
Backups and Disabled Features : 506.90 MB
Cache and Temporary Data : 279.52 KB

Date of Last Cleanup : 2013-06-10 23:32:22

Number of Reclaimable Packages : 0
Component Store Cleanup Recommended : No

The operation completed successfully.
```

In this example, the WinSxS folder appears to be 4.98 GB, but the actual overhead (the sum of the size of backups and disabled features and the size of cache and temporary data) is 507.18 MB.

### Determine if you should clean up the component store (WinSxS folder) based on the analysis results

1. Open an elevated command window and type:

```
Dism.exe /Online /Cleanup-Image /AnalyzeComponentStore
```

2. If cleanup is recommended then follow steps in the related topic, [Clean Up the WinSxS Folder](#).

## Related topics

[Manage the Component Store](#)

[Clean Up the WinSxS Folder](#)

[Where Did My Space Go? \(blog post\)](#)

[Servicing changes in Windows 8.1/Server 2012 R2](#)

[NTFS Metafiles blog post](#)

[How to create and manipulate NTFS junction points](#)

[DISM Operating System Package Servicing Command-Line Options](#)

# Clean Up the WinSxS Folder

1/24/2019 • 4 minutes to read

This topic is about the different ways to reduce the size of the WinSxS folder on a running version of Windows 10.

One commonly asked question is, "Can I delete the WinSxS folder to regain some disk space?" The short answer is no. You can, however, reduce the size of the WinSxS folder using tools built into Windows. For more information about the WinSxS folder, see [Manage the Component Store](#).

Windows 10 and Windows Server 2016 automatically reduce the size of the WinSxS folder by using methods similar to the ones described in this topic, in addition to internal processes, such as uninstalling and deleting packages with components that have been replaced by other components with newer versions. Previous versions of some components are kept on the system for a period of time, allowing you to rollback if necessary. After a period of time, these older components are automatically removed from the installation.

You can also reduce the size of a Windows image using some of the same techniques, as discussed in [Reduce the Size of the Component Store in an Offline Windows Image](#).

To learn about finding the size of your WinSxS folder, see [Determine the actual size of the WinSxS folder](#).

## WARNING

Deleting files from the WinSxS folder or deleting the entire WinSxS folder may severely damage your system so that your PC might not boot and make it impossible to update.

In Windows 10 and Windows Server 2016, you have a number of ways to start the cleanup of the component store, which use a combination of package deletion and component compression to clean up the WinSxS folder:

## Task Scheduler

The **StartComponentCleanup** task was created in Windows 8 to regularly clean up components automatically when the system is not in use. This task is set to run automatically when triggered by the operating system. When run automatically, the task will wait at least 30 days after an updated component has been installed before uninstalling the previous versions of the component.

If you choose to run this task, the task will have a 1 hour timeout and may not completely clean up all files.

### Run the StartComponentCleanup task in Task Scheduler to clean up and compress components

1. If **Task Scheduler** is not open, start the **Task Scheduler**. For more information, see [Start Task Scheduler](#).
2. Expand the console tree and navigate to **Task Scheduler** **Library\Microsoft\Windows\Servicing\StartComponentCleanup**.
3. Under **Selected Item**, click **Run**

```
schtasks.exe /Run /TN "\Microsoft\Windows\Servicing\StartComponentCleanup"
```

## NOTE

The StartComponentCleanup task can also be started from the command line.

## Dism.exe

The **/Cleanup-Image** parameter of **Dism.exe** provides advanced users more options to further reduce the size of the WinSxS folder. For more information, see [DISM Operating System Package Servicing Command-Line Options](#).

### Use the **/StartComponentCleanup** parameter

- Using the **/StartComponentCleanup** parameter of Dism.exe on a running version of Windows 10 gives you similar results to running the **StartComponentCleanup** task in **Task Scheduler**, except previous versions of updated components will be immediately deleted (without a 30 day grace period) and you will not have a 1-hour timeout limitation.

From an elevated command prompt, type the following:

```
Dism.exe /online /Cleanup-Image /StartComponentCleanup
```

### Use the **/ResetBase** switch with the **/StartComponentCleanup** parameter

- Using the **/ResetBase** switch with the **/StartComponentCleanup** parameter of DISM.exe on a running version of Windows 10 removes all superseded versions of every component in the component store.

From an elevated command prompt, type the following:

```
Dism.exe /online /Cleanup-Image /StartComponentCleanup /ResetBase
```

#### WARNING

All existing service packs and updates cannot be uninstalled after this command is completed. This will not block the uninstallation of future service packs or updates.

### Use the **/SPSuperseded** parameter

- To reduce the amount of space used by a Service Pack, use the **/SPSuperseded** parameter of Dism.exe on a running version of Windows 10 to remove any backup components needed for uninstallation of the service pack. A service pack is a collection of cumulative updates for a particular release of Windows.

From an elevated command prompt, type the following:

```
Dism.exe /online /Cleanup-Image /SPSuperseded
```

**Warning** The service pack cannot be uninstalled after this command is completed.

## Disk Cleanup

You can use Disk Cleanup to reduce the number of unnecessary files on your drives, which can help your PC run faster. It can delete temporary files and system files, empty the Recycle Bin, and remove a variety of other items that you might no longer need. The option to cleanup updates helps reduce the size of the component store.

### Run Disk Cleanup to delete system files

- To delete system files run the steps as provided in [Delete files using Disk Cleanup](#).

## Related topics

[Manage the Component Store](#)

[Determine the Actual Size of the WinSxS Folder](#)

[Reduce the Size of the Component Store in an Offline Windows Image](#)

[Uninstall-WindowsFeature](#)

[How to Reduce the Size of the Winsxs directory and Free Up Disk Space on Windows Server 2012 Using Features on Demand](#)

[How to address disk space issues that are caused by a large Windows component store \(WinSxS\) directory](#)

# Reduce the Size of the Component Store in an Offline Windows Image

1/24/2019 • 2 minutes to read

You can use the Deployment Image Servicing and Management (DISM) tool to mount a Windows image from a WIM, VHD, or VHDX file and modify it.

## Analyze and clean up the Component Store (WinSxS folder) in an offline Windows image

To complete the walkthrough, you need the following:

- A computer running Windows 10, Windows 8.1, Windows 8, Windows 7, Windows Server 2016 Technical Preview, Windows Server 2012 R2, Windows Server 2012, or Windows Server 2008 R2 with the Windows 8.1 version of the Windows ADK tools installed on it.
- A .wim, .vhd, or .vhdx file of a Windows 10 or Windows Server 2016 Technical Preview image.

### Analyze the size of the Component Store in an offline Windows image

1. Copy a .wim file, a .vhd, or a .vhdx that contains a Windows image, to the local drive. For example, C:\test\images.
2. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
3. Create a folder for your mounted image. For example, C:\test\offline.
4. Run the **DISM /Get-ImageInfo** command to retrieve the name or index number for the image that you want to update. For example:

```
Dism /Get-ImageInfo /ImageFile:C:\test\images\MyImage.wim
```

5. Mount the Windows image. For example:

```
Dism /Mount-Image /ImageFile:C:\test\images\MyImage.wim /Index:1 /MountDir:C:\test\offline
```

Since WIM files can contain one or more images, you must specify an index or name value. To mount an image from a VHD, you must specify `/Index:1`.

6. Analyze the size of the component store. For example:

```
Dism /Image:C:\test\offline /Cleanup-Image /AnalyzeComponentStore
```

To understand the different values provided in the display, see [Determine the Actual Size of the WinSxS Folder](#).

7. If the component store cleanup was recommended in the displayed report, then you can start cleanup of the image. For example:

```
Dism /Image:C:\test\offline /Cleanup-Image /StartComponentCleanup
```

8. You can reduce the size of the component store further by adding the /ResetBase parameter. For example:

```
Dism /Image:C:\test\offline /Cleanup-Image /StartComponentCleanup /ResetBase
```

Beginning with Windows 10, version 1607, you can specify the /Defer parameter with /Resetbase to defer any long-running cleanup operations to the next automatic maintenance. But we highly recommend you **only** use /Defer as an option in the factory where DISM /Resetbase requires more than 30 minutes to complete.

The maintenance task is scheduled to run weekly, with a two-week deadline. In the first week, the maintenance task will only run during system idle maintenance windows. If it is unable to complete (for example, the computer is turned off when not in use) then the task scheduler runs more often, and the task may run while the system is not idle.

To see the performance effects while the task is running, click Start > Run and type the following command:

```
Schtasks.exe /Run /I /TN \Microsoft\Windows\Servicing\StartComponentCleanup
```

9. Commit the changes and unmounts the image in order to save the changes that you've made. For example:

```
Dism /Unmount-Image /MountDir:C:\test\offline /Commit
```

## Related topics

[Manage the Component Store](#)

[Clean Up the WinSxS Folder](#)

[Determine the Actual Size of the WinSxS Folder](#)

[Where Did My Space Go? \(blog post\)](#)

[NTFS Metafiles blog post](#)

[How to create and manipulate NTFS junction points](#)

[DISM Operating System Package Servicing Command-Line Options](#)

# Take Inventory of an Image or Component Using DISM

1/24/2019 • 23 minutes to read

You can take an inventory of what drivers, packages, and other files and settings are included in a Windows image. To do so, use Deployment Image Servicing and Management (DISM) servicing commands.

You must mount an offline image from a WIM or VHD file before you can take inventory of or service a specific Windows image. For more information, see [Mount and Modify a Windows Image Using DISM](#).

In this section:

[Get Windows Image Information](#)

[Get Windows PE Information](#)

[Get Driver Information](#)

[Get Package and Feature Information](#)

[Get App Package \(.appx\) Servicing Information](#)

[Get International Settings and Languages](#)

[Get Windows Edition Information](#)

[Get Application Patch Information](#)

## Get Windows Image Information

You can use image commands to list the information about a specific Windows image in a (WIM) file or virtual hard disk (VHD) file, about the images contained in a specific WIM or VHD file, and about mounted WIM or VHD files. This information can help you identify mount locations, image names, or verify the architecture of the image that you are mounting.

You can gather information about all of the images in a WIM or VHD file by using the **/Get-ImageInfo** servicing command in DISM. You can also gather information about a specific image in a WIM or VHD file, such as operating system, architecture, and settings, by specifying the name or index number of the image. To specify the image in a VHD file, you must use **/Index:1** .

You can identify the images that are currently mounted on your computer, and you can list information about the mounted image such as read/write permissions, mount location, mounted file path, and mounted image index by using the **/Get-MountedImageInfo** servicing command.

For more information about image commands available in DISM, see [DISM - Deployment Image Servicing and Management Technical Reference for Windows](#).

### To list images that are contained in a WIM or VHD file

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about all of the images in a WIM or VHD file, at the elevated command prompt, type:

```
Dism /Get-ImageInfo /imagefile:C:\test\images\install.wim
```

When used with the **/Index** or **/Name** options, more detailed information about the specified image is displayed. To specify the image in a VHD file, you must use **/Index:1**.

The report that is generated includes the following information.

FIELD	DESCRIPTION	EXAMPLE
Index	The index value of the image in the WIM or VHD file.	1
Name	The Windows edition name of the image in the WIM or VHD file.	Windows 8 Pro
Description	The description of the image in the WIM or VHD file.	Windows 8 Pro
Size	The size of the image.	8,045,951,502 bytes

### To list mounted images

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. At the elevated command prompt, type:

```
Dism /Get-MountedImageInfo
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
Mount Dir	The location where the image is mounted.	C:\Test\Mount
Image File	The full path to the WIM or VHD file.	C:\Test\Images\install.wim
Image Index	The index number of the mounted image that is enclosed in WIM or VHD file.	1
Mounted Read/Write	<b>Yes</b> if the mounted image allows for both read and write access or <b>No</b> if the mounted image allows for read-only access only.	Yes

FIELD	DESCRIPTION	EXAMPLE
Status	<p>The mount status of the image. The possible values include the following:</p> <p><b>OK.</b> The image is mounted. There are no problems.</p> <p><b>Needs Remount.</b> The image must be remounted. This can be caused by rebooting the host system when the image is mounted.</p> <p><b>Invalid.</b>: the image is in an invalid state. You might have to use <b>/Cleanup-Mountpoints</b> on the image.</p>	OK

## Get Windows PE Information

You can mount a Windows Preinstallation Environment (Windows PE) image for servicing in the same way you would any Windows image. There are also Windows PE servicing commands that are specific to a Windows PE image. These commands can be used to list Windows PE settings such as scratchspace, targetpath, and profiling information. For more information about Windows PE servicing commands available in DISM, see [DISM Windows PE Servicing Command-Line Options](#).

### To list all settings in the mounted Windows PE image.

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about all of the Windows PE settings in the mounted Windows PE image, at the elevated command prompt, type:

```
Dism /image:C:\test\offline /Get-PESettings
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
Profiling	Reports whether Windows PE profiling is enabled or disabled.	Disabled
Scratch Space	The amount of writeable space available on the Windows PE system volume when booted in ramdisk mode.	32MB
TargetPath	The path to the root of the Windows PE image at boot time.	X:</p>

## Get Driver Information

The driver-servicing commands can be used to enumerate driver packages in the driver store based on their .inf

files. You can use the **/Get** commands to display basic information about third-party driver packages or all driver packages in the offline image. When you point to an offline image or a running operating system, you can determine what driver packages are in the image, and get information about the drivers.

You can display detailed information about a specific installed .inf file, or one that is not yet installed. Installed drivers in the driver store will be named Oem0.inf, Oem1.inf, and so on.

For more information about driver-servicing commands available in DISM, see [DISM Driver Servicing Command-Line Options](#).

### To list driver packages in the offline image

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. Use one of the following commands to list information about all of the driver packages in a mounted offline Windows image:

```
Dism /image:C:\test\offline /Get-Drivers
```

```
Dism /image:C:\test\offline /Get-Drivers /all
```

For a running operating system, type one of the following commands:

```
Dism /online /Get-Drivers
```

```
Dism /online /Get-Drivers /all
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
Published Name	The name of the driver package after it is added to the driver store.	Oem0.inf
Original File Name	The original .inf file name of the driver package.	Toaster.inf
Inbox	<b>Yes</b> for a default driver (inbox driver) or <b>No</b> for third-party driver packages.	No
Class Name	The friendly name of the device class the driver is a member of.	Printer
Provider Name	The provider or digital signature for the driver package.	Microsoft

FIELD	DESCRIPTION	EXAMPLE
Date	The date associated with the driver, as it is specified in the .inf file. The date will be formatted appropriately for your locale.	10/31/2006
Version	The version number that is specified in the INF driverVer directive.	6.1.6801.0

### To get information about a specific driver

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. List information about a specific driver package in the offline Windows image. For example, type:

```
Dism /image:C:\test\offline /Get-DriverInfo /driver:oem1.inf
```

For a running operating system, type:

```
Dism /online /Get-DriverInfo /driver:oem1.inf
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
Published Name	The name of the driver package after it is added to the driver store.	Oem0.inf
Driver Store Path	The path to the driver location. If the driver is installed, the path to the driver store is listed. If the driver is not installed yet, the path to the driver on the servicing host is listed.	E:\Images\Mount_depset\Windows\System32\DriverStore\FileRepository\Fasttx2k.inf_x86_neutral_0328f62e\Fasttx2k.inf
Class Name	The friendly name of the device class the driver is a member of.	Printer
Class Description	The description of the device class the driver is a member of.	Printers
Class GUID	The GUID of the device class that the driver is a member of.	{4D36E97B-E325-11CE-BFC1-08002BE10318}
Date	The date associated with the driver, as it is specified in the .inf file. The date will be formatted appropriately for your locale.	8/6/2003

FIELD	DESCRIPTION	EXAMPLE
Version	The driver version number that is specified in the INF driverVer directive.	1.0.1.37
Boot Critical	<b>Yes</b> if the driver is boot critical or <b>No</b> if it is not.	<b>No</b>
Drivers for architecture	The architecture of the image that it is installed on. If the driver is not installed yet, the field is reported repeatedly for each supported operating system architecture.	x86
Manufacturer	The manufacturer of the supported device.	Adventure Works
Description	A description of the supported device.	<b>Windows XP Adventure Works 376 Controller</b>
Architecture	The architecture of the driver.	x86
Hardware ID	The hardware ID of the supported device.	<b>ABC_3376</b>
Service Name	The service name of the driver.	<b>C1232k</b>
Compatible IDs	Alternate Plug and Play (PnP) IDs for the device, if any apply.	<b>12ABC</b>
Exclude IDs	PnP IDs that will not match the device, if any apply.	<b>A_123</b>

#### Note

If you point to a driver that is not yet installed, the report will be slightly different.

## Get Package and Feature Information

You can use operating system package-servicing commands to obtain information about Windows packages. You can also use DISM and package-servicing commands to obtain information about Windows features, either offline or on a running Windows installation.

You can use the **/PackagePath** option to specify a .cab file or a folder where the .cab file is extracted. You cannot use this command to obtain package information for .msu files. Alternately, you can use **/Get-Packages** to find the name of a package, and then use **/PackageName** to specify the name of the package.

You can display detailed information about a feature. You must use the **/FeatureName** option with the **/Get** command. Use the **/Get-Features** option to find the name of the feature in the image. Feature names are case

sensitive if you are servicing a Windows image other than Windows 8.

For more information about operating system package-servicing commands available in DISM, see [DISM Operating System Package Servicing Command-Line Options](#).

### To list all packages in the image

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about all of the packages in the offline Windows image, type the following command:

```
Dism /image:C:\test\offline /Get-Packages
```

For a running operating system, type the following command:

```
Dism /online /Get-Packages
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
Package Identity	The name of the package as it appears in the image.	<b>Microsoft-Windows-NetFx3-OC-Package~31bf3856ad364e35~x86~en-US~6.1.6772.0</b>
State	The current state of the package. Such as:  <b>Installed.</b> The package is installed.  <b>Install Pending.</b> The package is installed but requires a reboot to complete the pending online actions.  <b>Staged.</b> The package is staged for installation.	<b>Installed</b>
Release Type	The type of package that it is. Such as:  <b>Feature Pack.</b> A Windows operating system feature.  <b>Language Pack.</b> A Windows operating system Language pack or Language Interface Pack (LIP).  <b>Foundation.</b> Core operating system components including optional features.	Feature Pack
Install Time	The UTC date and time when the installation occurred. If the package is not installed yet, the Install Time field is left blank.	8/18/2008 7:58:00 PM

## To list information about a specific package

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about a specific package in the offline Windows image, type one of the following commands:

```
Dism /image:C:\test\offline /Get-PackageInfo /PackagePath:C:\packages\package.cab
```

```
Dism /image:C:\test\offline /Get-PackageInfo  
/PackageName:Microsoft.Windows.Calc.Demo~6595b6144ccf1df~x86~en~1.0.0.0
```

For a running operating system, type one of the following commands:

```
Dism /online /Get-PackageInfo /PackagePath:C:\packages\package.cab
```

```
Dism /online /Get-PackageInfo /PackageName:Microsoft.Windows.Calc.Demo~6595b6144ccf1df~x86~en~1.0.0.0
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
Package Identity	The name of the package as it appears in the image.	<b>Microsoft-Windows-NetFx3-OC-Package~31bf3856ad364e35~x86~en-US~6.1.6772.0</b>
Applicable	Indicates if the package applies to the image.	<b>No</b>
Copyright	Copyright information for the package.	Copyright© Microsoft Corporation. All Rights Reserved.
Company	The company that provided the package, if available.	Microsoft Corporation
Creation Time	The date and time the package was created, if available.	8/18/2008 7:58:00 PM
Description	A brief description of the package.	Fix for KB300106
Install Client	The client tool that installed the package.	DISM Package Manager Provider
Install Package Name	The installed package.mum file name.	<b>Microsoft-Windows-NetFx3-OC-Package~31bf3856ad364e35~x86~en-US~6.1.6772.0.mum</b>

FIELD	DESCRIPTION	EXAMPLE
Install Time	The date and time the package was installed. If the package is not installed yet, the <b>Install Time</b> field is left blank.	8/18/2008 7:58:00 PM
Last Update Time	The date the package was last updated, if available.	8/18/2008 7:58:00 PM
Name	<p>The display name of the package, localized if available.</p> <p>Generally, "default" will be displayed for all servicing packages.</p>	ActiveX® Installer Service
Product Name	The name of the product that the package belongs to, if available.	<b>Microsoft-Windows-NetFx3-OC-Package</b>
Product Version	The version of the product that the package belongs to, if available.	123.01.0000
Release Type	<p>The type of package that it is. Such as:</p> <p><b>Feature Pack.</b> A Windows operating system feature.</p> <p><b>Language Pack.</b> A Windows operating system Language pack or Language Interface Pack (LIP).</p> <p><b>Foundation.</b> Core operating system components including optional features.</p>	Feature Pack
Restart Required	Indicates if a reboot is required when you install or uninstall the package online.	Possible
Support Information	Where to find support information, if available.	<a href="https://support.microsoft.com/?kbid=300106">https://support.microsoft.com/?kbid=300106</a>

FIELD	DESCRIPTION	EXAMPLE
State	<p>Indicates if the package is installed in the operating system. Possible values include the following:</p> <p><b>Not Present.</b> The package is not installed.</p> <p><b>Installed.</b> The package is installed.</p> <p><b>Install Pending.</b> The package will be installed but requires a reboot to complete pending online actions.</p> <p><b>Staged.</b> The package is staged for installation.</p>	Installed
Completely offline capable	<p><b>Yes.</b> The package can be installed offline without booting the image.</p> <p><b>No.</b> You must boot into the image in order to complete installation of this package.</p> <p><b>Undetermined.</b> You may have to boot into the image in order to complete the installation of this package. Many packages can be installed offline entirely. If you attempt to install a package offline and a reboot is required, it will be reported in the log file. You can check the status of a package using the Get-PackageInfo command.</p> <p>This field is only applicable to Windows 8, Windows Server® 2012, and Windows Preinstallation Environment (Windows PE) 4.0 target images.</p>	
Custom Properties	A list of custom properties defined in the package manifest file. If there are no custom properties, (No custom properties found) will be displayed.	Dependency: Language Pack
Features listing for package	<p>A list of the features found in the package.</p> <p>If there is no feature in the package, the package identity will be displayed followed by (No features found for this package).</p>	<b>Microsoft-Windows-NetFx3-OC-Package~31bf3856ad364e35~x86~en-US~6.1.6772.0 (No features found for this package)</b>

### To list all features in the image

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about the features in the offline Windows image, type one of the following commands:

```
Dism /image:C:\test\offline /Get-Features
```

```
Dism /image:C:\test\offline /Get-Features  
/PackageName:Microsoft.Windows.Calc.Demo~6595b6144ccf1df~x86~en~1.0.0.0
```

```
Dism /image:C:\test\offline /Get-Features /PackagePath:C:\packages\package.cab
```

For a running operating system, type one of the following commands:

```
Dism /online /Get-Features
```

```
Dism /online /Get-Features /PackageName:Microsoft.Windows.Calc.Demo~6595b6144ccf1df~x86~en~1.0.0.0
```

```
Dism /online /Get-Features /PackagePath:C:\packages\package.cab
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
Feature Name	The name of the feature as it appears in the image.	<b>InboxGames</b>

FIELD	DESCRIPTION	EXAMPLE
State	<p>The current state of the feature. Possible values include the following:</p> <ul style="list-style-type: none"> <li>• <b>Enabled.</b> The feature is enabled.</li> <li>• <b>Disabled.</b> The feature is disabled.</li> <li>• <b>Enable Pending.</b> The feature will be enabled but requires a reboot to complete pending online actions.</li> <li>• <b>Disable Pending.</b> The feature will be disabled but requires a reboot to complete pending online actions.</li> <li>• <b>Disabled with Payload Removed.</b> The feature is disabled and its payload has been removed. Only the package metadata is present in the image. The payload can be restored and the feature can be enabled on demand after the image is deployed. For more information about features on demand, see <a href="#">Configure a Windows Repair Source</a>.</li> </ul>	<b>Disabled</b>

### To list information about a specific feature

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about a specific feature in the offline Windows image, type one of the following commands:

```
Dism /image:C:\test\offline /Get-FeatureInfo /FeatureName:Hearts
```

```
Dism /image:C:\test\offline /Get-FeatureInfo /FeatureName:LocalPack-GB /PackageName:Microsoft-Windows-LocalPack-GB-Package~6595b6144ccf1df~x86~~1.0.0.0
```

For a running operating system, type the following command:

```
Dism /online /Get-FeatureInfo /FeatureName:Hearts
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
Feature Name	Name of the feature.	<b>InboxGames</b>
Display Name	The name of the feature as it appears in the user interface.	<b>Games</b>
Description	A brief description of the feature.	Standard inbox games.
Restart Required	Indicates if a restart is required when you enable or disable this feature.	<b>Yes</b>
State	<p>The current state of the feature. Possible values include the following:</p> <p><b>Enabled.</b> The feature is enabled.</p> <p><b>Disabled.</b> The feature is disabled.</p> <p><b>Enable Pending.</b> The feature will be enabled but requires a reboot to complete pending online actions.</p> <p><b>Disable Pending.</b> The feature will be disabled but requires a reboot to complete pending online actions.</p> <p><b>Disabled with Payload</b></p> <p><b>Removed.</b> The feature is disabled and its payload has been removed. Only the package metadata is present in the image. The payload can be restored and the feature can be enabled on demand after the image is deployed. For more information about features on demand, see <a href="#">Configure a Windows Repair Source</a>.</p>	<b>Disabled</b>
Custom Properties	A list of custom properties defined in the package manifest file. If there are no custom properties, (No custom properties found) will be displayed.	Dependency: Language Pack

## Get App Package (.appx) Servicing Information

You can use the app package (.appx) servicing commands to list the provisioned apps in a Windows image. Provisioned apps will be registered for every user profile that is created for the Windows image.

For more information about app package servicing commands available in DISM, see [DISM App Package \(.appx or .appxbundle\) Servicing Command-Line Options](#).

### To list provisioned apps in the Windows image

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list provisioned apps in a mounted offline Windows image, type:

```
Dism /image:c:\test\offline /Get-ProvisionedAppxPackages
```

For a running operating system, type:

```
Dism /online /Get-ProvisionedAppxPackages
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
DisplayName	The name of the app.	Fabrikam.Sample.CS
Version	The version number of the app package.	1.0.0.0
Architecture	The architecture of the app.	neutral
ResourceId	For more information, see <a href="#">App packaging glossary</a> .	
PackageName	The full name of the app package.	Fabrikam.Sample.CS_1.0.0.0_neutral_s9y1p3hwd5qda

## Get International Settings and Languages

The international servicing commands can be used to query existing international settings in Windows and Windows PE images. For more information about operating system package-servicing commands available in DISM, see [DISM Languages and International Servicing Command-Line Options](#).

### Important

International servicing commands cannot be used on a Windows Vista or Windows Server 2008 image.

Use the **/online** option to display information about international settings and languages in the running operating system. Use **/image: <path\_to\_offline\_image\_directory>** to display information about international settings and languages in the offline image. When used with the **/image** and **/distribution** options, information about international settings and languages in the distribution is displayed.

### To list all international settings and languages

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about all of the international settings in the offline Windows image, type one of the following commands:

```
Dism /image:C:\test\offline /Get-Intl
```

```
Dism /image:C:\test\offline /distribution:C:\windows_distribution\langpacks /Get-Intl
```

For a running operating system, type the following command:

```
Dism /online /Get-Intl
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
Default system UI language	The language that is currently set as the default system UI language.	<b>en-US</b>
System locale	The language for non-Unicode programs (also referred to as system locale) and font settings.	<b>en-US</b>
Default timezone	The time zone that is currently set as the default.	Pacific Standard Time
User locale for default user	The "standards and formats" language (also referred to as user locale) that is set for the default user.	<b>en-US</b>
Location	The geographical location that is currently set for the operating system. For more information about geographical locations, see <a href="#">Table of Geographical Locations</a> .	United States
Active keyboards	The value pair for the active keyboard. In the example provided, 0409 is the language identifier and 00000409 is the keyboard identifier.	<b>0409:00000409</b>
Default keyboards	The value pair for the default keyboard. In the example provided, 0409 is the language identifier and 00000409 is the keyboard identifier.	<b>0409:00000409</b>
Installed language(s)	A list of all installed language packs.	<b>en-US</b>

FIELD	DESCRIPTION	EXAMPLE
Type	The type of each installed language pack. For more information, see <a href="#">Add Language Packs to Windows</a> .	<p><b>en-US</b>  Type: Fully localized language  ar-SA  Type: Partially localized language, MUI type  Fallback Languages <b>en-US, fr-FR</b></p>
Distribution languages	A list of the languages that are available in the distribution share.	<p>The default language in the distribution is: <b>ja-JP</b></p> <p>The other available languages in the distribution are: <b>bg-BG, nl-NL</b></p> <p><b>Note</b>  This list includes the name of the folder in the distribution share. The language of the actual LP.cab file in the folder is not validated. For example, if the path to the distribution is ... \Langpacks\bg-BG\lp.cab, the value of bg-BG will be reported as the language in the distribution share even if the LP.cab file is not the correct .cab file for bg-BG.</p>
Keyboard layered driver	A list of the keyboard drivers for Japanese or Korean keyboards, if any are installed.	Japanese Keyboard (106/109 Key)

## Get Windows Edition Information

You can use the edition-servicing commands to obtain information about which editions of Windows are available for upgrade.

Target editions are the editions of Windows that you can upgrade to. You can display information about the current edition or the target edition of an offline Windows image or a running operating system.

For more information about Windows edition servicing commands available in DISM, see [DISM Windows Edition-Servicing Command-Line Options](#).

### To get information about the current Windows editions

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about the current edition of the offline Windows image, type the following command:

```
Dism /image:C:\test\offline /Get-CurrentEdition
```

For a running operating system, type the following command:

```
Dism /online /Get-CurrentEdition
```

## To get information about target editions of Windows

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about the target edition of the offline Windows image, type the following command:

```
Dism /image:C:\test\offline /Get-TargetEditions
```

For a running operating system, type the following command:

```
Dism /online /Get-TargetEditions
```

## Get Application Patch Information

Application servicing command-line options can be used on a offline image to check the applicability of Microsoft® Windows® Installer application patches (.msp files) and to query your offline image for information about installed Windows Installer applications (.msi files) and application patches (.msp files).

You can display detailed information about installed MSP patches filtered by patch and application. If the **/PatchCode** option is specified, detailed information is displayed for all Windows Installer applications that the patch is applied to. If the **/ProductCode** option is specified, information about all MSP patches in the specified application is displayed.

If the **/PatchCode** and **/ProductCode** options are both specified, information is displayed only if that specific patch is applied to the specified Windows Installer application. If the **/PatchCode** and **/ProductCode** options are not specified, all installed Windows Installer packages and MSP patches are displayed.

For more information about application servicing commands available in DISM, see [DISM Application Servicing Command-Line Options](#).

### To list information about installed MSP patches

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about the MSP patches, type one of the following commands:

```
Dism /image:C:\test\offline /Get-AppPatchInfo
```

```
Dism /image:C:\test\offline /Get-AppPatchInfo /PatchCode:{B0B9997C-GUID-GUID-GUID-74D866BBDFFF}
```

```
Dism /image:C:\test\offline /Get-AppPatchInfo /ProductCode:{B0F9497C-GUID-GUID-GUID-74D866BBDF59}
```

```
Dism /image:C:\test\offline /Get-AppPatchInfo /PatchCode:{B0B9997C-GUID-GUID-GUID-74D866BBDFFF} /ProductCode:{B0F9497C-GUID-GUID-GUID-74D866BBDF59}
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
Patch Code	A GUID identifying a specific Windows Installer package. The package code associates an .msi file together with an application or product and can also be used for the verification of sources.	<b>{8ACD2816-595D-48AA-A43B-3523CAA4F692}</b>
Product Code	A GUID that is the principal identification of an application or product.	{7764DEFC-C5D1-413C-8428-2AA903BF6DAA}
Patch Name	The registered display name for the patch. For patches that do not include the <b>DisplayName</b> property in the MsiPatchMetadata table, the returned display name is an empty string.	QFE9 - Non Removable
Patch State	1 if this patch is currently applied to the product. 2 if this patch has been superseded by another patch. 4 if this patch has been made obsolete by another patch.	1 (Applied)
Patch Uninstallable	1 if the patch is marked as possible to uninstall from the product. In this case, the installer can still block the uninstallation if this patch is required by another patch that cannot be uninstalled. Otherwise 0 is reported.	0
Help Link	Where to find support information, if available.	<a href="https://www.microsoft.com">https://www.microsoft.com</a>
Transforms	The set of patch transforms applied to the product by the last patch installation. This value may not be available for per-user unmanaged applications if the user is not logged on to the computer.	<b>:App1RTMToApp1QFE9;:#App1RTMToApp1QFE9</b>
Local Package	The location of the local cached patch file that is used by the product.	<b>C:\Windows\Installer\132f5c.msP</b>
Install Date	The date when the patch was applied to the product.	20080912

## To list information about MSP patches applied to an application

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about the MSP patches, type one of the following commands:

```
Dism /image:C:\test\offline /Get-AppPatches
```

```
Dism /image:C:\test\offline /Get-AppPatches /ProductCode:{B0F9497C-GUID-GUID-GUID-74D866BBDF59}
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
Patch Code	A GUID identifying a particular Windows Installer package. The package code associates an .msi file together with an application or product and can also be used for the verification of sources.	<b>{8ACD2816-595D-48AA-A43B-3523CAA4F692}</b>
Product Code	A GUID that is the principal identification of an application or product.	<b>{7764DEFC-C5D1-413C-8428-2AA903BF6DAA}</b>
Patch Name	The registered display name for the patch. For patches that do not include the <b>DisplayName</b> property in the MsiPatchMetadata table, the returned display name is an empty string.	QFE9 - Non Removable

## To list information about all Windows Installer applications

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about the MSP patches, type the following command:

```
Dism /image:C:\test\offline /Get-Apps
```

The report generated lists the product code and product name for applications that are installed in the offline image. For example:

Product Code : **{DB935363-5A68-47AF-A55A-CFC90F2E83BC}**

Product Name : MsiTestApplication

## To list information about a specific Windows Installer application

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. To list information about the MSP patches, type the following command:

```
Dism /image:C:\test\offline /Get-AppInfo /ProductCode:{B0F9497C-GUID-GUID-GUID-74D866BBDF59}
```

The report generated includes the following information:

FIELD	DESCRIPTION	EXAMPLE
Product Code	A GUID that is the principal identification of an application or product.	<b>{DB935363-5A68-47AF-A55A-CFC90F2E83BC}</b>
Product Name	The name of the application.	MsiTestApplication2
Product State	<p>The installation state for the product at initialization.</p> <p>-1 if the product is neither advertised nor installed.</p> <p>1 if the product is advertised but not installed.</p> <p>2 if the product is installed for a different user.</p> <p>5 if the product is installed for the current user.</p>	5 (Installed)
Package Code	A GUID identifying a particular Windows Installer package. The package code associates an .msi file together with an application or product and can also be used for the verification of sources.	<b>{C67CA1AE-6074-4810-BD74-F6BBB609744A}</b>
Product Version	The version of the product in string format.	1.0.0
Assignment Type	<p>0 if the product is advertised or installed per-user.</p> <p>1 if the product is advertised or installed per-computer for all users.</p>	1 (Per-Machine)
Publisher	The name of the manufacturer for the product.	Microsoft MSI Test
Language	The decimal identifier for the product language.	1033
Install Source	The directory that contains the source .cab file or the source file tree of the installation package.	<b>E:\Testpkg\App2_RTM&lt;/strong&gt;</b>

FIELD	DESCRIPTION	EXAMPLE
Package Name	The name of the original installation package.	MsiTestApplication2.msi
Help Link	Where to find support information, if available.	<a href="https://www.microsoft.com/management">https://www.microsoft.com/management</a>
Transforms	The set of patch transforms applied to the product by the last patch installation. This value may not be available for per-user unmanaged applications if the user is not logged on to the computer.	C:\Windows\Installer\{BDB20E90-3ACD-450B-BBDE-61E39687C6B1}\ACBlueT02.mst
Local Package	The location of the local cached package.	C:\Windows\Installer\132f3b.msi
Install Date	The date the application was installed.	<b>20080912</b>

## Related topics

[Service a Windows Image Using DISM](#)

[Service a Windows PE Image with DISM](#)

[Deployment Image Servicing and Management \(DISM\) Best Practices](#)

# Prepare recovery tools for your Windows images

1/24/2019 • 2 minutes to read

## TIP

If you want to reset a computer that runs Windows 10, see [Recovery options in Windows 10](#).

OEMs: When you create custom Windows 10 and Windows Server images, update the recovery tools.

## Windows Recovery Environment (Windows RE)

Both Windows 10 and Windows Server include a recovery environment that can repair common causes of unbootable operating systems.

Some customizations that you add to Windows should also be added to Windows RE:

CUSTOMIZATION	WHAT DO I NEED TO UPDATE?
<b>Boot-critical drivers or languages</b>	If you add these to your Windows image, add those same <a href="#">drivers and languages (when available)</a> to Windows RE.
<b>Custom recovery tools</b>	Add a custom tool to the Windows RE boot options menu
<b>Dedicated hardware button</b>	Add a dedicated hardware button to boot immediately to WinRE

More options:

- [Deploy Windows RE](#): If you're deploying Windows using a .wim file, use these instructions to deploy the recovery partition and hide the recovery drive letters.

## Push-button reset

Windows 10 users can reset their device, either keeping data and apps intact or resetting it for a new user.

Some customizations that you add to Windows should also be added to the push-button reset tools:

CUSTOMIZATION	WHAT DO I NEED TO UPDATE?
<b>Universal Windows apps</b>	Automatically restored.
<b>Drivers</b> installed using an .inf file	Automatically restored.
<b>Desktop apps</b> or <b>drivers</b> installed using an .exe file	<a href="#">Capture into a provisioning package</a> or a <a href="#">siloe provisioning package (SPP)</a> .
<b>Out of Box Experience customizations</b> , <b>Start Menu</b> , and <b>Unattend.xml settings</b>	In Windows 10, version 1809 and later, save a copy in your <a href="#">Auto-apply folders</a> . In earlier versions or as an alternative, use <a href="#">extensibility scripts instead</a> .

<b>Settings</b> created using Windows Configuration Designer (also known as ICD)	Copy the settings .ppkg file to C:\Recovery\Customizations.
--	---

More options:

- Use [Compact OS, single-sourcing, and image optimization](#) to save space on the disk.

## Bare metal recovery

If the user needs to replace their hard drive or completely wipe it, they can use bootable USB key to restore their device.

Some customizations that you add to Windows should also be added to the push-button reset tools:

CUSTOMIZATION	WHAT DO I NEED TO UPDATE?
<b>Drive partitions</b>	If your device uses a non-standard drive partition layout, <a href="#">update bare-metal reset so users can create their own recovery media</a> .
Ship <b>USB recovery media</b> to customers	<a href="#">Create recovery media while deploying new devices</a>

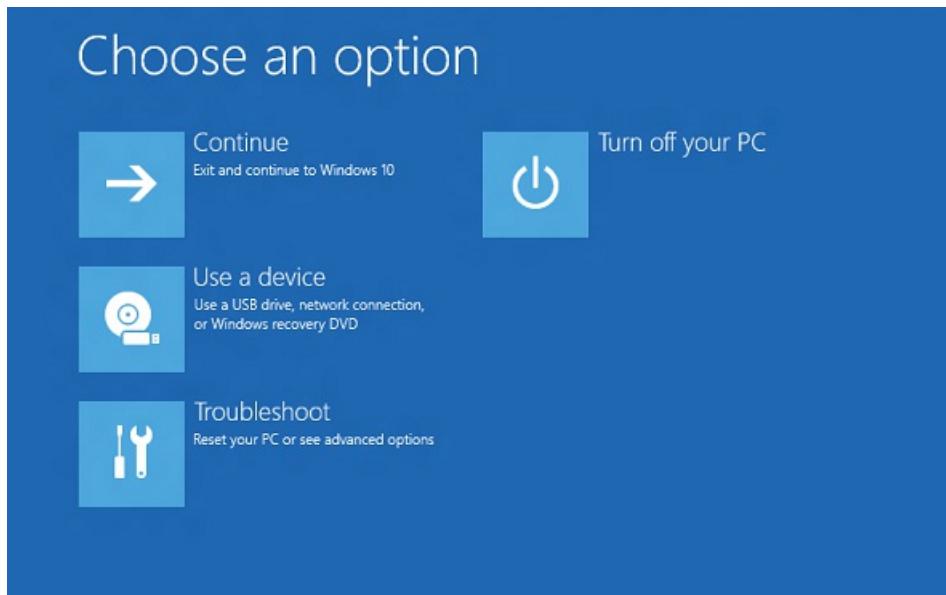
## Reference

- [How push-button reset features work](#)
- [Push-button reset frequently-asked questions \(FAQ\)](#)
- [REAgentC command-line options](#)
- [ResetConfig XML reference](#)
- [WinREConfig XML reference](#)
- [Windows RE troubleshooting features](#)

# Windows Recovery Environment (Windows RE)

4/4/2019 • 8 minutes to read

Windows Recovery Environment (WinRE) is a recovery environment that can repair common causes of unbootable operating systems. WinRE is based on Windows Preinstallation Environment (Windows PE), and can be customized with additional drivers, languages, Windows PE Optional Components, and other troubleshooting and diagnostic tools. By default, WinRE is preloaded into the Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) and Windows Server 2016 installations.



## What's new with WinRE for Windows 10?

- By default, if you install Windows using media created from Windows Imaging and Configuration Designer (ICD), you'll get a dedicated WinRE tools partition on both UEFI and BIOS-based devices, located immediately after the Windows partition. This allows Windows to replace and resize the partition as needed. (If you install Windows by using Windows Setup, you'll get the same partition layout that you did in Windows 8.1.)
- If you [add a custom tool to the WinRE boot options menu](#), it can only use optional components that are already in the default WinRE tools. For example, if you have a app from Windows 8 that depended on the .NET optional components, you'll need to rewrite the app for Windows 10.
- If you [add a custom tool to the WinRE boot options menu](#), it must be placed in the \Sources\Recovery\Tools folder so that it can continue to work after future WinRE upgrades.
- When adding languages to the push-button reset tools, you'll now need to add the WinPE-HTA optional component.

## Tools

WinRE includes these tools:

- **Automatic repair and other troubleshooting tools.** For more info, see [Windows RE Troubleshooting Features](#).
- **Push-button reset** (Windows 10 for desktop editions , Windows 8.1 and Windows 8 only). This tool enables your users to repair their own PCs quickly while preserving their data and important customizations, without having to back up data in advance. For more info, see [Push-Button Reset Overview](#).
- **System image recovery** (Windows Server 2016, Windows Server 2012 R2 and Windows Server 2012 only).

This tool restores the entire hard drive. For more info, see [Recover the Operating System or Full Server](#).

In addition, you can create your own custom recovery solution by using the [Windows Imaging API](#), or by using the [Deployment Image Servicing and Management \(DISM\) API](#).

## Entry points into WinRE

Your users can access WinRE features through the **Boot Options** menu, which can be launched from Windows in a few different ways:

- From the login screen, click Shutdown, then hold down the Shift key while selecting **Restart**.
- In Windows 10, select **Start > Settings > Update & security > Recovery** > under **Advanced Startup**, click **Restart now**.
- Boot to recovery media.
- Use a [hardware recovery button \(or button combination\)](#) configured by the OEM.

After any of these actions is performed, all user sessions are signed off and the **Boot Options** menu is displayed. If your users select a WinRE feature from this menu, the PC restarts into WinRE and the selected feature is launched.

WinRE starts automatically after detecting the following issues:

- Two consecutive failed attempts to start Windows.
- Two consecutive unexpected shutdowns that occur within two minutes of boot completion.
- A Secure Boot error (except for issues related to Bootmgr.efi).
- A BitLocker error on touch-only devices.

### Boot options menu

This menu enables your users to perform these actions:

- Start recovery, troubleshooting, and diagnostic tools.
- Boot from a device (UEFI only).
- Access the **Firmware** menu (UEFI only).
- Choose which operating system to boot, if multiple operating systems are installed on the PC.

#### NOTE

You can add one custom tool to the **Boot options** menu. Otherwise, these menus can't be further customized. For more info, see [Add a Custom Tool to the Windows RE Boot Options Menu](#).

## Security considerations

When working with WinRE, be aware of these security considerations:

- If users open the **Boot options** menu from Windows and select a WinRE tool, they must provide the user name and password of a local user account with administrator rights.
- By default, networking is disabled in WinRE. You can turn on networking when you need it. For better security, disable networking when you don't need connectivity.

## Customizing WinRE

You can customize WinRE by adding packages (Windows PE Optional Components), languages, drivers, and custom diagnostic or troubleshooting tools. The base WinRE image includes these Windows PE Optional Components:

- Microsoft-Windows-Foundation-Package
- WinPE-EnhancedStorage
- WinPE-Rejuv
- WinPE-Scripting
- WinPE-SecureStartup
- WinPE-Setup
- WinPE-SRT
- WinPE-WDS-Tools
- WinPE-WMI
- WinPE-StorageWMI-Package (added to the base image in Windows 8.1 and Windows Server 2012 R2)
- WinPE-HTA (added to the base image in Windows 10)

**Note** The number of packages, languages, and drivers is limited by the amount of memory available on the PC. For performance reasons, minimize the number of languages, drivers, and tools that you add to the image.

## Hard drive partitions

When you install Windows by using Windows Setup, WinRE is configured like this:

1. During Windows Setup, Windows prepares the hard drive partitions to support WinRE.
2. Windows initially places the WinRE image file (winre.wim) in the Windows partition, in the \Windows\System32\Recovery folder.

Before delivering the PC to your customer, you can modify or replace the WinRE image file to include additional languages, drivers, or packages.

3. During the specialize configuration pass, the WinRE image file is copied into the recovery tools partition, so that the device can boot to the recovery tools even if there's a problem with the Windows partition.

When you deploy Windows by applying images, you must manually configure the hard drive partitions. When WinRE is installed on a hard drive, the partition must be formatted as NTFS.

Add the baseline WinRE tools image (winre.wim) to a separate partition from the Windows and data partitions. This enables your users to use WinRE even if the Windows partition is encrypted with Windows BitLocker Drive Encryption. It also prevents your users from accidentally modifying or removing the WinRE tools.

Store the recovery tools in a dedicated partition, directly after the Windows partition. This way, if future updates require a larger recovery partition, Windows will be able to handle it more efficiently by adjusting the Windows and recovery partition sizes, rather than having to create a new recovery partition size while the old one remains in place.

To learn more, see [Configure UEFI/GPT-Based Hard Drive Partitions](#) or [Configure BIOS/MBR-Based Hard Drive Partitions](#).

## Memory requirements

In order to boot Windows RE directly from memory (also known as RAM disk boot), a contiguous portion of physical memory (RAM) which can hold the entire Windows RE image (winre.wim) must be available. To optimize memory use, manufacturers should ensure that their firmware reserves memory locations either at the beginning or at the end of the physical memory address space.

## Updating the on-disk Windows Recovery Environment

In Windows 10, the on-disk copy of Windows RE can be serviced as part of rollup updates for the OS. Not all

rollup updates will service Windows RE.

Unlike the normal OS update process, updates for Windows RE do not directly service the on-disk Windows RE image (winre.wim). Instead, a newer version of the Windows RE image replaces the existing one, with the following contents being injected or migrated into the new image:

- Boot critical and input device drivers from the full OS environment are added to the new Windows RE image.
- Windows RE customizations under \Sources\Recovery of the mounted winre.wim are migrated to the new image.

The following contents from the existing Windows RE image are not migrated to the new image:

- Drivers which are in the existing Windows RE image but not in the full OS environment
- Windows PE optional components which are not part of the default Windows RE image
- Language packs for Windows PE and optional components

The Windows RE update process makes every effort to reuse the existing Windows RE partition without any modification. However, in some rare situations where the new Windows RE image (along with the migrated/injected contents) does not fit in the existing Windows RE partition, the update process will behave as follows:

- If the existing Windows RE partition is located immediately after the Windows partition, the Windows partition will be shrunk and space will be added to the Windows RE partition. The new Windows RE image will be installed onto the expanded Windows RE partition.
- If the existing Windows RE partition is not located immediately after the Windows partition, the Windows partition will be shrunk and a new Windows RE partition will be created. The new Windows RE image will be installed onto this new Windows RE partition. The existing Windows RE partition will be orphaned.
- If the existing Windows RE partition cannot be reused and the Windows partition cannot successfully be shrunk, the new Windows RE image will be installed onto the Windows partition. The existing Windows RE partition will be orphaned.

**Important** To ensure that your customizations continue to work after Windows RE has been updated, they must not depend on functionalities provided by Windows PE optional components which are not in the default Windows RE image (e.g. WinPE-NetFX). To facilitate development of Windows RE customizations, the WinPE-HTA optional component has been added to the default Windows RE image in Windows 10.

**Note** The new Windows RE image deployed as part of the rollup update contains language resources only for the system default language, even if the existing Windows RE image contains resources for multiple languages. On most PCs, the system default language is the language selected at the time of OOB.

## Known Issue

If the GPO "Windows Settings/Security Settings/Local Policies/Security Options/Accounts: Block Microsoft accounts" is set to enable the policy "User can't add or log with Microsoft account", attempting to restore the System in WinRE will fail with the error message "You need to sign in as an administrator to continue, but there aren't any administrator accounts on this PC."

This is a known issue and the workaround is to either avoid setting the "Accounts: Block Microsoft accounts" to "User can't add or log with Microsoft Account" or set the MDM policy Security/RecoveryEnvironmentAuthentication to 2.

## See also

CONTENT TYPE	REFERENCES
<b>Deployment</b>	<a href="#">Customize Windows RE</a>   <a href="#">Deploy Windows RE</a>
<b>Operations</b>	<a href="#">REAgentC Command-Line Options</a>
<b>Troubleshooting</b>	<a href="#">Windows RE Troubleshooting Features</a>
<b>Add-on tools</b>	<a href="#">Add a Custom Tool to the Windows RE Boot Options Menu</a>   <a href="#">Add a Hardware Recovery Button to Start Windows RE</a>   <a href="#">Push-Button Reset Overview</a>

# Customize Windows RE

1/18/2019 • 5 minutes to read

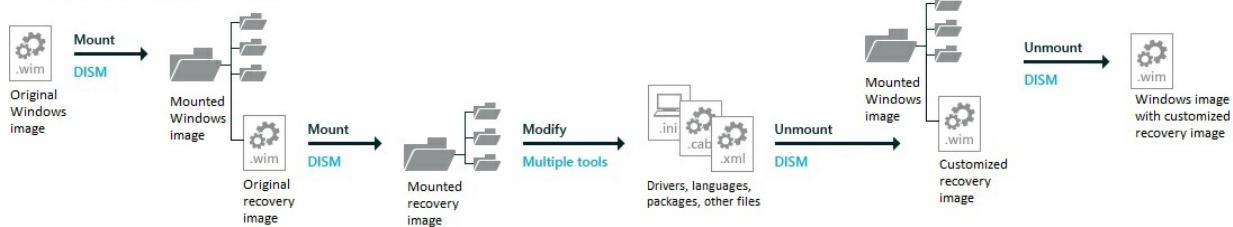
You can customize Windows Recovery Environment (Windows RE) by adding languages, packages drivers, and custom diagnostic or troubleshooting tools.

The WinRE image is included inside the Windows 10 and Windows Server 2016 images, and is eventually copied to the Windows RE tools partition on the destination PC or device. To modify it, you'll mount the Windows image, then mount the WinRE image inside it. Make your changes, unmount the WinRE image, then unmount the Windows image.

## NOTE

If your Windows image doesn't contain a WinRE image (usually because WinRE has been moved to the RE tools partition prior to capturing the image), Windows will still work but the recovery environment won't be configured or accessible. You can add WinRE.wim back into your Windows image by copying it from the `Windows\System32\Recovery` folder of a mounted Windows image back into the same folder of your custom image.

Customize the recovery image



We recommend that when you update your Windows images with languages and boot-critical drivers, update the Windows RE image at the same time.

This topic also gives optional steps to optimize the Windows RE image after updating it.

## Prerequisites

To complete this walkthrough, you need the following:

- A technician computer with the Windows Assessment and Deployment Kit (ADK) and [WinPE addon](#) installed.
- The Windows image (install.wim). This can be from the Windows installation media or from a reference image.

## Step 1: Mount the Windows and Windows RE image

### Mount the images

1. Open the **Deployment and Imaging Tools Environment** command prompt as an administrator:

Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator > Yes**.

2. Mount the Windows base image for editing.

```
md C:\mount\windows  
Dism /Mount-Image /ImageFile:C:\mount\install.wim /Index:1 /MountDir:C:\mount\windows
```

3. Mount the Windows RE image for editing.

```
md C:\mount\winre  
Dism /Mount-Image /ImageFile:c:\mount\windows\windows\system32\recovery\winre.wim /Index:1  
/MountDir:C:\mount\winre
```

**Note** The Windows RE image should always be index number 1.

## Step 2: Adding languages

When you add languages to Windows RE, you need to add the base language pack and the corresponding language packs for each of the Windows PE optional components in the Windows RE tools image.

Starting with Windows 10, Version 1607 and Windows Server 2016, the base language pack and optional component language packs required to customize Windows RE are included in the Language Pack DVDs for Windows 10 and Windows Server 2016. The Windows PE language packs in the Windows 10 ADK should not be used to customize Windows RE.

**Note** To ensure a consistent language experience in recovery scenarios, add the same set of languages to the Windows RE image that you add to the Windows image.

We recommend adding no more than ten language packs to a Windows or Windows RE image. Multiple language packs increase the size of the Windows image and also affect the overall performance of a system during deployment and servicing.

### To add language packs

1. Mount the Windows Language pack ISO with File Explorer. This will assign it a drive letter, for example, D.
2. List the Windows PE optional components in the Windows RE tools image:

```
Dism /Get-Packages /Image:C:\mount\winre
```

3. Review the resulting list of packages, and then add the corresponding language packs for each package in the image, including the base Windows PE language pack, but not including **WinPE-WiFi-Package**.

The following code shows how to add the French (fr-fr) language pack to the base Windows PE image, and then to each of the optional components that are present in the default Windows RE image:

```
Dism /Add-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\lp.cab"

Dism /Add-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-Rejuv_fr-fr.cab"

Dism /Add-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-EnhancedStorage_fr-fr.cab"

Dism /Add-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-Scripting_fr-fr.cab"

Dism /Add-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-SecureStartup_fr-fr.cab"

Dism /Add-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-SRT_fr-fr.cab"

Dism /Add-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-WDS-Tools_fr-fr.cab"

Dism /Add-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-WMI_fr-fr.cab"

Dism /Add-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-StorageWMI_fr-fr.cab"

Dism /Add-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-HTA_fr-fr.cab"
```

The **WinPE-WiFi-Package** is not language-specific and does not need to be added when adding other languages.

4. If you're adding language packs for Japan, Korea, or China, add the font packages for these languages.  
Here's an example for Japan:

```
Dism /image:C:\mount\winre /add-package /packagepath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-Font Support-JA-JP.cab"
```

To learn more, see [WinPE: Add packages \(Optional Components Reference\)](#).

5. To save space and speed up the recovery process, remove unneeded languages. Reverse the order to avoid problems with dependencies.

Note, the **WinPE-WiFi-Package** is not language specific and should not be removed.

```
Dism /Remove-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-HTA_en-us.cab"

Dism /Remove-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-StorageWMI_en-us.cab"

Dism /Remove-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-WMI_en-us.cab"

Dism /Remove-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-WDS-Tools_en-us.cab"

Dism /Remove-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-SRT_en-us.cab"

Dism /Remove-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-SecureStartup_en-us.cab"

Dism /Remove-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-Scripting_en-us.cab"

Dism /Remove-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-EnhancedStorage_en-us.cab"

Dism /Remove-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-Rejuv_en-us.cab"

Dism /Remove-Package /Image:C:\mount\winre /PackagePath:"D:\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\lp.cab"
```

## Step 3: Adding boot-critical drivers

Make sure that you add any third-party drivers that your reference device requires to boot, such as storage or video drivers. If you add boot-critical drivers to a Windows image using Windows Imaging and Configuration Designer (ICD), they'll be added to the Windows RE image inside that Windows image.

### Add a boot-critical driver

1. If necessary, unzip or unpack the driver file from your device manufacturer.
2. Identify the driver setup (.inf) file, and add it.

```
Dism /Image:C:\mount\winre /Add-Driver /Driver:"C:\SampleDriver\driver.inf"
```

where C:\SampleDriver\driver.inf is the location of the .inf file.

## Step 4: Adding a custom tool

You can add a custom troubleshooting or diagnostic tool to your Windows RE image. To learn more, see [Add a Custom Tool to the Windows RE Boot Options Menu](#).

## Step 5: Adding Windows updates

Occasionally, a Windows update may require you to update the Windows RE image.

- Add the Windows update package, for example, C:\MSU\Windows8.1-KB123456-x64.msu.

```
Dism /Add-Package /PackagePath:C:\MSU\Windows8.1-KB123456-x64.msu /Image:C:\mount\winre  
/LogPath:AddPackage.log
```

## Step 6: Optimizing the image, part 1 (optional)

After adding a language or Windows update package, you can reduce the size of the final Windows RE package by checking for duplicate files and marking the older versions as superseded.

1. Optimize the image:

```
Dism /Image:c:\mount\winre /Cleanup-Image /StartComponentCleanup
```

2. Later, you'll export the image to remove the superseded files.

## Step 7: Unmount the WinRE image

- Unmount and save the image:

```
Dism /Unmount-Image /MountDir:C:\mount\winre /Commit
```

## Step 8: Optimizing the image, part 2 (optional)

If you've optimized the image, you'll need to export the image in order to see a change in the file size. During the export process, DISM removes files that were superseded.

1. Export the Windows RE image into a new Windows image file.

```
Dism /Export-Image /SourceImageFile:c:\mount\windows\windows\system32\recovery\winre.wim  
/SourceIndex:1 /DestinationImageFile:c:\mount\winre-optimized.wim
```

2. Replace the old Windows RE image with the newly-optimized image.

```
del c:\mount\windows\windows\system32\recovery\winre.wim  
copy c:\mount\winre-optimized.wim c:\mount\windows\windows\system32\recovery\winre.wim
```

## Step 9: Unmount the Windows image

Save your changes back into the Windows base image.

- Unmount the base Windows image:

```
Dism /Unmount-Image /MountDir:C:\mount\windows /Commit
```

## Next Steps

If you're deploying Windows using **Windows Setup**, update the other Windows images inside the base Windows file (Install.wim).

If you're deploying your reference image by using **Windows PE**, **Diskpart**, and **DISM**, then continue to [Deploy Windows RE](#).

## Related topics

[Add a Custom Tool to the Windows RE Boot Options Menu](#)

[Deploy Windows RE](#)

[Deploy Push-Button Reset Features](#)

[REAgentC Command-Line Options](#)

# Add a custom tool to the Windows RE boot options menu

1/18/2019 • 3 minutes to read

You can add a custom troubleshooting or diagnostic tool to the Windows Recovery Environment (WinRE) image. This tool is displayed in the Boot Options menu.

By developing your custom tool to run in WinRE, you can leverage the touch and on-screen keyboard support available in WinRE.

New for Windows 10: You won't be able to add WinRE optional components that aren't already in the default WinRE tools. For example, if you have a app from Windows 8 that depended on the .NET optional components, you'll need to rewrite the app for Windows 10.

## To add a custom tool

1. Extract and mount a Windows image (install.wim) and its corresponding WinRE image (winre.wim):

```
md c:\mount
xcopy D:\sources\install.wim C:\mount
md C:\mount\windows
Dism /mount-image /imagefile:C:\mount\install.wim /index:1 /mountdir:C:\mount\windows
md C:\mount\winre
Dism /mount-image /imagefile:c:\mount\windows\windows\system32\recovery\winre.wim /index:1
/mountdir:C:\mount\winre
```

For more information about these steps, see the topic: [Customize Windows RE](#).

2. In Notepad, create a configuration file that specifies the custom tool's filename and parameters (if any):

```
<?xml version="1.0" encoding="utf-8"?>
<!-- WinREConfig.xml -->
<Recovery>
    <RecoveryTools>
        <RelativeFilePath>OEMDiagnostics.exe</RelativeFilePath>
        <CommandLineParam>/param1 /param2</CommandLineParam>
    </RecoveryTools>
</Recovery>
```

Where C:\Tools\OEMDiagnostics.exe is the custom troubleshooting or diagnostics tool, and where /param1 and /param2 are optional parameters used when running this custom tool.

**Note** You can only add one custom tool to the WinRE boot options menus.

Save the file using UTF-8 coding. Do not use ANSI:

Click **File**, and then click **Save As**. In the **Encoding** box, select **UTF-8**, and save this file as

C:\mount\WinREConfig.xml.

3. Create a \Sources\Recovery\Tools folder in the WinRE mount folder, and then copy the custom tool and its configuration file into the new folder:

```
md C:\mount\winre\sources\recovery\tools  
copy C:\Tools\OEMDiagnostics.exe C:\mount\winre\sources\recovery\tools  
copy C:\mount\WinREConfig.xml C:\mount\winre\sources\recovery\tools
```

The custom tool and any associated folders must be in this folder so that it can continue to work after future WinRE upgrades.

4. Commit your customizations and unmount the WinRE image:

```
Dism /unmount-image /mountdir:C:\mount\winre /commit
```

5. Optional: make a backup copy of the WinRE image.

```
copy C:\mount\windows\windows\system32\recovery\winre.wim C:\mount\winre_amd64_backup.wim
```

You can often reuse the same customizations on multiple images.

6. Unmount and save the changes from the base Windows image:

```
Dism /unmount-image /mountdir:C:\mount\windows /commit
```

## To deploy the image

1. In Notepad, create a configuration file that describes the custom tool in the boot options menu. Add descriptions for each language you support. This example specifies both English and French language versions of the tool name and description:

```
<?xml version="1.0" encoding="utf-8"?>  
<!-- AddDiagnosticsToolToBootMenu.xml -->  
<BootShell>  
  <WinRETool locale="en-us">  
    <Name>Fabrikam Utility</Name>  
    <Description>Troubleshoot your Fabrikam PC</Description>  
  </WinRETool>  
  <WinRETool locale="fr-fr">  
    <Name>Utilité de Fabrikam</Name>  
    <Description>Dépannez votre PC de Fabrikam</Description>  
  </WinRETool>  
</BootShell>
```

**Warning** Limit the `<Name>` and `<Description>` values to approximately 30 characters or less to make sure that they appear correctly in the boot options menu.

Save the file using UTF-8 coding:

Click **File**, and then click **Save As**. In the **Encoding** box, select **UTF-8**, and save this file as

```
E:\Recovery\BootMenu\AddDiagnosticsToolToBootMenu.xml .
```

Where *E:\* is the drive letter of a removable drive or network location.

2. On your destination computer, during image deployment, but after you register the custom WinRE boot image and the Windows operating system, you must register the description of the custom tool:

```
Reagentc /setbootshelllink /configfile E:\Recovery\BootMenu\AddDiagnosticsToolToBootMenu.xml
```

If the custom tool is registered properly, the output from running this command will be: <OEM Tool = 1>.

**Note** For more information about deploying Windows, see the [Deploy Windows RE](#) topic.

### To verify the custom tool appears in the Boot Options menu when launched from Windows

1. Restart the destination computer, and complete OOBЕ as your user.

#### Note

If you are prompted for a product key, click **Skip**.

2. Click **Start** > **PC settings**, and then select **General**.
3. In the **Advanced startup** section, select **Restart now**.

The Windows **Boot Options** menu appears.

4. In the **Boot Options** menu, select **Troubleshoot**, and then click the **Fabrikam Utility** link.

The computer restarts in WinRE, and the tool that is specified in the *<RecoveryTools>* section of the WinREConfig.xml file, appears.

5. Confirm that the custom tool works properly, and then close the tool.

If the custom tool does not appear on the Boot Options menu, you can try the following:

- Verify the WinREConfig.xml and the AddDiagnosticsToolToBootMenu.xml files are saved using the UTF-8 encoding format.
- Disable WinRE, register the custom tool again, and then enable WinRE. For example:

```
Reagentc /disable
Reagentc /setbootshelllink /configfile E:\Recovery\BootMenu\AddDiagnosticsToolToBootMenu.xml
Reagentc /enable
```

### To verify the custom tool appears in the WinRE recovery menu

6. In the recovery menu, select **Troubleshoot**, and then click the **Fabrikam Utility** link.
7. Confirm that the custom tool works properly, and then close the tool.
8. Click **Continue**.

The PC reboots into the operating system.

## Related topics

[Windows Recovery Environment \(Windows RE\) Technical Reference](#)

[Customize Windows RE](#)

[Deploy Windows RE](#)

[Windows RE Troubleshooting Features](#)

# Add a hardware recovery button to start Windows RE

1/18/2019 • 2 minutes to read

On UEFI-based computers, you can configure a hardware recovery button (or button combination) to start Windows RE, including push-button reset features for Windows 10 for desktop editions (Home, Pro, Enterprise, and Education). This can help users get to the Windows RE menus more easily.

Relative to Windows 8/8.1, the recommended implementation in Windows 10 for such hardware buttons has been greatly simplified. You no longer need to copy Windows boot files to an unmanaged location on the EFI system partition (ESP) to create a secondary boot path. Instead, Windows configures and manages all the on-disk resources required to support the hardware buttons. The design can be summarized as follows:

1. Windows 10 automatically creates a secondary Boot Configuration Data (BCD) store in the folder `\EFI\Microsoft\Recovery`.

When Windows RE is installed, this secondary BCD store is automatically populated with the appropriate settings to boot Windows RE by default.

If the location of Windows RE changes (for example, due to future updates), the secondary BCD store is updated automatically.

2. You will still need to create a static boot device entry for recovery at the end of the UEFI firmware boot order list.

This boot device entry should point to the default Windows Boot Manager (`bootmgfw.efi`) in the folder `\EFI\Microsoft\Boot` on the ESP.

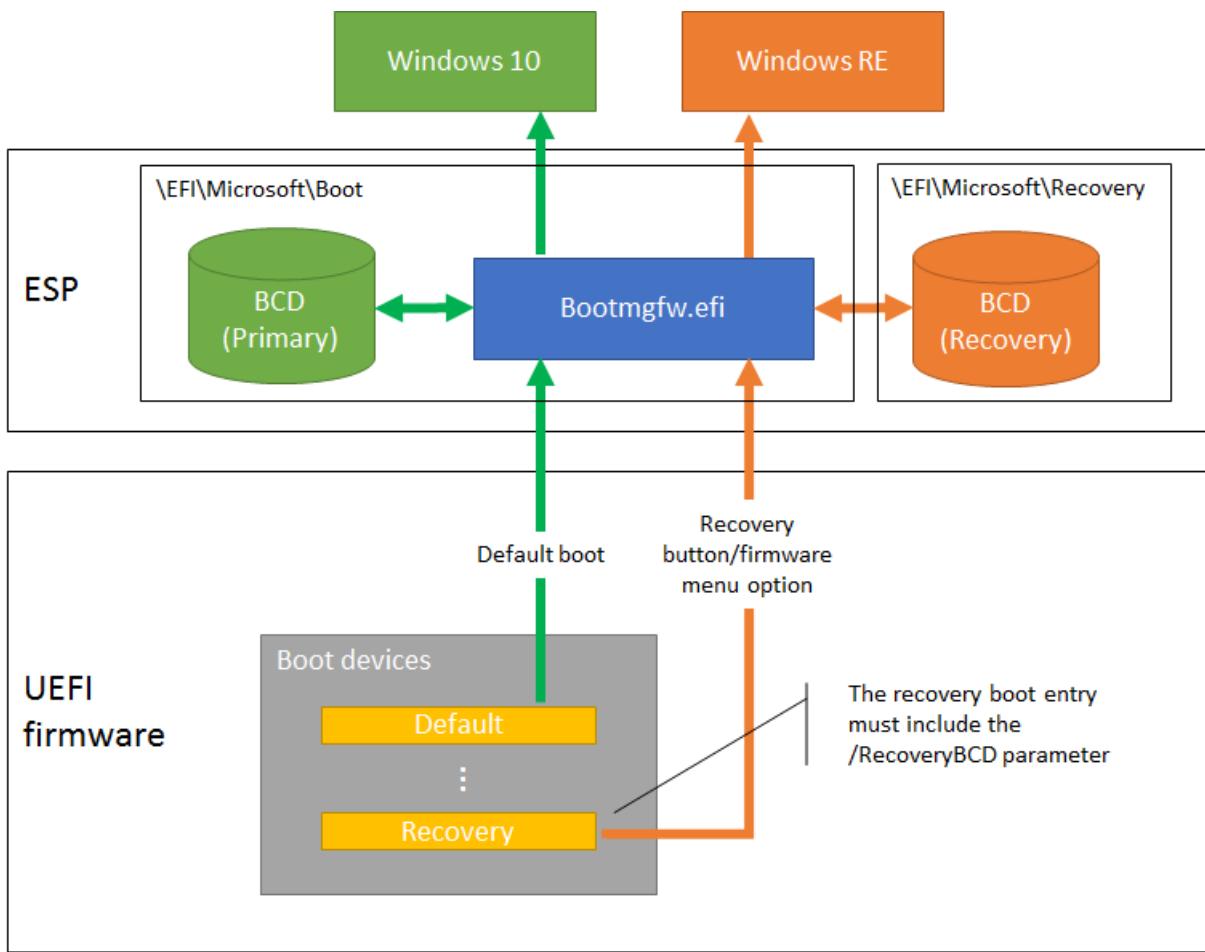
The boot device entry must specify the `/RecoveryBCD` parameter.

When the hardware button is triggered, the recovery boot device entry should be selected automatically.

To learn more, see your hardware manufacturer's instructions for modifying the UEFI firmware on the device.

3. When Windows Boot Manager is launched with the `/RecoveryBCD` parameter, it uses the secondary BCD store which is configured to boot Windows RE, instead of the default BCD store.

The following diagram illustrates the recommended implementation and the various boot paths:



## Design recommendations for the hardware button:

The hardware recovery button (or button combination) should be usable even when the PC is powered off. When triggered, the PC should power on and go through the secondary boot path. This eliminates the need for users to press the button within a very short time window during and after POST.

For PCs which support firmware options menu, triggering the button (or button combination) should first display a simple menu which gives users the options to either boot Windows RE or to enter the firmware options menu. This removes the need to support multiple button combinations.

**Note** The hardware button will not be able to boot the PC into Windows RE until Windows RE is installed. In general, this means after the PC has completed the Specialize configuration pass.

## Related topics

[Deploy Windows RE](#)

# Deploy Windows RE

1/18/2019 • 3 minutes to read

Use these steps to deploy Windows Recovery Environment (Windows RE) to a new computer, to help end users repair a PC when a system failure occurs.

## Prerequisites

To complete this walkthrough, you need the following:

- A destination computer that has been configured with a Windows RE tools partition, and optionally, a recovery image partition. For more information, see [Capture and Apply Windows, System, and Recovery Partitions](#).
- Optional: Customize your recovery media. For more information, see [Customize Windows RE](#).
- Optional: Customize your recovery media to include custom tools. For more information, see [Add a Custom Tool to the Windows RE Boot Options Menu](#).

## Step 1: Deploy Windows RE

1. Create a new directory in the Windows RE Tools partition, and then copy your custom Windows RE tools image (Winre.wim) to this directory. The following are examples based on your firmware type:

### UEFI:

```
mkdir T:\Recovery\WindowsRE  
xcopy /h W:\Windows\System32\Recovery\Winre.wim T:\Recovery\WindowsRE
```

where *T*: is the drive letter of your Windows RE Tools partition. For example:

### BIOS:

```
mkdir S:\Recovery\WindowsRE  
xcopy /h W:\Windows\System32\Recovery\Winre.wim S:\Recovery\WindowsRE
```

where *S*: is the system partition.

2. Register your custom Windows RE tools image:

### UEFI:

```
C:\Windows\System32\Reagentc /setreimage /path T:\Recovery\WindowsRE /target W:\Windows
```

where *T*: is the Windows RE Tools partition.

### BIOS

```
C:\Windows\System32\Reagentc /setreimage /path S:\Recovery\WindowsRE /target W:\Windows
```

where *S*: is the System partition.

3. Optional: If you have added a custom tool to your Windows RE boot image, register it so that it will appear on the **Boot Options** menu:

```
Reagentc /setbootshelllink /configfile E:\Recovery\BootMenu\AddDiagnosticsToolToBootMenu.xml
```

For more information about adding a custom tool, see [Add a Custom Tool to the Windows RE Boot Options Menu](#).

4. Optional: Configure a hardware recovery button (or button combination) to run a secondary boot path that contains Windows RE. For more information, see [Add a Hardware Recovery Button to Start Windows RE](#).

## Step 2: Identify the Recovery Partitions and Hide the Drive Letters

**Note** If you want to configure push-button reset features for Windows 8 editions, skip this section, and go to the topic: [Deploy Push-Button Reset Features](#).

Configure your partitions as recovery partitions, and then conceal the drive letters so the partitions don't appear in common Windows menus, such as File Explorer.

### Prepare a DiskPart script to identify the recovery partitions and to hide drive letters

1. In Notepad, create a text file that includes commands to identify and hide the recovery partitions. The following examples are based on your firmware type:

#### UEFI:

Use the ID: PARTITION\_MSFT\_RECOVERY\_GUID (de94bba4-06d1-4d40-a16a-bfd50179d6ac) to define the partitions as recovery partitions.

Use the GPT attributes: 0x8000000000000001 to hide the drive letters and to mark them as required, by using a combination of two attributes: GPT\_BASIC\_DATA\_ATTRIBUTE\_NO\_DRIVE\_LETTER and GPT\_ATTRIBUTE\_PLATFORM\_REQUIRED.

For more information about UEFI hard drive partition attributes, see [PARTITION\\_INFORMATION\\_GPT structure](#).

```
rem == HideRecoveryPartitions-UEFI.txt
select disk 0
select partition 1
remove
set id=de94bba4-06d1-4d40-a16a-bfd50179d6ac
gpt attributes=0x8000000000000001
rem == If Push-button reset features are included, add the following commands:
rem   select partition 5
rem   remove
rem   set id=de94bba4-06d1-4d40-a16a-bfd50179d6ac
rem   gpt attributes=0x8000000000000001
list volume
```

#### BIOS:

Use the attribute: `id=27` to define the system partition, and use the `remove` command to remove the drive letter.

```
rem == HideRecoveryPartitions-BIOS.txt
select disk 0
select partition 3
set id=27
remove
list volume
exit
```

2. Save your completed file as either E:\Recovery\HideRecoveryPartitions-UEFI.txt or E:\Recovery\HideRecoveryPartitions-BIOS.txt, based on your firmware type.

### Identify and hide the drive letters

- Run the diskpart script to identify and hide the recovery partitions:

```
Diskpart /s E:\Recovery\HideRecoveryPartitions-<firmware>.txt
```

Where <firmware> is either UEFI or BIOS.

### Verify that the Windows RE configuration is set correctly

- Open an administrative command prompt.

Verify the Windows RE information:

```
reagentc /info
```

Verify the following:

- Windows RE status is enabled.
- Windows RE location is on the correct partition.
- The BCD GUID entry for WinRE is the same as the WinRE GUID entry in the file: reagent.xml. On BIOS-based PCs, this file is on the system partition, at \Recovery\{GUID}\. On UEFI-based PCs, this file is on the Windows RE Tools partition, at \Recovery\WindowsRE\.
- WinRE is located in the \Recovery\WindowsRE directory

## Related topics

[Windows Recovery Environment \(Windows RE\) Technical Reference](#)

[DISM Image Management Command-Line Options](#)

[Customize Windows RE](#)

[Add a Custom Tool to the Windows RE Boot Options Menu](#)

# Push-button reset

1/24/2019 • 4 minutes to read

## TIP

If you're a user who wants to reset a computer that runs Windows 10, see [Recovery options in Windows 10](#).

This topic is intended for original equipment manufacturers (OEMs) who want to add push-button reset features to their Windows 10 desktop computer manufacturing processes.

Push-button reset is a recovery tool that repairs the OS while preserving data and important customizations. It reduces the need for custom recovery applications by providing users with more recovery options and the ability to fix their own PCs with confidence.

Push-button reset is included in Windows 10 for desktop editions (Home, Pro, Enterprise, and Education), and was introduced in Windows 8.

## What's new for Windows 10

In Windows 10, version 1809, Push-button reset has been updated to include the following change:

- **Auto-apply folders** make Push-button reset customization easier. Placing customizations and their related asset files in predetermined folders ensures they are restored when a device is reset. This method of configuring Push-button reset can be used instead of the legacy method that used custom scripts and extensibility points.
- **Preinstalled apps get restored to an updated state.** If a preinstalled app has been updated after installation, it will be restored in its updated state.

Previous versions of Windows 10 provided the following improvements to push-button reset:

- **Use default config files when using ScanState to capture customizations:** Starting with Windows 10 Version 1703, you have to use the `/config` option with ScanState when capturing customizations. Use only one of the default configuration files included with the Assessment and Deployment Kit (ADK). These files are:
  - **Config\_AppsAndSettings.xml** – Use this configuration file to specify that both desktop applications and OS settings should be captured by the ScanState tool.
  - **Config\_AppsOnly.xml** – Use this configuration file to specify that only desktop applications should be captured by the ScanState tool. Since desktop applications are not always well-defined, this configuration file does not guarantee that all setting related to desktop applications are captured.
  - **Config\_SettingsOnly.xml** – Use this configuration file to specify that only OS settings should be captured by the ScanState tool.

You can modify these configuration files by setting the `migrate` attribute for specific components to `no`, but components that are already excluded from capture/migration in the default configuration files must remain excluded.

- **Improved reliability:** When you start push-button reset features from the Settings app, Windows scans the system files in the Windows Component Store for corruptions. If it finds corrupt files and can download replacements through Windows Update, it fixes the problem automatically. Although this increases the

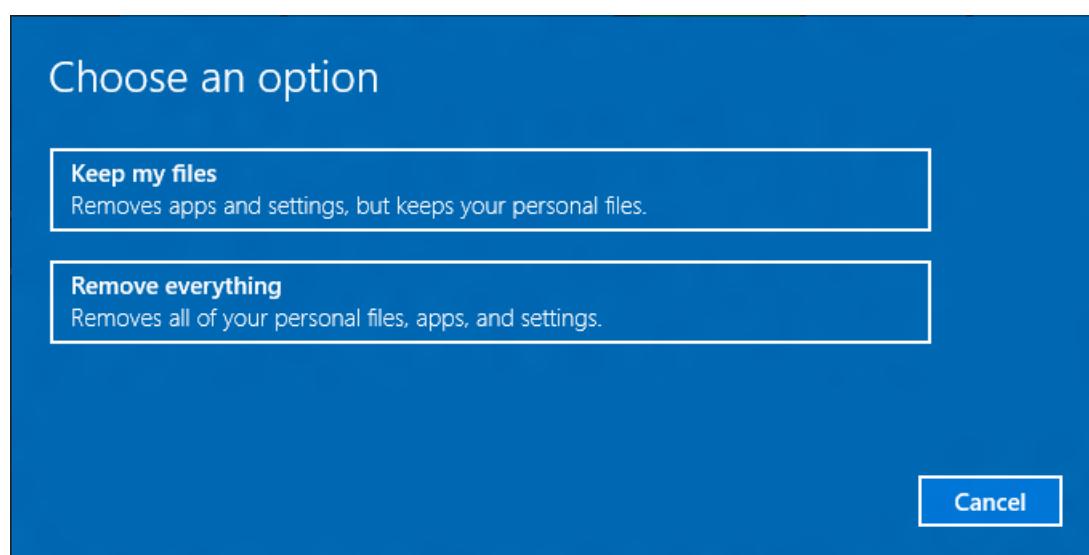
overall recovery time, it improves the reliability of the PC.

- **Recover from failed resets:** In Windows 10, Version 1507, and Windows 10, Version 1511, failures that occur during Reset this PC almost always rendered the PC unbootable/unrecoverable. This feature has been redesigned in the Anniversary Update to support limited rollback if a problem occurs while the PC is in Windows RE.
- **Recovery options when booted from recovery media:** When the PC is booted from recovery media, the **Keep my files** and **Remove everything** features are no longer supported. The only Push-button reset feature available when booted from media is bare metal recovery (i.e. Recover from a drive).
- **Image-less recovery:** Push-button reset no longer require or support a separate recovery image on a local partition or on media. This significantly reduces the disk space needed to support the features, and makes recovery possible even on devices with limited storage capacity.
- **Recovers to an updated state:** Push-button reset features now recover the Operating System (OS) and drivers (including device applets that are installed as part of INF-based driver packages) to an updated state. This reduces the amount of time users have to spend reinstalling the OS updates and drivers after performing a recovery.

The Push-button reset user experience continues to offer customization opportunities. Manufacturers use Auto-apply folder; or insert custom scripts, install applications or preserve additional data at available extensibility points.

The following Push-button reset features are available to users with Windows 10 PCs and devices:

- **Keep my files** Fixes software problems by reinstalling the OS while preserving the user data, user accounts, and important settings. All other preinstalled customizations are restored to their factory state. In Windows 10, this feature no longer preserves user-acquired Windows apps.
- **Remove everything** Prepares the PC for recycling or for transfer of ownership by reinstalling the OS, removing all user accounts and contents (e.g. data, Windows desktop applications, and Universal Windows apps), and restoring preinstalled customizations to their factory state.
- **Bare metal recovery** Restores the default or preconfigured partition layout on the system disk, and reinstalls the OS and preinstalled customizations from external media.



CONTENT TYPE	REFERENCES
<b>Overview</b>	<a href="#">How push-button reset features work</a>   <a href="#">Recovery strategy for common customizations</a>   <a href="#">Siloed provisioning packages</a>

CONTENT TYPE	REFERENCES
<b>Hard drive setup</b>	<a href="#">Hard Drives and Partitions</a>   <a href="#">UEFI/GPT-based hard drive partitions</a>   <a href="#">BIOS/MBR-based hard drive partitions</a>
<b>Operations</b>	<a href="#">Deploy push-button reset features using ScanState</a>   <a href="#">Deploy push-button reset features using Auto-Apply folders</a>   <a href="#">Add extensibility scripts to push-button reset</a>   <a href="#">Create a provisioning package with Windows desktop applications</a>   <a href="#">Validation scenarios</a>
<b>Bare metal reset/recovery</b>	<a href="#">How bare metal reset/recovery works</a>   <a href="#">Enable your users to create recovery media</a>   <a href="#">Create recovery media while deploying new devices</a>
<b>Configuration files</b>	<a href="#">ResetConfig XML reference</a>
<b>Technologies used by push-button reset</b>	<a href="#">Windows Recovery Environment</a>   <a href="#">Windows PE (WinPE)</a>   <a href="#">ScanState</a>

# How push-button reset features work

5/7/2019 • 7 minutes to read

## Restoring the operating system and customizations

This section discusses the mechanisms Push-button reset uses to restore software on the PC.

### Restoring Windows

Push-button reset restores Windows 10 by constructing a new copy of the OS using runtime system files located in the Windows Component Store (C:\Windows\WinSxS). This allows recovery to be possible even without a separate recovery image containing a backup copy of all system files.

In addition, Push-button reset restores Windows to an updated state rather than to the factory-preinstalled state. All updates installed on the PC (such as Windows 10, version 1809) will be restored. Due to this improvement it is not required nor recommended to mark updates as permanent by using the DISM /Cleanup-Image command with the /ResetBase option.

This approach provides a balance between user experience in terms of the number of updates which need to be reinstalled and the features' effectiveness in addressing update problems. It also allows Windows to remove older system files which are no longer needed for runtime use or for recovery, freeing up disk space.

### Restoring language packs

Language packs that are installed and used by at least one user account are restored. This includes languages installed by users.

Seven days after the Out-of-Box Experience (OOBE), any language packs that haven't yet been used are removed. Using Push-button reset features after that will not restore the removed language packs.

On PCs running single-language editions of Windows, such as Windows 10 Home, users cannot download or install additional language packs, and they cannot use push-button reset features to switch languages if the preinstalled language packs have been removed.

### Restoring drivers

Drivers are restored in a similar fashion as the OS. Instead of restoring them from a recovery image, existing drivers are preserved across recovery. Starting with Windows 10, version 1709, the latest drivers are restored as part of Push-button Reset.

Prior to Windows 10, version 1709, As with system files, drivers are restored to the state they were in when the most recent release or major update is installed. For example:

- If the customer performs recovery after booting up a new PC preinstalled with Windows 10, drivers that are present during OOBE will be restored, even if newer drivers have been installed since.
- If the customer performs recovery after upgrading from Windows 10 to Windows 10, version 1511, the drivers that are present during the upgrade will be restored, even if newer drivers have been installed since.

Device applets which are installed outside of the driver INF package are not restored as part of this process. They are restored to factory version and state in the same way as other customizations such as Windows desktop applications. (See Restoring other customizations for more information.) If the device applet must always stay in sync (version wise) with the driver, it is recommended that both the driver and the device applet be installed via the same INF package.

### Restoring previously installed Windows apps

Starting with Windows 10, version 1809, preinstalled Windows apps that have been updated after their initial installation get restored to their updated state. Prior to Windows 10, version 1809, preinstalled Windows apps get restored to their factory version and state. Instead of restoring them from a recovery image, a copy of the Windows apps is automatically backed up when they are provisioned during image customization and manufacturing, and the backups are restored when Push-button reset features are used.

### **Restoring other customizations**

To restore Windows desktop applications and settings, you can use provisioning packages created using the ScanState utility, and either push-button reset extensibility points or Auto-apply folders (Windows 10, version 1809 and later).

To learn more, see [Deploy push-button reset features using ScanState](#) and [Deploy push-button reset features using Auto-apply folders](#).

## Keep my files

The **Keep my files** feature preserves a number of system and user settings that are required to keep the system running while minimizing the need for users to reconfigure their PCs.

Preserved settings can be broadly categorized into one of the following categories:

- Are required for users to log on to their PCs after running the **Keep my files** feature.
- Affect how users access their documents and personal files.
- Are difficult for most users to recreate.
- Affect system security or user privacy.
- Personalize the PC.

### **Settings**

The preserved settings are summarized as follows:

- User accounts (local, domain, Microsoft account), and group memberships
- Domain settings
- Windows Update settings
- Library settings
- Lock screen background
- Desktop themes
- International settings
- Wireless network profiles
- Settings configured in Windows Welcome

### **User data**

Because user data can be stored in many locations, the **Keep my files** feature preserves most folders and files that are not part of a standard Windows installation. The **Keep my files** feature refreshes the following system locations and does not preserve the contents.

- \Windows
- \Program Files
- \Program Files(x86)
- \ProgramData
- \Users\<user name>\AppData (in each user profile)

**Note** Some applications store user data in the \AppData folder in user profiles. The \AppData folders are available in C:\Windows.old after using the **Keep my files** feature.

The **Keep my files** feature bypasses the following locations and preserves the contents:

- File History versioning data
- All files and folders on non-OS partitions

## Windows Applications

The **Keep my files** feature handles application types differently in order to ensure that the PC can be restored to a reliable state.

Applications are handled as follows:

- User-acquired Windows apps from the Microsoft Store are not preserved. Users will need to reinstall them from the Microsoft Store. This is a change from Windows 8/8.1.
- Starting with Windows 10, version 1809, preinstalled Windows apps that have been updated since initial installation will be restored to an updated state. Prior to Windows 10, version 1809, preinstalled Windows apps are restored to their factory version and state. Updates to these apps will be downloaded and reapplied automatically when internet connectivity is available.
- User-acquired Windows desktop applications are not preserved. Users will need to reinstall them manually.
- Preinstalled Windows desktop applications captured in the customizations provisioning package will be restored to their factory condition, even if users have previously uninstalled them.

The **Keep my files** feature does not preserve user-installed Windows desktop applications by default, and locations that are commonly used for storing application settings (\AppData and \ProgramData) are deleted. Manufacturers can leverage Auto-apply folders or the push-button reset extensibility points to save and later restore specific application settings and data, if necessary.

## Remove everything

When users use the **Remove everything** feature, they will be presented with options that affect the way that their data is removed from the PC.

- If the PC has more than one user-accessible hard drive volumes, users can choose to remove data from all volumes or only the Windows volume.

The Windows volume is never formatted, as the files needed to rebuild the OS are on it. Instead, user data files are deleted individually.

If user chooses to remove data from all volumes, the data volumes are formatted.

- Users can choose to simply delete their files or to also perform data erasure on the drive(s) so that recovery of the data by someone else is much more difficult.

Manufacturers must configure custom utility partitions as follows to ensure these partitions are not affected by the reset process.

- For UEFI-based PCs, utility partitions on GUID Partition Table (GPT) disks should have the GPT\_ATTRIBUTE\_PLATFORM\_REQUIRED attribute set. See [PARTITION\\_INFORMATION\\_GPT structure](#) for more information on GPT partition attributes.
- For BIOS-based PCs, utility partitions on Master Boot Record (MBR) disks must be of a type other than 0x7, 0x0c, 0x0b, 0x0e, 0x06, and 0x42.

The time it takes to perform data erasure depends on drive speed, partition size, and whether the drive is encrypted using Windows BitLocker Drive Encryption. The data erasure functionality is targeted at consumers and does not meet government and industry data erasure standards.

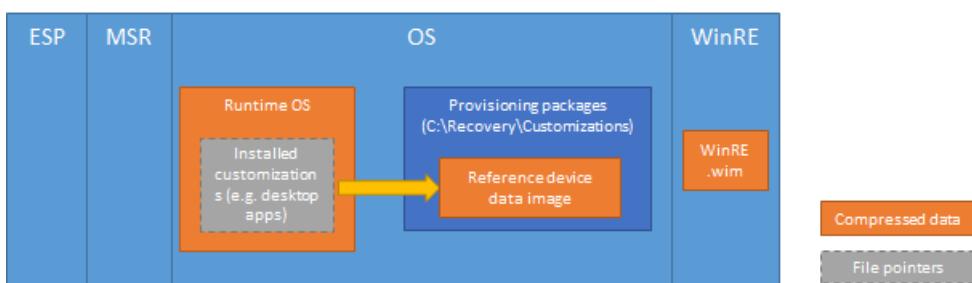
If [Compact OS](#) is enabled on the OS before the reset, Compact OS will remain enabled after the PC has been reset.

# Compact OS

Compact OS is a collection of technologies which allow Windows 10 to be deployed on PCs with storage capacity as low as 16 gigabytes (GB). The following two technologies in particular work in conjunction with the Push-button reset changes to reduce Windows' disk footprint:

- Per-file compression When applying a reference image file (WIM) to a PC, the files written to the disk can be compressed individually using the XPRESS Huffman codec. This is the same codec used by the WIMBoot technology in Windows 8.1. When Push-button reset features rebuilds the OS, the runtime system files remain compressed.
- Single-instancing of installed customizations After the installed customizations (e.g. Windows desktop applications) have been captured (using ScanState) into a reference device data image stored inside a provisioning package, the two copies of the customizations can be singled-instanced to reduce disk footprint impact. This is accomplished by converting the installed customizations (e.g. C:\Program Files\Foo\Foo.exe) into file pointers linked to the contents of the reference device data image.

The following diagram illustrates the high-level content layout of PCs with Compact OS enabled:



Both technologies are optional and can be configured during deployment.

## Related topics

[Bare metal recovery](#)

# Recovery components

1/18/2019 • 8 minutes to read

Push-button reset features by default restore only drivers (installed through INF packages) and preinstalled Windows apps. To configure the features to restore other customizations such as settings and Windows desktop applications, you will need to prepare one or more customization packages which contain the customizations. These customizations packages are in the form of provisioning packages (.ppkg).

Push-button reset looks for, and automatically restores provisioning packages which are located in the folder C:\Recovery\Customizations. **New in Windows 10, version 1809:** Windows will also look for and restore certain customizations in the C:\Recovery\AutoApply folder. If customizations are in this folder and extensibility points aren't configured, the customizations in this folder will be restored.

To protect the packages from tampering or accidental deletion, the Write/Modify permissions of C:\Recovery\Customizations should be restricted to the local Administrators user group.

Some settings and customizations cannot be included in provisioning packages. Instead, you can restore them using an unattend file applied using the Push-button reset extensibility points. For settings which are supported by both provisioning packages and unattend, it is recommended that you specify them using only one of the mechanisms, not both. To learn more, see [How push-button reset features work](#).

## Auto-Apply folders

**New in Windows 10, version 1809** Auto-apply folders make Push-button reset customizations easier to configure for the reset experience. This new method copies and applies the most common Windows customization files after the device is reset. This can help simplify the configuration process and eliminate commonly-made mistakes that result in a misconfigured device.

Auto-apply folders can't be used in conjunction with [extensibility points](#).

### IMPORTANT

If both extensibility points are configured and Auto-apply folders are present in C:\Recovery, the Auto-apply folders will be ignored.

### Auto-apply customizations

The following customizations are supported by Auto-apply folders:

- TaskbarLayoutModification.xml
- LayoutModification.xml
- OOBEx
- Unattend.xml
- Any required asset files

### Configure Auto-apply

To use Auto-apply folders, you have to copy configuration files and any related asset files C:\Recovery\AutoApply. Related asset files are files that the configuration files rely on, like an graphic that unattend will set as a wallpaper or .lnk files that are used by TaskbarLayoutModification.xml.

During a recovery, files in this AutoApply folder will get copied to the correct folders in the restored image. For example, when you place unattend.xml in the AutoApply folder, it will be copied to the C:\Windows\Panther folder

during the recovery process.

The following table shows the available customizations and where to copy the configuration and related asset files so that PBR can restore them to the restored OS:

CUSTOMIZATION	COPY CONFIGURATION TO:	COPY RELATED ASSETS TO:
TaskbarLayoutModification.xml	C:\Recovery\AutoApply\	C:\Recovery\AutoApply\Customizationfiles
LayoutModification.xml	C:\Recovery\AutoApply\	C:\Recovery\AutoApply\Customizationfiles
OOBE.xml	Copy %windir%\System32\OOBE\info and all its contents to C:\Recovery\AutoApply\OOBE	N/A - The copied OOBE\Info folder should include all the files to support OOBE
Unattend.xml	C:\Recovery\AutoApply\	C:\Recovery\AutoApply\CustomizationFiles

## Capturing Windows desktop applications using Windows User State Migration Tool (USMT)'s ScanState tool

The Windows User State Migration Tool (USMT) ScanState.exe has been updated in Windows 10 to support capturing Windows desktop applications applications. This functionality can be activated by specifying the `/apps` option.

When `/apps` is specified, ScanState uses a set of application discovery rules to determine what should be captured, and stores the output as a reference device data image inside a provisioning package. In general, the reference device data includes the following:

- Windows desktop applications installed using either Microsoft Windows Installer or other installers
- All files and folders outside of the Windows namespace (in other words, outside of \Windows, \Program Files, \Program Files (x86), \ProgramData, and \Users). This applies only to the volume on which Windows is installed.
- Not captured: Windows apps.
- Not captured: User state/data.

You can also specify additional rules to include or exclude specific files, folders, and registry settings. For example, if you are using ScanState during factory deployment, you might need to exclude manufacturing-specific tools so that they will not be restored when end users use Push-button reset features. To specify additional rules, you will need to author a migration XML and specify the `/i` option when using ScanState.exe.

ScanState's `/apps` option also supports the following optional parameters:

PARAMETER	USE

PARAMETER	USE
<code>+/-sysdrive</code>	<p>Specifies whether applications, files, and folders outside of the Windows namespace should be captured.</p> <p>If <code>+sysdrive</code> is specified, all contents on the system drive are examined and eligible to be captured according to the discovery rules.</p> <p>If <code>-sysdrive</code> is specified, only contents within the Windows namespace are examined and eligible to be captured according to the discovery rules.</p> <p><code>+sysdrive</code> is the default.</p>
<code>+/-oeminfo</code>	<p>Specifies whether the OEM-specific help and support info should be captured.</p> <p>If <code>+oeminfo</code> is specified, OEM and support info are captured.</p> <p>If <code>-oeminfo</code> is specified, OEM and support info are not captured.</p> <p><code>+oeminfo</code> is the default.</p>

## Important

- Although push-button reset features can restore multiple provisioning packages, only one of the packages can contain reference device data image captured using ScanState.
- ScanState should be used only after all customizations have been applied to the PC. It does not support appending additional changes to an existing reference device data image.
- When you prepare ScanState for capturing customizations, you should exclude Windows Defender settings to prevent possible failures during recovery that can be caused by file conflicts. For more information, see Step 1 in [Deploy push-button reset features](#).

## Restoring settings using unattend.xml and extensibility points

### TIP

**New in Windows 10, version 1809** You can use Auto-apply folders to automatically restore unattend.xml, layoutmodification.xml, and oobe.xml. If using Auto-apply folders, you don't have to configure extensibility scripts as outlined below.

Most settings which are configured using unattend.xml and other configuration files (e.g. oobe.xml, LayoutModification.xml) cannot be restored using provisioning packages. Instead, you will need to use the Push-button reset extensibility points in order to restore them during recovery. These extensibility points allow you run scripts which can:

- Inject an unattend.xml into the recovered OS
- Copy other configuration files and assets into the recovered OS

## Important

- You should not use unattend.xml (or other mechanisms) to boot the recovered OS into Audit Mode. The recovered OS must remain configured to boot to OOBE.
- A copy of the configuration files and assets which need to be restored must be placed under C:\Recovery\OEM.

Contents in this folder are not modified by push-button reset features and are automatically backed up to recovery media created using the **Create a recovery drive** utility. To protect the unattend.xml and configuration files/assets from tampering or accidental deletion, Write/Modify permissions of C:\Recovery\OEM should be restricted to the local Administrators user group.

To learn how to author scripts to be run using extensibility points, see [Add extensibility scripts to push-button reset](#).

To learn how to use ScanState to capture and store the resulting PPKG under C:\Recovery\Customizations, which is restored automatically during PBR, see [Deploy push-button reset features using ScanState](#).

## Recovery strategies for common customizations

The following table outlines the recovery strategy for common customizations which are described in the User Experience Windows Engineering Guide (UX WEG) as well as those covered in the OEM Policy Document (OPD). For up-to-date details on these customizations, refer to the latest version of the UX WEG and OPD.

CUSTOMIZATION	HOW IT IS CONFIGURED	HOW IT CAN BE RESTORED DURING PBR
OOBE – HID pairing	Settings in the <hidSetup> section of OOBExml and images (e.g. .png files)	<ul style="list-style-type: none"> <li>• Use PBR extensibility script to restore OOBExml and images from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
OOBE – OEM EULA	<Eulafilename> setting in OOBExml and license terms .rtf file(s) stored under %WINDIR%\System32\Oobe\Info	<ul style="list-style-type: none"> <li>• Use PBR extensibility script to restore OOBExml and .rtf files from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
OOBE – Preconfigured language and time zone	Settings in the <defaults> section of OOBExml	<ul style="list-style-type: none"> <li>• Use PBR extensibility script to restore OOBExml from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
OOBE – Hide mobile broadband page	Microsoft-Windows-WwanUI   NotInOOBE setting in unattend.xml	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore unattend.xml from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
OOBE – OEM Registration page	Settings in the <registration> section of OOBExml and HTML files for in-place links	<ul style="list-style-type: none"> <li>• Use PBR extensibility script to restore OOBExml and HTML files from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>

CUSTOMIZATION	HOW IT IS CONFIGURED	HOW IT CAN BE RESTORED DURING PBR
Start – Pinned tiles and groups	LayoutModification.xml stored under %SYSTEMDRIVE%\Users\Default\AppData\Local\Microsoft\Windows\Shell or settings under Microsoft-Windows-Shell-Setup   StartTiles in unattend.xml	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore LayoutModification.xml or unattend.xml from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
Start – Prepopulated MFU list	LayoutModification.xml stored under %SYSTEMDRIVE%\Users\Default\AppData\Local\Microsoft\Windows\Shell	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore LayoutModification.xml from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
Continuum – Form factor	Settings in unattend.xml: <ul style="list-style-type: none"> <li>• Microsoft-Windows-Deployment   DeviceForm</li> <li>• Microsoft-Windows-GPIOButtons   ConvertibleSlateMode</li> </ul>	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore unattend.xml from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
Continuum – Default mode	Microsoft-Windows-Shell-Setup   SignInMode setting in unattend.xml	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore unattend.xml from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
Desktop – Default and additional accent colors	RunSynchronous command in unattend.xml which adds the AGRB hex color values to the registry under HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Themes\Accents	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore unattend.xml from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
Desktop – Background image	Microsoft-Windows-Shell-Setup   Themes   DesktopBackground setting in unattend.xml and image (e.g. jpg/.png/.bmp file)	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore unattend.xml and background image file from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
Desktop – Pinned taskbar items	Settings under Microsoft-Windows-Shell-Setup   TaskbarLinks in unattend.xml and shortcut (.lnk) files stored in a folder under %ALLUSERSPROFILE%\Microsoft\Windows\Start Menu\Programs\	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore unattend.xml and .lnk files from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>

CUSTOMIZATION	HOW IT IS CONFIGURED	HOW IT CAN BE RESTORED DURING PBR
Desktop – Systray icons	Settings under Microsoft-Windows-Shell-Setup   NotificationArea in unattend.xml	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore unattend.xml from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
Mobile broadband – Rename "WiFi" to "WLAN" in network list	Microsoft-Windows-SystemSettings   WiFiToWlan setting in unattend.xml	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore unattend.xml from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
Mobile broadband – Enable Network Selection control in Settings	Microsoft-Windows-SystemSettings   DisplayNetworkSelection setting in unattend.xml	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore unattend.xml from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
PC Settings – Preinstalled settings apps	Settings apps are preinstalled in the same way as any other app, and automatically appear in Settings. Capability declared in the app manifest determines whether it is a settings app or not.	Restored automatically along with other preinstalled apps
Default browser and handlers of protocols	Default application association settings XML file imported using the /Import-DefaultAppAssociations command in DISM	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to <a href="#">Import Default App associations</a></li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Use an unattend file with a <a href="#">RunSynchronousCommand</a> to <a href="#">Import Default App associations</a>, adding the unattend file to the auto-apply folders</li> </ul>
Support information in Contact Support app	Settings under Microsoft-Windows-Shell-Setup   OEMInformation in unattend.xml and logo.bmp file	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore unattend.xml and .bmp file from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>
Store content modifier	Microsoft-Windows-Store-Client-UI   StoreContentModifier setting in unattend.xml	<ul style="list-style-type: none"> <li>• Use PBR extensibility points to restore unattend.xml from C:\Recovery\OEM</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• Auto-apply folders</li> </ul>

CUSTOMIZATION	HOW IT IS CONFIGURED	HOW IT CAN BE RESTORED DURING PBR
Windows desktop applications (including driver applets installed via setup.exe)	MSI or custom installers	Use ScanState to capture and store the resulting PPKG under C:\Recovery\Customizations, which is restored automatically during PBR.
RDX contents	See UX WEG for details	Should not be restored during PBR

# Deploy push-button reset features

1/18/2019 • 13 minutes to read

Push-button reset features are included with Windows 10 for desktop editions (Home, Pro, Enterprise, and Education), though you'll need to perform additional steps to deploy PCs with the following customizations.

- Windows desktop applications
- Windows settings, such as customized OOB screens or Start Menus.
- Customized partition layouts.

These steps also show you how to add your own scripts during a reset to capture logs or perform other cleanup tasks.

## Prerequisites

To complete these procedures, you'll need a technician PC which has Windows 10 and the following Windows Assessment and Deployment Kit (ADK) for Windows 10 components installed. Note that starting with Windows 10, version 1809, WinPE is an addon to the ADK, and isn't included in the ADK installer:

- Deployment Tools
- Imaging and Configuration Designer (ICD)
- User State Migration Tool (USMT)
- Windows Preinstallation Environment (Windows PE)

You'll also need:

- A destination PC with drive size of 100 GB or larger
- A Windows 10 for desktop editions image (install.wim)
- A Windows RE boot image (Winre.wim) (You'll extract this from a Windows 10 image).

For an overview of the entire deployment process, see the [Desktop manufacturing guide](#).

Use the follow steps to prepare the ScanState tool to capture Windows desktop applications after they have been installed:

## Step 1: Prepare the ScanState tool

1. On the technician PC, copy the Windows ADK files from Windows User State Migration Tool (USMT) and Windows Setup to a working folder. You'll need to match the architecture of the destination device. You don't need to copy the subfolders.

```
md C:\ScanState_amd64
xcopy /E "C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\User State Migration
Tool\amd64" C:\ScanState_amd64
xcopy /E /Y "C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows
Setup\amd64\Sources" C:\ScanState_amd64
```

2. Copy the contents of the working folder to a network location or USB flash drive.

Use the following steps to customize your Windows RE boot image if additional drivers and language packs are needed.

## Step 2: Extract and customize the Windows RE boot image (optional)

1. On the technician PC, click **Start**, and type deployment. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. In **Deployment and Imaging Tools Environment**, create the folder structure to store the Windows image and its mount point.

```
Mkdir C:\OS_image\mount
```

3. Create the folder structure to store the Windows RE boot image and its mount point.

```
Mkdir C:\winre_amd64\mount
```

4. Mount the Windows image (install.wim) to the folder \OS\_image\mount by using DISM.

```
Dism /mount-image /imagefile:C:\OS_image\install.wim /index:1 /mountdir:C:\OS_image\mount
```

where `Index:1` is the index of the selected image in the Install.wim file.

5. Copy the Windows RE image from the mounted Windows image to the new folder.

```
xcopy /H C:\OS_image\mount\windows\system32\recovery\winre.wim C:\winre_amd64
```

6. Unmount the Windows image. Tip: If you haven't made any other changes in the Windows image, you can unmount the image faster by using the `/discard` option.

```
Dism /unmount-image /mountdir:C:\OS_image\mount /discard
```

7. Mount the Windows RE boot image for editing.

```
Dism /mount-image /imagefile:C:\winre_amd64\winre.wim /index:1 /mountdir:C:\winre_amd64\mount
```

where `Index:1` is the number of the selected image in the Winre.wim file.

Once the Winre.wim file is extracted from the Install.wim file, you can customize the Windows RE boot image.

8. Add language packs, boot-critical device drivers, and input device drivers to the Windows RE boot image.  
To learn more, see [Customize Windows RE](#).

9. Commit your customizations and unmount the image.

```
Dism /unmount-image /mountdir:C:\winre_amd64\mount /commit
```

If you are planning to customize only the settings common to all editions of Windows 10 (including Windows 10 Mobile), use the following steps to create a provisioning package which specifies settings to be restored during recovery:

## Step 3: Create a provisioning package with settings to be restored

## (optional)

1. On the technician PC, start Windows Imaging and Configuration Designer (ICD).
2. Click **File > New Project**.
3. Enter a project name and description, and then click **Next**
4. In the **Select project workflow** step, select the **Provisioning Package** option, and then click **Next**.
5. In the **Choose which settings to view and configure** step, select the **Common to all Windows editions** option, and then click **Next**.
6. In the **Import a provisioning package (optional)** step, click **Finish** to create the new project.
7. Use the **Available customizations** pane to add settings and specify the defaults which should be restored during recovery. The settings will appear in the **Selected customizations** pane.
8. Click **Export > Provisioning package**.
9. In the **Describe the provisioning package** step, click **Next**.
10. In the **Select the security details for the provisioning package** step, click **Next**.
11. In the **Select where to save the provisioning package** step, enter a location to save the package (such as a network share) and then click **Next**.
12. Click **Build** to create the provisioning package.
13. After the provisioning package is created, click **Finish**.

If your customizations include settings specific to editions of Windows 10 for desktop editions, use the following steps to create an unattend.xml which specifies the settings to be restored during recovery:

## Step 4: Create an unattend file to restore settings (optional)

1. On the technician PC, start **Windows System Image Manager**.
2. Click **File > Select Windows image**.
3. When prompted to create a catalog file, click **Yes**.
4. Use the **Windows Image** and **Answer File** panes to add settings to the Specialize or oobeSystem phase (or both), and specify the defaults which should be restored during recovery.
5. Click **Tool > Validate Answer File** to check for errors. Correct any problem identified.
6. Click **File > Save Answer File**. Enter a location to save the answer file (such as a network share) and then click **Save**.

If you plan to use Push-button reset's extensibility points, use the following steps to prepare your extensibility points and register them using a Push-button reset configuration file.

**Important** If you have created an unattend file, you must also create a script to reapply it using the BasicReset\_AfterImageApply and FactoryReset\_AfterImageApply extensibility points.

## Step 5: Prepare push-button reset extensibility point scripts (optional)

1. Create scripts (.cmd) or executables (.exe) to run at the available extensibility points when the Keep my files feature runs:
  - A: At BasicReset\_BeforeImageApply
  - B: At BasicReset\_AfterImageApply
2. Create scripts (.cmd) or executables (.exe) to run at the available extensibility points when the Remove everything feature runs:
  - C: At FactoryReset\_AfterDiskFormat
  - D: At FactoryReset\_AfterImageApply

3. Save the scripts to a network location, or USB flash drive.
4. Create a ResetConfig.xml file that specifies the location of the scripts that you created for the four extensibility points. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<Reset>
    <Run Phase="BasicReset_BeforeImageApply">
        <Path>Fabrikam\SampleScript_A.cmd</Path>
        <Duration>2</Duration>
    </Run>
    <Run Phase="BasicReset_AfterImageApply">
        <Path>Fabrikam\SampleScript_B.cmd</Path>
        <Param></Param>
        <Duration>2</Duration>
    </Run>
    <Run Phase="FactoryReset_AfterDiskFormat">
        <Path>Fabrikam\SampleScript_C.cmd</Path>
        <Duration>2</Duration>
    </Run>
    <Run Phase="FactoryReset_AfterImageApply">
        <Path>Fabrikam\SampleScript_D.cmd</Path>
        <Param></Param>
        <Duration>2</Duration>
    </Run>
</Reset>
```

**Important** If you use a text editor to author the ResetConfig.xml file, save the document with an .xml file name extension and use **UTF-8 encoding**. Do not use Unicode or ANSI.

5. Save the ResetConfig.xml file together with the scripts that you created.

## Step 6: Create bare-metal recovery configuration (optional)

- To specify the partition layout to be used when users perform bare metal recovery using recovery media created from their PCs, modify resetconfig.xml to include the following elements:

```
<?xml version="1.0" encoding="utf-8"?>
<Reset>
    <SystemDisk>
        <MinSize>160000</MinSize>
        <DiskpartScriptPath>ReCreatePartitions.txt</DiskpartScriptPath>
        <OSPartition>3</OSPartition>
        <WindowsREPartition>4</WindowsREPartition>
        <WindowsREPath>Recovery\WindowsRE</WindowsREPath>
        <Compact>False</Compact>
    </SystemDisk>
</Reset>
```

- **MinSize** - Specifies the minimum size of the system disk in megabytes (MB). Recovery process will not proceed if the system disk does not meet this minimum size.
- **DiskpartScriptPath** - Path to Diskpart script relative to install.wim location. The script should assume that all existing partitions have been deleted, and the system disk has focus in Diskpart.
- **OSPartition** - The partition to which the recovery image should be applied must be specified. The ESP or active partition must be on the same disk as the OS.
- **WindowsREPartition; WindowsREPath** (Optional) The location in which WinRE should be staged. The WinRE boot image on the media will be copied and registered with the OS. (Same as running "reagentc.exe /setreimage")

If partitioning information is not specified in resetconfig.xml, users can still perform bare metal recovery

using media they have created. However, the default/recommended partition layout for Windows 10 will be used instead.

## Step 7: Create a diskpart script for initial deployment

1. Create a disk partitioning script for initial deployment.

### **UEFI example:**

```
rem These commands are used with DiskPart tool.
rem Erase the drive and create four partitions
rem for a UEFI/GPT-based PC.
select disk 0
clean
convert gpt
rem == 1. System Partition =====
create partition efi size=100
rem ***NOTE: For 4KB-per-sector drives, change
rem this value to size=260.***
format quick fs=fat32 label="System"
assign letter="S"
rem == 2. Microsoft Reserved (MSR) Partition ====
create partition msr size=16
rem == 3. Windows Partition =====
rem == a. Create Windows Partition =====
create partition primary
rem == b. Create space for Windows RE tools partition
shrink minimum=450
rem == c. Prepare the Windows partition
format quick fs=ntfs label="Windows"
assign letter="W"
rem == 4. Windows RE Tools Partition =====
create partition primary
format quick fs=ntfs label="Windows RE tools"
set id=de94bba4-06d1-4d40-a16a-bfd50179d6ac
assign letter="T"
exit
```

### **BIOS example:**

```

rem These commands are used with DiskPart to
rem erase the drive and create three partitions
rem for a BIOS/MBR-based PC.
rem Adjust the partition sizes to fill the drive.
select disk 0
clean
rem === 1. System Partition =====
create partition primary size=100
format quick fs=ntfs label="System"
assign letter="S"
active
rem === 2. Windows Partition =====
rem == a. Create Windows partition =====
create partition primary
rem == b. Create space for Windows RE tools partition ====
shrink minimum=450
rem == c. Prepare the Windows partition =====
format quick fs=ntfs label="Windows"
assign letter="W"
rem === 3. Windows RE Tools Partition =====
create partition primary
format quick fs=ntfs label="Windows RE tools"
set id=27
assign letter="R"
exit

```

2. Name the script CreatePartitions-UEFI or CreatePartitions-BIOS.txt, and save it to a network location, or USB flash drive. Note: In these Diskpart examples, the partitions are assigned the letters S:\, W:\, and T:\ to simplify partition identification. After the PC reboots, Windows PE automatically assigns the letter C:\ to the Windows partition. The other partitions do not receive drive letters.

## Step 8: Create a diskpart script for bare-metal recovery (optional)

1. Create a diskpart script for bare-metal recovery.

**Important** The diskpart script used for bare metal recovery should not include a `select disk` or `clean` command. The system disk will be selected automatically before the diskpart script is processed.

**UEFI example:**

```

rem These commands are used with DiskPart tool.
rem Erase the drive and create five partitions
rem for a UEFI/GPT-based PC.
convert gpt
rem == 1. System Partition =====
create partition efi size=100
rem ***NOTE: For 4KB-per-sector drives, change
rem this value to size=260.***
format quick fs=fat32 label="System"
assign letter="S"
rem == 2. Microsoft Reserved (MSR) Partition =====
create partition msr size=16
rem == 3. Windows Partition =====
rem == a. Create Windows Partition =====
create partition primary
rem == b. Create space for Windows RE tools partition
shrink minimum=450
rem == c. Prepare the Windows partition
format quick fs=ntfs label="Windows"
assign letter="W"
rem == 4. Windows RE Tools Partition =====
create partition primary
format quick fs=ntfs label="Windows RE tools"
set id=de94bba4-06d1-4d40-a16a-bfd50179d6ac
assign letter="T"
exit

```

### **BIOS example:**

```

rem These commands are used with DiskPart to
rem erase the drive and create three partitions
rem for a BIOS/MBR-based PC.
rem Adjust the partition sizes to fill the drive.
rem == 1. System Partition =====
create partition primary size=100
format quick fs=ntfs label="System"
assign letter="S"
active
rem == 2. Windows Partition =====
rem == a. Create Windows partition =====
create partition primary
rem == b. Create space for Windows RE tools partition ====
shrink minimum=450
rem == c. Prepare the Windows partition =====
format quick fs=ntfs label="Windows"
assign letter="W"
rem == 3. Windows RE Tools Partition =====
create partition primary
format quick fs=ntfs label="Windows RE tools"
set id=27
assign letter="R"
exit

```

2. Name the script RecreatePartitions-UEFI.txt or RecreatePartitions-BIOS.txt, and save it to the same network location, or USB flash drive as create partitions.

## Step 9: Deploy and customize Windows

1. On the destination PC, boot to Windows PE.
2. At the Windows PE command prompt, run the script to create the recommended hard drive partitions.

```
Diskpart /s N:\CreatePartitions.txt
```

where N:\CreatePartition is the location of the file.

3. Apply the Windows reference image to the Windows partition.

```
Dism /Apply-Image /ImageFile:N:\Install.wim /Index:1 /ApplyDir:W:\
```

Optional: You can also specify the /compact option so that the files written to disk are compressed. For example:

```
Dism /Apply-Image /ImageFile:N:\Install.wim /Index:1 /ApplyDir:W:\ /Compact:on
```

This is useful if you are deploying Windows onto PCs with limited storage capacity, but is not recommended on PCs with rotational storage devices.

4. Configure the system partition by using BCDboot.

```
W:\Windows\System32\Bcdboot W:\Windows
```

5. Create a folder in the Windows RE tools partition, and copy your custom Windows RE boot image to it.

```
Mkdir T:\Recovery\WindowsRE  
xcopy /H N:\Winre.wim T:\Recovery\WindowsRE
```

where T:\ is the Windows RE tools partition.

**Important** You must store Winre.wim in \Recovery\WindowsRE.

6. Register the Windows RE boot image together with the Windows image.

```
W:\Windows\System32\Reagents /setreimage /path T:\Recovery\WindowsRE /target W:\Windows
```

7. Use Diskpart to conceal the Windows RE tools (T:\) partition from Windows Explorer.

**For UEFI-based PCs:**

```
select disk 0  
select partition 4  
remove  
set id=de94bba4-06d1-4d40-a16a-bfd50179d6ac  
gpt attributes=0x8000000000000001  
exit
```

**For BIOS-based PCs:**

```
select disk 0  
select partition 3  
remove  
set id=27  
exit
```

8. Customize the Windows image on the destination PC:

- a. Perform offline customizations to the Windows image, such as installing INF-based driver packages specific to the destination PC, installing OS updates and language packs, or provisioning additional Windows apps.
  - b. Boot the destination PC to audit mode. This can be accomplished by using an answer file with the Microsoft-Windows-Deployment | Reseal | Mode = audit setting, or by first booting the PC to OOBE, and then pressing CTRL+SHIFT+F3.
  - c. Perform any remaining customizations such as installing applications and device software packages that are specific to the destination PC.
9. Clean up the image:

```
DISM.exe /Cleanup-Image /StartComponentCleanup
```

## Step 10: Capture and deploy customizations for recovery

1. Use the ScanState tool to capture the installed customizations into a provisioning package. Use the /config option to specify one of the default configuration files included with the ADK, and save the .ppkg file in the folder C:\Recovery\Customizations.

```
N:\ScanState_amd64\scanstate.exe /apps /config:<path_to_config_file> /ppkg  
C:\Recovery\Customizations\apps.ppkg /o /c /v:13 /l:C:\ScanState.log
```

where N:\ is the location of the ScanState tool installed in Step 1.

2. If you have used Windows ICD to create additional provisioning packages with customizations which should be restored during recovery, copy the packages to the destination PC. For example:

```
xcopy N:\RecoveryPPKG\*.ppkg C:\Recovery\Customizations
```

where N:\ is the location where the additional provisioning packages are located.

3. Copy any Push-button reset configuration file (resetconfig.xml) and extensibility scripts to the destination PC, and then configure permissions to write/modify them. For example:

```
mkdir C:\Recovery\OEM  
xcopy /E N:\RecoveryScripts\* C:\Recovery\OEM
```

where N:\ is the location where the configuration file and scripts are located.

4. Restrict the Write/Modify permissions of the customizations, and hide the root folder. For example:

```
icacls C:\Recovery\Customizations /inheritance:r /T  
icacls C:\Recovery\Customizations /grant:r SYSTEM:(F) /T  
icacls C:\Recovery\Customizations / grant:r *S-1-5-32-544:(F) /T  
icacls C:\Recovery\OEM /inheritance:r /T  
icacls C:\Recovery\OEM /grant:r SYSTEM:(F) /T  
icacls C:\Recovery\OEM / grant:r *S-1-5-32-544:(F) /T  
attrib +H C:\Recovery
```

5. Use the Sysprep tool to reseal the Windows image without using the /generalize option.

```
Sysprep /oobe /exit
```

**Note** Important: You must configure the image that you are shipping to the customer to boot to OOBE.

6. (Optional) To save space, you can also convert your installed Windows desktop applications into file pointers referencing the customizations package. To do so, boot the destination PC to Windows PE and run the following:

```
DISM /Apply-CustomDataImage /CustomDataImage:C:\Recovery\Customizations\USMT.ppkg /ImagePath:C:\  
/SingleInstance
```

7. Shut down the destination PC for packaging and shipment. When the user starts the PC for the first time, it will boot to OOBE.

## Step 11: Verify your customizations

Verify that your customizations are restored after recovery, and that they continue to function by running the Keep my files and Remove everything features. To start, check the following entry points:

- **Settings:** From the Start Menu, select **Settings > Update & security > Recovery** > Reset this PC: **Get started**. Follow the on-screen instructions.
- **Windows RE:** From the Start Menu, select **Settings > Update & security > Recovery** > Advanced startup: **Restart now**. After Windows RE boots, select **Troubleshoot > Reset this PC** and then follow the on-screen instructions.

To test more recovery options, see [validation scenarios](#).

## Related topics

[ScanState Syntax](#)

[Bare metal reset/recovery: Create recovery media while deploying new devices](#)

[Deploy push-button reset features using ScanState](#)

# Deploy push-button reset features using Auto-apply folders

1/18/2019 • 8 minutes to read

Push-button reset features are included with Windows 10 for desktop editions (Home, Pro, Enterprise, and Education), though you'll need to perform additional steps to deploy PCs with the following customizations.

- Windows desktop applications
- Windows settings, such as customized OOBE screens or Start Menus.
- Customized partition layouts.

These steps also show you how to add your own scripts during a reset to capture logs or perform other cleanup tasks.

## Prerequisites

To complete these procedures, you'll need a technician PC which has Windows 10 and the following Windows Assessment and Deployment Kit (ADK) for Windows 10 components installed. Note that starting with Windows 10, version 1809, WinPE is an addon to the ADK, and isn't included in the ADK installer:

- Deployment Tools
- Imaging and Configuration Designer (ICD)
- User State Migration Tool (USMT)
- Windows Preinstallation Environment (Windows PE)

You'll also need:

- A destination PC with drive size of 100 GB or larger
- A Windows 10 for desktop editions image (install.wim)
- A Windows RE boot image (Winre.wim) (You'll extract this from a Windows 10 image).

For an overview of the entire deployment process, see the [Desktop manufacturing guide](#).

Use the follow steps to prepare the ScanState tool to capture Windows desktop applications after they have been installed:

## Step 1: Prepare the ScanState tool

1. On the technician PC, copy the Windows ADK files from Windows User State Migration Tool (USMT) and Windows Setup to a working folder. You'll need to match the architecture of the destination device. You don't need to copy the subfolders.

```
md C:\ScanState_amd64
xcopy /E "C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\User State Migration Tool\amd64" C:\ScanState_amd64
xcopy /E /Y "C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Setup\amd64\Sources" C:\ScanState_amd64
```

2. Copy the contents of the working folder to a network location or USB flash drive.

Use the following steps to customize your Windows RE boot image if additional drivers and language packs are needed.

## Step 2: Extract and customize the Windows RE boot image (optional)

1. On the technician PC, click **Start**, and type deployment. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. In **Deployment and Imaging Tools Environment**, create the folder structure to store the Windows image and its mount point.

```
Mkdir C:\OS_image\mount
```

3. Create the folder structure to store the Windows RE boot image and its mount point.

```
Mkdir C:\winre_amd64\mount
```

4. Mount the Windows image (install.wim) to the folder \OS\_image\mount by using DISM.

```
Dism /mount-image /imagefile:C:\OS_image\install.wim /index:1 /mountdir:C:\OS_image\mount
```

where `Index:1` is the index of the selected image in the Install.wim file.

5. Copy the Windows RE image from the mounted Windows image to the new folder.

```
xcopy /H C:\OS_image\mount\windows\system32\recovery\winre.wim C:\winre_amd64
```

6. Unmount the Windows image. Tip: If you haven't made any other changes in the Windows image, you can unmount the image faster by using the `/discard` option.

```
Dism /unmount-image /mountdir:C:\OS_image\mount /discard
```

7. Mount the Windows RE boot image for editing.

```
Dism /mount-image /imagefile:C:\winre_amd64\winre.wim /index:1 /mountdir:C:\winre_amd64\mount
```

where `Index:1` is the number of the selected image in the Winre.wim file.

Once the Winre.wim file is extracted from the Install.wim file, you can customize the Windows RE boot image.

8. Add language packs, boot-critical device drivers, and input device drivers to the Windows RE boot image. To learn more, see [Customize Windows RE](#).

9. Commit your customizations and unmount the image.

```
Dism /unmount-image /mountdir:C:\winre_amd64\mount /commit
```

If you are planning to customize only the settings common to all editions of Windows 10 (including Windows 10 Mobile), use the following steps to create a provisioning package which specifies settings to be restored during recovery:

### Step 3: Create a provisioning package with settings to be restored (optional)

1. On the technician PC, start Windows Imaging and Configuration Designer (ICD).
2. Click **File > New Project**.
3. Enter a project name and description, and then click **Next**.
4. In the **Select project workflow** step, select the **Provisioning Package** option, and then click **Next**.
5. In the **Choose which settings to view and configure** step, select the **Common to all Windows editions** option, and then click **Next**.
6. In the **Import a provisioning package (optional)** step, click **Finish** to create the new project.
7. Use the **Available customizations** pane to add settings and specify the defaults which should be restored during recovery. The settings will appear in the **Selected customizations** pane.
8. Click **Export > Provisioning package**.
9. In the **Describe the provisioning package** step, click **Next**.
10. In the **Select the security details for the provisioning package** step, click **Next**.
11. In the **Select where to save the provisioning package** step, enter a location to save the package (such as a network share) and then click **Next**.
12. Click **Build** to create the provisioning package.
13. After the provisioning package is created, click **Finish**.

If your customizations include settings specific to editions of Windows 10 for desktop editions, use the following steps to create an unattend.xml which specifies the settings to be restored during recovery:

### Step 4: Copy configuration files and related asset files into Auto-apply folders

Auto-apply folders are new in Windows 10, version 1809. These folders make it easier for you to restore common settings, including unattend, OOBED, and taskbar layout during PBR. You can use Auto-apply folders or extensibility points, but not both. If Auto-apply folders and extensibility points are configured, Windows will use the extensibility points.

1. Create a folder in your Windows image called `C:\Recovery\AutoApply`

```
Mkdir C:\Recovery\AutoApply
```

2. Copy configuration files and any related asset files into the Autoapply folders:

DESCRIPTION	FILES TO BE COPIED	WHERE TO PUT IT	DURING A RESTORE, WHERE DOES IT GO?
Start menu	LayoutModification.xml	C:\Recovery\AutoApply\	%SYSTEMDRIVE%\Users\Default\AppData\Local\

DESCRIPTION	FILES TO BE COPIED	WHERE TO PUT IT	DURING A RESTORE, WHERE DOES IT GO?
Taskbar pins	TaskbarLayoutModification.xml	C:\Recovery\AutoApply\	C:\Windows\OEM\TaskbarLayoutModification.xml
OOBE.xml	%windir%\System32\OOBE\info	C:\Recovery\AutoApply\OOBE	%windir%\System32\OOBE\info
Unattend file	unattend.xml	C:\Recovery\AutoApply\	C:\Windows\Panther\Unattend.xml
Other asset files		C:\Recovery\AutoApply\CustomizationFiles	C:\Windows\OEM\CustomizationFiles

Note, don't worry about restoring link (.lnk) files used by the Start menu and Taskbar. These are saved and restored using [provisioning packages](#).

## Step 5: Deploy and customize Windows

1. On the destination PC, boot to Windows PE.
2. At the Windows PE command prompt, run the script to create the recommended hard drive partitions.

```
Diskpart /s N:\CreatePartitions.txt
```

where N:\CreatePartition is the location of the file.

3. Apply the Windows reference image to the Windows partition.

```
Dism /Apply-Image /ImageFile:N:\Install.wim /Index:1 /ApplyDir:W:\
```

Optional: You can also specify the /compact option so that the files written to disk are compressed. For example:

```
Dism /Apply-Image /ImageFile:N:\Install.wim /Index:1 /ApplyDir:W:\ /Compact:on
```

This is useful if you are deploying Windows onto PCs with limited storage capacity, but is not recommended on PCs with rotational storage devices.

4. Configure the system partition by using BCDboot.

```
W:\Windows\System32\Bcdboot W:\Windows
```

5. Create a folder in the Windows RE tools partition, and copy your custom Windows RE boot image to it.

```
Mkdir T:\Recovery\WindowsRE
xcopy /H N:\Winre.wim T:\Recovery\WindowsRE
```

where T:\ is the Windows RE tools partition.

### IMPORTANT

You must store Winre.wim in \Recovery\WindowsRE.

6. Register the Windows RE boot image together with the Windows image.

```
W:\Windows\System32\Reagentc /setreimage /path T:\Recovery\WindowsRE /target W:\Windows
```

7. Use Diskpart to conceal the Windows RE tools (T:\) partition from Windows Explorer.

### For UEFI-based PCs:

```
select disk 0
select partition 4
remove
set id=de94bba4-06d1-4d40-a16a-bfd50179d6ac
gpt attributes=0x8000000000000001
exit
```

### For BIOS-based PCs:

```
select disk 0
select partition 3
remove
set id=27
exit
```

8. Customize the Windows image on the destination PC:

- a. Perform offline customizations to the Windows image, such as installing INF-based driver packages specific to the destination PC, installing OS updates and language packs, or provisioning additional Windows apps.
  - b. Boot the destination PC to audit mode. This can be accomplished by using an answer file with the Microsoft-Windows-Deployment | Reseal | Mode = audit setting, or by first booting the PC to OOBE, and then pressing CTRL+SHIFT+F3.
  - c. Perform any remaining customizations such as installing applications and device software packages that are specific to the destination PC.
9. If you have installed OS updates, clean up the superseded components and mark the updates as permanent so that they will be restored during recovery:

```
DISM.exe /Cleanup-Image /StartComponentCleanup
```

## Step 6: Capture and deploy customizations for recovery

1. Use the ScanState tool to capture the installed customizations into a provisioning package. Use the /config option to specify one of the default configuration files included with the ADK, and save the .ppkg file in the folder C:\Recovery\Customizations.

```
N:\ScanState_amd64\scanstate.exe /apps /config:<path_to_config_file> /ppkg C:\Recovery\Customizations\apps.ppkg /o /c /v:13
/l:C:\ScanState.log
```

where N:\ is the location of the ScanState tool installed in Step 1.

2. If you have used Windows ICD to create additional provisioning packages with customizations which should be restored during recovery, copy the packages to the destination PC. For example:

```
xcopy N:\RecoveryPPKG\*.ppkg C:\Recovery\Customizations
```

where N:\ is the location where the additional provisioning packages are located.

3. Copy any Push-button reset configuration file (resetconfig.xml) to the destination PC, and then configure permissions to write/modify them. For example:

```
mkdir C:\Recovery\OEM
```

where N:\ is the location where the configuration file and scripts are located.

4. Restrict the Write/Modify permissions of the customizations, and hide the root folder. For example:

```
icacls C:\Recovery\Customizations /inheritance:r /T
icacls C:\Recovery\Customizations /grant:r SYSTEM:(F) /T
icacls C:\Recovery\Customizations / grant:r *S-1-5-32-544:(F) /T
icacls C:\Recovery\OEM /inheritance:r /T
icacls C:\Recovery\OEM /grant:r SYSTEM:(F) /T
icacls C:\Recovery\OEM /grant:r *S-1-5-32-544:(F) /T
icacls C:\Recovery\AutoApply /inheritance:r /T
icacls C:\Recovery\AutoApply /grant:r SYSTEM:(F) /T
icacls C:\Recovery\AutoApply /grant:r *S-1-5-32-544:(F) /T
attrib +H C:\Recovery
```

5. Use the Sysprep tool to reseal the Windows image without using the /generalize option.

```
Sysprep /oobe /exit
```

**Note** Important: You must configure the image that you are shipping to the customer to boot to OOBE.

6. (Optional) To save space, you can also convert your installed Windows desktop applications into file pointers referencing the customizations package. To do so, boot the destination PC to Windows PE and run the following:

```
DISM /Apply-CustomDataImage /CustomDataImage:C:\Recovery\Customizations\USMT.ppkg /ImagePath:C:\ /SingleInstance
```

7. Shut down the destination PC for packaging and shipment. When the user starts the PC for the first time, it will boot to OOBE.

## Step 7: Verify your customizations

Verify that your customizations are restored after recovery, and that they continue to function by running the Keep my files and Remove everything features. To start, check the following entry points:

- **Settings:** From the Start Menu, select **Settings > Update & security > Recovery** > Reset this PC: **Get started**. Follow the on-screen instructions.
- **Windows RE:** From the Start Menu, select **Settings > Update & security > Recovery** > Advanced startup: **Restart now**. After Windows RE boots, select **Troubleshoot > Reset this PC** and then follow the on-screen instructions.

To test more recovery options, see [validation scenarios](#).

## Related topics

[ScanState Syntax](#)

[Bare metal reset/recovery: Create recovery media while deploying new devices](#)

[Deploy push-button reset features using ScanState](#)

# Add extensibility scripts to push-button reset

1/18/2019 • 8 minutes to read

OEMs can insert custom extensibility scripts that run when a user runs the push-button reset features: **Keep my files** and **Remove everything**.

You can use either extensibility scripts or [Auto-apply folders](#) to restore common customizations that aren't otherwise restored, including:

- The Start Menu
- Taskbar
- OOBE
- Unattend.xml customizations

In addition, extensibility scripts can help perform other tasks, like:

- Modifying data or utility partitions
- [Saving and restoring files](#) that aren't normally kept by the **Keep my files** feature.

Note: If you include Auto-apply folders, you shouldn't include extensibility scripts. If you include both Auto-apply folders and extensibilty scripts, the Auto-apply folders will be ignored.

To **configure the scripts**, add all of the following in the `C:\Recovery\OEM` folder:

- A push-button reset configuration file (`ResetConfig.xml`) that defines which scripts to run.
- The extensibility scripts
- Any files required by the extensibility scripts.

## Extensibility scripts

### Requirements:

- The scripts are formatted as a .cmd or .exe files.
- The scripts do not depend on Windows PE optional components not present in the default Windows RE image (`winre.wim`).
- The scripts do not depend on binaries (e.g. .exe or .dll files) not present in the default Windows RE image (`winre.wim`).
- The scripts run without displaying a graphical user interface (GUI).
- The scripts complete all intended functions within 5 minutes for each extensibility point.
- The script must not modify the drive letters. This can potentially cause the recovery to fail.

The script must return a 0 (zero) if successful. If push-button reset receives a non-0 value, the following steps occur:

- If running the **Keep my files** feature: All system changes are rolled back. If the script or executable file is initiated from the Windows **PC settings** menu, the system reboots in Windows. If the script or executable file is initiated from Windows RE or the **Boot Options** menu, the system remains in Windows RE and displays an error message.
- If running the **Remove everything** feature: The failure is ignored. The script or executable file proceeds to the next step in the reset process and logs the failure.

## Push-button reset configuration file (ResetConfig.xml)

Add a [ResetConfig.xml](#) file to point to your push-button reset extensibility scripts.

This file must be saved with the file type of **UTF-8**. Do not use ANSI coding. For example: in Notepad, click **File**, and then click **Save As**. In the **Encoding** box, select **UTF-8**.

Save this file and copy it into the Windows images as `C:\Recovery\OEM\ResetConfig.xml`.

You can use the same ResetConfig.xml file to configure Windows to create recovery media. For more information, see [Deploy Push-Button Reset Features](#).

There's four [extensibility points](#) that you can use to point to scripts that run near the beginning and end of the **Keep my files** or **Remove everything** operations. For common customizations, you usually only need a single script, as shown in the [sample script](#) below.

## Sample script: restore the Start Menu, Taskbar, OOBE, and unattend.xml customizations

Save the following into the `C:\Recovery\OEM` folder:

- The sample script, **CommonCustomizations.cmd**
- The push-button reset configuration file, **ResetConfig.xml**
- A copy of the Start menu configuration file (**LayoutModification.xml**)
- A copy of the Taskbar configuration file (**TaskbarLayoutModification.xml**)
- A copy of the **unattend.xml** file

Save the following into the `C:\Recovery\OEM\OOBE\Info` folder:

- A copy the entire **OOBE** folder, `%WINDIR%\System32\Oobe\Info\`.

### CommonCustomizations.cmd

```
rem CommonCustomizations.cmd

rem Define %TARGETOS% as the Windows folder (This later becomes C:\Windows)
for /F "tokens=1,2,3 delims= " %%A in ('reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RecoveryEnvironment" /v TargetOS') DO SET TARGETOS=%%C

rem Define %TARGETOSDRIVE% as the Windows partition (This later becomes C:)
for /F "tokens=1 delims=\" %%A in ('Echo %TARGETOS%') DO SET TARGETOSDRIVE=%%A

rem Add back Windows settings, Start menu, Taskbar, and OOBE.xml customizations
copy "%TARGETOSDRIVE%\Recovery\OEM\Unattend.xml" "%TARGETOS%\Panther\Unattend.xml" /y
copy "%TARGETOSDRIVE%\Recovery\OEM\LayoutModification.xml"
"%TARGETOSDRIVE%\Users\Default\AppData\Local\Microsoft\Windows\Shell\LayoutModification.xml" /y
copy "%TARGETOSDRIVE%\Recovery\OEM\TaskbarLayoutModification.xml"
"%TARGETOS%\OEM\TaskbarLayoutModification.xml" /y
xcopy "%TARGETOSDRIVE%\Recovery\OEM\OOBE\Info" "%TARGETOS%\System32\Info\" /s

rem Recommended: Create a pagefile for devices with 1GB or less of RAM.
wpeutil CreatePageFile /path=%TARGETOSDRIVE%\PageFile.sys /size=256

EXIT 0
```

**ResetConfig.xml:** Note, this example points to the same script twice, so it can be used by both the **Keep my files** or **Remove everything** features.

```

<?xml version="1.0" encoding="utf-8"?>
<!-- ResetConfig.xml -->
<Reset>
    <Run Phase="BasicReset_AfterImageApply">
        <Path>CommonCustomizations.cmd</Path>
        <Duration>2</Duration>
    </Run>
    <Run Phase="FactoryReset_AfterImageApply">
        <Path>CommonCustomizations.cmd</Path>
        <Duration>2</Duration>
    </Run>
    <!-- May be combined with Recovery Media Creator
        configurations - insert SystemDisk element here -->
</Reset>

```

## Preserving and retrieving files

With the **Keep my files** feature, you can use sample scripts to preserve files that would otherwise be removed, by placing them in a temporary location in memory. You cannot keep files with the **Remove everything** feature.

You can use the following locations for storage, if needed.

- **Windows PE RAM drive (X:)**. This virtual drive is created by Windows PE, and stays active during the **Keep my files** process. You can use it with the **Keep my files** feature to save data before the partition is refreshed, and to restore the data after the partition refresh is complete. The amount of available memory is limited to the amount of RAM on the system, minus the amount of RAM needed for the Windows RE tools when fully expanded. For instructions about mounting Windows RE and determining the fully-expanded file size, see [Customize Windows RE](#).
- **Designated OEM partition**. You can leave extra room on a partition. For example, you can leave room on the recovery image partition, and use scripts to temporarily assign a drive letter and then save files to that partition. However, if your user uses the recovery media to repartition the disks, the data on these partitions might be lost during the recovery process.

These sample scripts preserve the Windows log files. Save these scripts in the `C:\Recovery\OEM` folder.

### ResetConfig.xml

```

<?xml version="1.0" encoding="utf-8"?>
<!-- ResetConfig.xml -->
<Reset>
    <Run Phase="BasicReset_BeforeImageApply">
        <Path>SaveLogFiles.cmd</Path>
        <Duration>4</Duration>
    </Run>
    <Run Phase="BasicReset_AfterImageApply">
        <Path>RetrieveLogFiles.cmd</Path>
        <Duration>2</Duration>
    </Run>
    <!-- May be combined with Recovery Media Creator
        configurations - insert SystemDisk element here -->
</Reset>

```

**SaveLogFiles.cmd**: Saves log files to a temporary folder in memory

```

:rem == SaveLogFiles.cmd

:rem == 1. Use the registry to identify the location of
:rem      the new operating system and the primary hard
:rem      drive. For example,
:rem      %TARGETOS% may be defined as C:\Windows
:rem      %TARGETOSDRIVE% may be defined as C:
for /F "tokens=1,2,3 delims= " %%A in ('reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RecoveryEnvironment"
/v TargetOS') DO SET TARGETOS=%%C

for /F "tokens=1 delims=\\" %%A in ('Echo %TARGETOS%') DO SET TARGETOSDRIVE=%%A

:rem == 2. Copy old Windows logs to a temporary folder in memory
mkdir X:\Temp
xcopy %TARGETOS%\Logs\*.* X:\temp\OldLogs /cheryki

EXIT 0

```

**RetrieveLogFiles.cmd:** Retrieves the files that were saved in memory by the SaveLogFiles.cmd script.

```

:rem == RetrieveLogFiles.cmd

:rem == This sample script retrieves the files that
:rem      were saved in memory by
:rem      SaveLogFiles.cmd,
:rem      and adds them back to the system.

:rem == 1. Use the registry to identify the location of
:rem      the new operating system and the primary drive.
:rem
:rem      %TARGETOS% is the Windows folder
:rem          (This later becomes C:\Windows)
:rem      %TARGETOSDRIVE% is the Windows partition
:rem          (This later becomes C:)
for /F "tokens=1,2,3 delims= " %%A in ('reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RecoveryEnvironment"
/v TargetOS') DO SET TARGETOS=%%C

for /F "tokens=1 delims=\\" %%A in ('Echo %TARGETOS%') DO SET TARGETOSDRIVE=%%A

:rem == 2. Copy the old logs to the new OS
:rem      at C:\Windows\OldLogs
mkdir %TARGETOS%\OldLogs
xcopy X:\Temp\OldLogs\* %TARGETOS%\OldLogs /cheryki

EXIT 0

```

## Extensibility points

The **Keep my files** feature can be summarized in the following steps:

1. PC boots into the Windows Recovery Environment (Windows RE).
2. **EXTENSIBILITY POINT A (BasicReset\_BeforeImageApply):** Add a script here to copy files, drivers, or settings that are not migrated by default when the user runs the **Keep my files** feature.
3. User accounts, settings, and data are gathered and moved to a temporary location.
4. A new copy of the OS is constructed in a temporary location using files from the Windows Component Store.
5. Customizations stored in provisioning packages under C:\Recovery\Customizations are applied to the new OS.
6. Drivers are copied from the existing OS and injected into the new OS.
7. Preinstalled Windows apps are restored from their backup location.
8. System-critical settings are applied to the new OS.
9. Existing OS is moved to C:\Windows.old.

10. New OS is moved to the root of the OS volume.
11. **EXTENSIBILITY POINT B (BasicReset\_AfterImageApply)**: Add a script here to restore customization files (unattend.xml, layoutmodification.xml), or restore files and settings you might have backed up at extensibility point A.
12. PC reboots to the new OS.
13. During first boot, user data and settings are reapplied.

The **Remove everything** feature can be summarized in the following steps:

1. PC boots into the Windows Recovery Environment (Windows RE).
2. User accounts, data and installed Windows apps and Windows desktop applications are removed from the OS volume.
3. Data volumes are formatted (if requested by the user).
4. Data erasure is performed on OS and data volumes (if requested by the user).
5. **EXTENSIBILITY POINT C (FactoryReset\_AfterDiskFormat)**: Add a script here to reconfigure data partitions if needed. **Important**: Do not modify the Windows partition.
6. A new copy of the OS is constructed in a temporary location using files from the Windows Component Store.
7. Customizations stored in provisioning packages under C:\Recovery\Customizations are applied to the new OS.
8. Drivers are copied from the existing OS and injected into the new OS.
9. Preinstalled Universal Windows apps are restored from their backup location.
10. Existing OS is removed.
11. New OS is moved to the root of the OS volume.
12. **EXTENSIBILITY POINT D (FactoryReset\_AfterImageApply)**: Add a script here to restore customization files (unattend.xml, layoutmodification.xml).
13. PC reboots to the new OS.
14. OOBE starts.

## Alternate method: copy scripts after deployment

A short time after your user completes OOBE, the recovery scripts are moved from the `C:\Recovery\OEM` folder to the recovery partition, at `R:\RecoveryImage\`.

In the unlikely event that push-button reset is used before this operation has taken place, these scripts may not run. To prevent that possibility, you can copy your recovery files directly to the recovery partition, `R:\RecoveryImage\` after your image has been deployed.

## Next steps

Now that you have customized the push-button reset experience, you can deploy the recovery image for push-button reset (Install.wim) to the recovery image partition.

To copy the Diskpart script, the ResetConfig.xml file, and the push-button reset recovery image (install.wim) to the recovery image partition of the destination PC, follow the instructions in the [Deploy Push-Button Reset Features](#) topic.

## Related topics

[Push-Button Reset Overview](#)

[Create Media to Run Push-Button Reset Features](#)

[Deploy Push-Button Reset Features](#)

[REAgentC Command-Line Options](#)

[ResetConfig XML Reference](#)

# Push-button reset frequently-asked questions (FAQ)

1/18/2019 • 4 minutes to read

QUESTION	ANSWER
Is Window RE required for a user to run the Push-button reset features?	<p>Yes. To run a Push-button reset feature, you must make the Windows RE boot image (Winre.wim) available on the local hard drive, and register its location by using the Reagents tool. You can use the default Winre.wim (available at C:\Windows\System32\Recovery), or a custom Winre.wim image. If Windows RE is not enabled on the local hard drive, users will have to boot Windows RE from media to access Push-button reset features.</p>
What is Compact OS?	<p>Compact OS is a collection of features which allow Windows 10 to be deployed on PCs with storage capacity as low as 16GB. The two primary technologies include:</p> <ul style="list-style-type: none"><li>• Compression of the runtime system files</li><li>• Single-instancing of installed customizations with the customizations package used by Push-button reset features</li></ul>
When should I use Compact OS?	<p>Both the compression of system files and single-instancing of customizations have similar characteristics as the WIMBoot technology from Windows 8.1. While Compact OS is supported on all hardware configurations, it is only recommended to be used on PCs with flash-based storage.</p>
How do I know if the OS is compressed?	<p>Compact.exe can be used to query the current compression state.</p>
How can I tell if a .ppkg is single-instanced?	<p>Run Fsutil.exe and specify the drive where the .ppkg is stored. For example: <code>fsutil.exe wim enumwims c:</code></p>
Are there any formatting requirements for the ResetConfig.xml file?	<p>Yes. Always use UTF-8 encoding, and do not use Unicode or ANSI. Add the following declaration in the ResetConfig.xml file, and in other .xml files:</p> <div style="border: 1px solid black; padding: 2px;"><code>&lt;?xml version="1.0" encoding="utf-8"?&gt;</code></div>
What types of removable media are supported for manufacturer-created recovery media?	<p>DVDs or USB flash drives can be used as recovery media. Note that Push-button reset features requires all recovery resources to be located on the same piece of media.</p>
Is recimg.exe supported in Windows 10?	<p>No recimg.exe is deprecated in Windows 10.</p>
Is Push-button reset supported on Windows Server ?	<p>No, this functionality is not supported on Windows Server 2016 Technical Preview.</p>
Can custom recovery solutions (i.e. not Push-button reset) restore the provisioning packages created using either Windows ICD or USMT's ScanState tool.	<p>Provisioning packages can only be applied by Push-button reset or deployment media created using Windows Imaging and Configuration Designer (ICD). Application of these packages by custom recovery solutions is not supported.</p>

QUESTION	ANSWER
If the provisioning package created using USMT's ScanState tool is larger than 4GB, will the "Create a recovery drive" utility allow customers to create USB recovery media?	Yes, the <b>Create a recovery drive</b> utility will split the provisioning package into smaller pieces before copying them to the USB flash drive. During recovery, the pieces will be reassembled into the original provisioning package.
I've preinstalled OS updates on the PC, how can I ensure that they are restored during recovery?	Updates are always restored during recovery. DISM.exe's /Cleanup-Image command with the /StartComponentCleanup and /ResetBase options are no longer required to mark all installed OS updates as permanent
I have files that need to be persisted/restored when Remove everything and Keep my files are performed, but I don't want to capture them using ScanState. Where should I put these files?	All contents under C:\Recovery\OEM are left unmodified during Remove everything and Keep my files. However, it should be noted that these contents will also be backed up onto the USB recovery media when using the Create a recovery drive utility.
I can't find the Keep my files option in Settings or Windows RE anymore. Where did the feature go?	Both Keep my files and Remove everything are now part of the same user experience, under the Reset this PC option in Settings and in Windows RE. When you launch the Reset this PC experience, you'll see additional options: <ul style="list-style-type: none"> <li>• <b>Keep my files</b> – This initiates the Keep my files feature.</li> <li>• <b>Remove everything</b> – This initiates the Remove everything feature.</li> <li>• <b>Restore factory settings</b> – On PCs upgraded from Windows 8/8.1, this initiates factory recovery using the existing recovery image.</li> </ul>
Should I specify the /drivers option when using ScanState to capture customizations?	The /drivers option is not required if the provisioning package being created is to be used for Push-button reset features. Push-button reset features persist the drivers which are already installed, making it unnecessary to reapply the factory-preinstalled drivers. Note: Driver applets installed outside of the driver INF package are captured using ScanState's /apps option.
How much available disk space is required in order for the Keep my files feature to run successfully?	If you have converted the installed customizations into file pointers referencing the customizations package created using ScanState, the required disk space is: 4GB + size_of_ppkg0.2 <i>Otherwise, the required disk space is: 4GB + size_of_ppkg2</i>
Am I required to reduce the size of the MSR partition from 128MB to 16MB based on the updated partition layout recommendations?	No. Windows continues to support 128MB MSR partitions. However, on PCs with limited storage capacity, a 16MB MSR partition is recommended to give end users as much available storage as possible.

QUESTION	ANSWER
Is there any known issue with using Remove everything to restore PCs back to factory condition after going through factory floor testing?	<p>Although PBR features are not intended to be used on factory floors, there's no technical limitation which prevents it. However, keep the following in mind when using Remove everything on the factory floor:</p> <ul style="list-style-type: none"><li>• If your factory floor testing includes activating Windows, Remove everything will not revert the unit back to a non-activated state</li><li>• Preinstalled RDX contents will be removed</li><li>• If the unit is not reset for multiple days after factory validation but remains powered on, the preinstalled languages except for the one selected during OOBE will be removed during maintenance</li><li>• End users will be able to tell that a unit has been reset during factory by looking for the PBR logs under C:\Windows\Logs\PBR</li></ul>

## Related topics

[Push-button reset overview](#)

# Push-button reset recommended validation scenarios

1/24/2019 • 5 minutes to read

Scenario	Initiation steps	Validations
<b>Reset this PC with Keep my files option from Settings (Priority: High)</b>	<ol style="list-style-type: none"><li>1. Select Settings &gt; Update &amp; security &gt; Recovery &gt; Reset this PC: Get started &gt; Keep my files</li><li>2. Follow the on-screen instructions</li></ol>	<ul style="list-style-type: none"><li>• Recovery completes successfully. Login screen is shown after recovery</li><li>• Classic Windows applications captured via ScanState.exe are reinstalled and working correctly</li><li>• Installed drivers are available and corresponding devices working as expected. Applications installed via drivers' setup.exe will NOT be restored unless captured via ScanState.exe</li><li>• Customizations restored via extensibility points are available after recovery</li><li>• WinRE is enabled (reagentc /info)</li></ul>
<b>Reset this PC with Remove everything option from Settings (Priority: High)</b>	<ol style="list-style-type: none"><li>1. Select Settings &gt; Update &amp; security &gt; Recovery &gt; Reset this PC: Get started &gt; Remove everything</li><li>2. When prompted, choose "Just remove my files" and follow the on-screen instructions.</li></ol>	<ul style="list-style-type: none"><li>• Recovery completes successfully. OOBE is shown after recovery</li><li>• Classic Windows applications captured via ScanState.exe are reinstalled and working correctly</li><li>• Installed drivers are available and corresponding devices working as expected. Applications installed via drivers' setup.exe will NOT be restored unless captured via ScanState.exe</li><li>• Customizations restored via extensibility points are available after recovery</li><li>• WinRE is enabled (reagentc /info)</li></ul>

<p><b>Creation of USB recovery drive</b></p> <p><b>(Priority: High)</b></p>	<ol style="list-style-type: none"> <li>1. Select Control Panel &gt; search for "Recovery" &gt; Create a recovery drive</li> <li>2. With the "Back up system files to the recovery drive" option selected, follow the on-screen instructions</li> </ol>	<ul style="list-style-type: none"> <li>• USB recovery drive created successfully</li> <li>• The USB recovery drive is bootable</li> <li>• Customization packages and resources under C:\Recovery\Customizations and C:\Recovery\OEM, if available, are backed up on the USB recovery drive under \sources\Customizations and \sources\OEM</li> </ul>
<p><b>Bare metal recovery from USB recovery drive</b></p> <p><b>(Priority: High)</b></p>	<ol style="list-style-type: none"> <li>1. Create a USB recovery drive and attach it to the device</li> <li>2. Boot the device using the USB recovery drive</li> <li>3. Once booted to the USB recovery drive, select language and keyboard</li> <li>4. Select Troubleshoot &gt; Recover from a drive</li> <li>5. When prompted, choose "Just remove my files" and follow the on-screen instructions</li> </ol>	<ul style="list-style-type: none"> <li>• Recovery completes successfully and OOBE is shown after recovery</li> <li>• Classic Windows applications captured via ScanState.exe are reinstalled and working correctly</li> <li>• Installed drivers are available and corresponding devices working as expected. Applications installed via drivers' setup.exe will NOT be restored unless captured via ScanState.exe</li> <li>• Customizations restored via extensibility points are available after recovery</li> <li>• Customization packages and resources, if available, are restored to C:\Recovery\Customizations and C:\Recovery\OEM</li> <li>• Partition layout of the device is restored correctly</li> <li>• WinRE is enabled (reagentc /info)</li> </ul>

<p><b>Reset this PC with Keep my files option from USB recovery drive</b></p> <p><b>(Priority: Medium)</b></p>	<ol style="list-style-type: none"> <li>1. Create a USB recovery drive and attach to the device</li> <li>2. Boot the device using the USB recovery drive</li> <li>3. Once booted to the USB recovery drive, select language and keyboard</li> <li>4. Select Troubleshoot &gt; Reset this PC &gt; Remove everything &gt; Keep my files</li> <li>5. Follow the on-screen instructions</li> </ol>	<ul style="list-style-type: none"> <li>• Recovery completes successfully. Login screen is shown after recovery</li> <li>• Classic Windows applications captured via ScanState.exe are reinstalled and working correctly</li> <li>• Installed drivers are available and corresponding devices working as expected. Applications installed via drivers' setup.exe will NOT be restored unless captured via ScanState.exe</li> <li>• Customizations restored via extensibility points are available after recovery</li> <li>• WinRE is enabled (reagentc /info)</li> </ul>
<p><b>Reset this PC with Remove everything option from USB recovery drive</b></p> <p><b>(Priority: Medium)</b></p>	<ol style="list-style-type: none"> <li>1. Create a USB recovery drive and attach to the device</li> <li>2. Boot the device using the USB recovery drive</li> <li>3. Once booted to the USB recovery drive, select language and keyboard</li> <li>4. Select Troubleshoot &gt; Reset this PC &gt; Remove everything</li> <li>5. When prompted, choose "Just remove my files" and follow the on-screen instructions</li> </ol>	<ul style="list-style-type: none"> <li>• Recovery completes successfully. OOBE is shown after recovery</li> <li>• Classic Windows applications captured via ScanState.exe are reinstalled and working correctly</li> <li>• Installed drivers are available and corresponding devices working as expected. Applications installed via drivers' setup.exe will NOT be restored unless captured via ScanState.exe</li> <li>• Customizations restored via extensibility points are available after recovery</li> <li>• WinRE is enabled (reagentc /info)</li> </ul>

<p><b>Reset this PC with Keep my files option from on-disk WinRE</b></p> <p><b>(Priority: Medium)</b></p>	<ol style="list-style-type: none"> <li>1. Select Settings &gt; Update &amp; security &gt; Recovery &gt; Advanced startup: Restart now. The device reboots to WinRE.</li> <li>2. Select Troubleshoot &gt; Reset this PC &gt; Keep my files</li> <li>3. Follow the on-screen instructions</li> </ol>	<ul style="list-style-type: none"> <li>• Recovery completes successfully. Login screen is shown after recovery</li> <li>• Classic Windows applications captured via ScanState.exe are reinstalled and working correctly</li> <li>• Installed drivers are available and corresponding devices working as expected. Applications installed via drivers' setup.exe will NOT be restored unless captured via ScanState.exe</li> <li>• Customizations restored via extensibility points are available after recovery</li> <li>• WinRE is enabled (reagentc /info)</li> </ul>
<p><b>Reset this PC with Remove everything option from on-disk WinRE</b></p> <p><b>(Priority: Medium)</b></p>	<ol style="list-style-type: none"> <li>1. Select Settings &gt; Update &amp; security &gt; Recovery &gt; Advanced startup: Restart now. The device reboots to WinRE.</li> <li>2. Select Troubleshoot &gt; Reset this PC &gt; Remove everything</li> <li>3. When prompted, choose "Just remove my files"</li> <li>4. Follow the on-screen instructions</li> </ol>	<ul style="list-style-type: none"> <li>• Recovery completes successfully. OOBE is shown after recovery</li> <li>• Classic Windows applications captured via ScanState.exe are reinstalled and working correctly</li> <li>• Installed drivers are available and corresponding devices working as expected. Applications installed via drivers' setup.exe will NOT be restored unless captured via ScanState.exe</li> <li>• Customizations restored via extensibility points are available after recovery</li> <li>• WinRE is enabled (reagentc /info)</li> </ul>

<p><b>Reset this PC with Remove everything and Remove files and clean the drive options</b></p> <p><b>(Priority: Low)</b></p>	<ol style="list-style-type: none"> <li>1. Select Settings &gt; Update &amp; security &gt; Recovery &gt; Advanced startup: Restart now. The device reboots to WinRE.</li> <li>2. Select Troubleshoot &gt; Reset this PC &gt; Remove everything</li> <li>3. When prompted, choose "Fully clean the drive" and follow the on-screen instructions</li> </ol>	<ul style="list-style-type: none"> <li>• Recovery completes successfully. OOBE is shown after recovery</li> <li>• Classic Windows applications captured via ScanState.exe are reinstalled and working correctly</li> <li>• Installed drivers are available and corresponding devices working as expected. Applications installed via drivers' setup.exe will NOT be restored unless captured via ScanState.exe</li> <li>• Customizations restored via extensibility points are available after recovery</li> <li>• WinRE is enabled (reagentc /info)</li> </ul>
<p><b>Bare metal recovery with "Clean the drive" option</b></p> <p><b>(Priority: Low)</b></p>	<ol style="list-style-type: none"> <li>1. Create a USB recovery drive and attach it to the device</li> <li>2. Boot the device using the USB recovery drive</li> <li>3. Once booted to the USB recovery drive, select language and keyboard</li> <li>4. Select Troubleshoot &gt; Recover from a drive</li> <li>5. When prompted, choose "Fully clean the drive" and follow the on-screen instructions</li> </ol>	<ul style="list-style-type: none"> <li>• Recovery completes successfully. OOBE is shown after recovery</li> <li>• Classic Windows applications captured via ScanState.exe are reinstalled and working correctly</li> <li>• Installed drivers are available and corresponding devices working as expected. Applications installed via drivers' setup.exe will NOT be restored unless captured via ScanState.exe</li> <li>• Customizations restored via extensibility points are available after recovery</li> <li>• Customization packages and resources, if available, are restored to C:\Recovery\Customizations and C:\Recovery\OEM</li> <li>• Partition layout of the device is restored correctly</li> <li>• WinRE is enabled (reagentc /info)</li> </ul>

## Related topics

[Push-button reset overview](#)

# Bare metal recovery

1/25/2019 • 3 minutes to read

If the user needs to replace their hard drive or completely wipe it, they can use bootable recovery media to perform bare metal recovery. Bare metal recovery removes all existing partitions on the system disk and recreates all partitions, before restoring software onto the PC. Two types of recovery media are supported:

- **User-created recovery media** using the **Create a recovery drive** utility in Windows 10. This backs up the files needed to restore the PC to a pristine state.
- **Manufacturer-created recovery media** for support and refurbishing scenarios by placing a recovery image on a piece of bootable Windows RE media.

**When user-created recovery media are used, the bare metal recovery feature can be summarized in the following steps:**

1. The system disk is identified.
2. All partitions from the system disk are removed.
3. Data erasure is performed on the system disk (if requested by the user).
4. Factory or default partition layout is recreated on the system disk.
5. All partitions are formatted.
6. Recovery files from recovery media are copied to the OS volume.
7. A new copy of the OS is constructed at the root of the OS volume.
8. Customizations stored in provisioning packages are applied.
9. Drivers are injected into the new OS.
10. Preinstalled Windows apps are restored.
11. Boot files are configured on the system partition.
12. PC reboots to the new OS.
13. OOBESTARTS.

## Data removal options

When users use the bare metal recovery feature, they can choose to perform data erasure on the entire system disk before the factory partition layout is reapplied. On most PCs, this data erasure process is done in software, writing cryptographically random patterns to the entire LBA range of the system disk once.

However, on certain hardware configurations, the data erasure process is performed by the storage device's hardware controller. This often takes less time to complete and is usually more thorough in removing remnant data. Hardware-based data erasure is supported on PCs with storage devices which meet the following criteria:

- eMMC
- Supports the **Secure Trim** and **Sanitize** commands

## System disk selection

Bare metal recovery automatically identifies the system disk using the following methods:

- Adaptor location path and GUID of the system disk are written to a UEFI variable during OOBESTARTS.
  - Performed only when both the system and Windows partitions are on the system disk.
  - The variable is updated if necessary when Windows RE gets disabled and then re-enabled.
- During bare metal recovery, if multiple internal disks are detected, the system disk is searched in this order:

- Disk with GUID matching the value stored in the UEFI variable.
- Disk with location path matching the value stored in firmware.
- Disk with an existing ESP.
  - If multiple disks with ESP are found, bare metal recovery will not proceed.
- Uninitialized (raw) disk.
  - If multiple uninitialized disks are found, bare metal recovery will not proceed.
- On legacy BIOS/MBR systems, the BIOS-reported system disk is used.

### User-created recovery media

When users create USB recovery media using the **Create a recovery drive** utility, the resulting media always contain a bootable copy of Windows RE. This gives users access to troubleshooting and recovery tools when booting from recovery media.

Users can optionally back up files required to perform bare metal recovery. When the option is selected, the following are copied onto the USB recovery media as well:

- Windows Component Store
- Installed drivers
- Backup of preinstalled Windows apps
- Provisioning packages containing preinstalled customizations (under C:\Recovery\Customizations)
- Push-button Reset configuration XML and scripts (under C:\Recovery\OEM)

### Manufacturer-created recovery media

Bare metal recovery supports the use of a recovery WIM image when the media are prepared by manufacturers. This type of media is primarily used in support and refurbishing scenarios.

Manufacturer-created media must contain the following:

1. A bootable Windows RE image.
2. A Push-button reset-compatible recovery image (install.wim).
3. A Push-button reset configuration file (Resetconfig.xml) which specifies disk partitioning information.
4. A DISKPART script to perform partitioning of the disk.

## Let's set it up!

- **Bare metal reset/recovery: enable your users to create recovery media** using the **Create a recovery drive** utility in Windows 10. This backs up the files needed to restore the PC to a pristine state.
- **Bare metal reset/recovery: create recovery media while deploying new devices** for support and refurbishing scenarios by placing a recovery image on a piece of bootable Windows RE media.

## Related topics

- [Push-button reset](#)
- [How push-button reset features work](#)

# Bare metal reset/recovery: create recovery media while deploying new devices

1/18/2019 • 3 minutes to read

Recovery media (bare metal recovery) helps restore a Windows device to the factory state, even if the user needs to replace the hard drive or completely wipe the drive clean.

You can include this media with new devices that you provide to your customers using the same Windows images used to deploy the devices.

## Note

- The PC firmware/BIOS must be configured so that the PC can boot from the media (USB drive or DVD drive).
- The USB flash drive or DVD recovery media must have enough space for the Windows image.
- If the Windows images are larger than 32GB or are larger the media you're using (for example, 4.7GB DVDs), you'll need to [split the Windows image file to span across multiple DVDs](#).

To create a bootable USB recovery drive for a personal device, see [Create a USB recovery drive](#).

## Create a bootable Windows RE image

To create the recovery media that you can include with the PC, you must have the following:

- A Windows image (Install.wim). You can either use the base Windows image or a customized recovery image.
- A Windows RE tools image (Winre.wim). You can either extract the base Windows RE tools image from the Windows image, or use a [customized Windows RE image](#).

### Step 1: Open the Deployment and Imaging Tools Environment

1. Download and install the [Windows Assessment and Deployment Kit \(ADK\)](#).
2. On your technician PC: Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.

### Step 2: Extract the Windows RE image from the Windows image

1. Mount the Windows image:

```
md c:\mount\Windows  
Dism /Mount-Image /ImageFile:D:\sources\install.wim /Index:1 /MountDir:C:\mount
```

2. Copy the Windows RE image.

```
md C:\Images  
xcopy C:\mount\Windows\System32\Recovery\winre.wim C:\Images\winre.wim /h
```

3. Unmount the Windows image:

```
Dism /Unmount-Image /MountDir:C:\mount\winre /Discard
```

### Step 3: Create a working folder for Windows RE files

1. Create a folder structure for Windows RE, which is based on Windows PE:

```
copytree amd64 C:\resetmedia_amd64
```

where *amd64* is the architecture of the system you are creating media for.

2. Replace the default Windows PE boot image (Boot.wim) with a Windows RE tools image.

```
xcopy C:\MyImages\winre.wim C:\resetmedia_amd64\media\sources\boot.wim /h
```

### Step 4: Add the Windows image

- Copy the Windows image to the working folder.

```
copy D:\sources\install.wim C:\resetmedia_amd64\media\sources\install.wim
```

where *D:\sources\install.wim* is either the base Windows image or a customized push-button reset recovery image.

### Step 5: Add bare metal recovery configuration scripts

- If you're using a customized partition layout, add bare metal recovery configuration scripts to the working folder, under \sources. For more info, see [Bare Metal Reset/Recovery: Enable Your Users to Create Media](#).

```
copy E:\Recovery\RecoveryImage\ResetConfig.xml C:\resetmedia_amd64\media\sources\ResetConfig.xml  
copy E:\Recovery\RecoveryImage\ResetPartitions-UEFI.txt  
C:\resetmedia_amd64\media\sources\ResetPartitions-UEFI.txt
```

## Create bootable media

### To create a bootable USB flash drive:

1. Install Windows RE to a USB flash drive:

```
MakewinpeMedia /ufd C:\resetmedia_amd64 F:
```

where *F* is the drive letter of the USB flash drive.

2. Label the USB flash drive with a descriptive name:

In File Explorer, right-click the drive, and select **Rename**, and type **Full-PC Recovery**.

### To create a bootable DVD:

1. Create a DVD image file:

```
MakewinpeMedia /iso C:\resetmedia_amd64 C:\resetmedia_amd64\RecoveryImage.iso
```

2. Insert a DVD.
3. In File Explorer, navigate to `C:\resetmedia_amd64`, right-click `RecoveryImage.iso`, and then click **Burn disc image**.

## Test the bare metal recovery features

1. On a PC with an empty hard drive, insert your new recovery media.
2. Start the PC, press a key to open the firmware boot menus, and then select the appropriate boot device.
3. At the **Windows RE Tools** menus, select a keyboard layout, for example, **US**.
4. Click **Troubleshoot > Remove everything > Next**

### Note

If you are testing on the same PC, and you have not cleaned the hard drive, you may be prompted to select a drive. Select Windows 10.

Select \*\*Yes, repartition the drives\*\* &gt; \*\*Just remove my files\*\* &gt; \*\*Reset\*\*.

Windows resets the computer to its original state by using the recovery image.

## Large-Scale Deployment

If you are deploying USB keys with your computers, you can create a basic copy of the Windows recovery media on USB by using the steps above. After you have performed final customization of the image, you can boot the computer to Windows PE, and update the install.wim image on the USB recovery media.

You can potentially save manufacturing time by appending the Windows image on the USB flash drive, rather than recapturing the entire Windows image. If you do this, you must also update the ResetConfig.xml configuration file element: `RestoreFromIndex` to the appropriate index number. For more information, see [Append a Volume Image to an Existing Image Using DISM and ResetConfig XML Reference](#).

## Related topics

[Bare Metal Reset/Recovery: Enable Your Users to Create Media](#)

[Push-Button Reset Overview](#)

[ResetConfig XML Reference](#)

[REAgentC Command-Line Options](#)

# Bare metal reset/recovery: enable your users to create recovery media

1/18/2019 • 6 minutes to read

Recovery media (bare metal recovery) helps restore a Windows device to the factory state, even if the user needs to replace the hard drive or completely wipe the drive clean.

Windows uses the built-in Windows files, including recent Windows and driver updates, plus any customizations included in the OEM provisioning package, to create the recovery media.

If you deploy Windows using the default partition layout, your users will be able to create bare metal recovery media by default.

If you're deploying Windows with a custom partition layout, you'll need to add a few configuration files to enable your users to create bare metal recovery media:

- A **partition reset script**, which is a modified DiskPart script that resets your custom partition layout.
- A **push-button reset configuration file (ResetConfig XML)** that identifies the Windows and Windows RE partitions.

**Note:** In Windows 10, version 1607 and later, desktop applications and settings captured in [siloed provisioning packages](#) will not be restored using this media. Regular customizations packages (.ppkg) captured using the ScanState tool are not affected by this issue.

## Creating configuration files

### Partition reset script

1. In Notepad, create a configuration file that partitions the hard drive after the hard drive has been reset. This script should be the same as the script used to create partitions on the hard drive, with the following exceptions:

- The script should not contain commands to select or clean the drive. Windows identifies the system drive automatically. To learn more, see [Identifying the System Drive](#) later in this topic.
- The script should assign letters to the system partition, the Windows partition, and the Windows RE tools partition.

Examples:

UEFI (based on [UEFI/GPT-based hard drive partitions](#)):

```
rem == ResetPartitions-UEFI.txt ==
rem == These commands are used with DiskPart to
rem    reset the drive and recreate five partitions
rem    for a UEFI/GPT-based computer.
rem    Adjust the partition sizes to fill the drive
rem    as necessary. ==
rem == The differences between this file and
rem    CreatePartitions-UEFI.txt
rem    are noted in parenthesis.
rem        (NOT USED: select disk 0)
rem        (NOT USED: clean)
convert gpt
rem == 1. System partition =====
create partition efi size=100
rem    ** NOTE: For Advanced Format 4Kn drives,
rem           change this value to size = 260 **
format quick fs=fat32 label="System"
assign letter="S"
rem == 2. Microsoft Reserved (MSR) partition =====
create partition msr size=128
rem == 3. Windows partition =====
rem ==   a. Create the Windows partition =====
create partition primary
rem ==   b. Create space for the recovery tools ===
shrink minimum=500
rem    ** Update this size to match the size of
rem       the recovery tools (winre.wim)
rem       plus some free space.
rem
rem ==   c. Prepare the Windows partition =====
format quick fs=ntfs label="Windows"
assign letter="C"
rem == 4. Recovery tools partition =====
create partition primary
format quick fs=ntfs label="Recovery tools"
assign letter="R"
set id="de94bba4-06d1-4d40-a16a-bfd50179d6ac"
gpt attributes=0x8000000000000001
list volume
```

BIOS (based on [BIOS/MBR-based hard drive partitions](#)):

```

rem == ResetPartitions-BIOS.txt ==
rem == These commands are used with DiskPart to
rem    reset the drive and create three partitions
rem    for a BIOS/MBR-based computer.
rem    Adjust the partition sizes to fill the drive
rem    as necessary. ==
rem == The differences between this file and
rem    CreatePartitions-BIOS.txt
rem    are noted in parenthesis.
rem        (NOT USED: select disk 0 )
rem        (NOT USED: clean )
rem == 1. System partition =====
create partition primary size=100
format quick fs=ntfs label="System"
assign letter="S"
active
rem == 2. Windows partition =====
rem ==   a. Create the Windows partition =====
create partition primary
rem ==     b. Create space for the recovery tools
shrink minimum=500
rem        ** Update this size to match the size of
rem        the recovery tools (winre.wim)
rem        plus some free space.
rem ==   c. Prepare the Windows partition =====
format quick fs=ntfs label="Windows"
assign letter="C"
rem == 3. Recovery tools partition =====
create partition primary
format quick fs=ntfs label="Recovery"
assign letter="R"
set id=27
list volume

```

- Save your file, for example, E:\Recovery\Recovery\Image\ResetPartitions-UEFI.txt.

### **Push-button reset configuration file (ResetConfig.xml)**

- In Notepad, create a configuration file that points to your push-button reset partition script.

For information about configuring this file, see [ResetConfig XML Reference](#).

UEFI:

```

<?xml version="1.0" encoding="utf-8"?>
<!!-- ResetConfig.xml for UEFI -->
<Reset>
    <!-- May be combined with custom scripts - insert Run Phase elements here -->
    <SystemDisk>
        <DiskpartScriptPath>ResetPartitions-UEFI.txt</DiskpartScriptPath>
        <MinSize>75000</MinSize>
        <WindowsREPartition>4</WindowsREPartition>
        <WindowsREPath>Recovery\WindowsRE</WindowsREPath>
        <OSPartition>3</OSPartition>
    </SystemDisk>
</Reset>

```

BIOS:

```

<?xml version="1.0" encoding="utf-8"?>
<!-- ResetConfig.xml for BIOS -->
<Reset>
    <!-- May be combined with custom scripts - insert Run Phase elements here -->
    <SystemDisk>
        <DiskpartScriptPath>ResetPartitions-BIOS.txt</DiskpartScriptPath>
        <MinSize>75000</MinSize>
        <WindowsREPartition>3</WindowsREPartition>
        <WindowsREPath>Recovery\WindowsRE</WindowsREPath>
        <OSPartition>2</OSPartition>
    </SystemDisk>
</Reset>

```

2. Save the file using the UTF-8 file format:

Click **File**, and then click **Save As**. In the **Encoding** box, select **UTF-8**, and save this file as E:\Recovery\RecoveryImage\ResetConfig.xml.

## Enable users to create media

Users can use this option to create recovery media when needed.

### Step 1: Add the configuration files to the destination computer

1. On your destination computer, insert the USB flash drive with the configuration files.
2. Copy the configuration files to the destination computer:

```
Copy E:\Recovery\RecoveryImage\* R:\RecoveryImage\*
```

where *E* is the drive letter of the USB flash drive and *R* is the drive letter of the recovery image partition.

### Step 2: Test that Windows can create recovery media

1. Restart the destination computer, and complete Out-Of-Box Experience (OOBE).
2. Click **Start**, type **create a recovery drive**, and select **Create a recovery drive**, and click **Yes** at the UAC prompt.
3. Insert a USB flash drive.
4. Select **Copy the recovery partition from the PC to the recovery drive** > **Next** > **Next** > **Create**.

### Step 3: Test the recovery media

1. On a computer that has no operating system, insert your recovery media.
2. Start the computer, press a key to open the firmware boot menus, and then select the appropriate boot device.
3. At the **Windows RE Tools** menus, select a keyboard layout, for example, **US**.
4. Click **Troubleshoot** > **Remove everything** > **Next**. If you're prompted to clean the drive, select **Yes**.
5. Select **Yes, repartition the drives** > **Just remove my files** > **Reset**.

### Troubleshooting:

- Make sure that ResetConfig.xml is saved as a UTF-8 file.
- Make sure that the filename listed in the **<DiskpartScriptPath>** element of the ResetConfig.xml file matches the filename in the Diskpart script.
- Make sure that the Diskpart script doesn't include commands to select the drive or clean the drive (  
select disk 0 , clean ).

# Identifying the system drive

Windows identifies the system drive using the following methods:

**BIOS-based computers:** the BIOS-reported system drive is used.

**UEFI-based computers:** When Windows RE is enabled by using the `reagentc /setreimage` command, Windows writes the adaptor location path and GUID of the system disk to a UEFI variable. This step is only performed when both the system and OS partitions are on the system drive. The variable is updated if necessary when Windows RE gets disabled and then re-enabled.

**If multiple local drives are detected, Windows identifies the system drive by searching in the following order:**

1. Windows searches for a drive with a GUID matching the value stored in firmware.
2. Windows searches for a drive with a location path matching the value stored in firmware.
3. Windows searches for a drive with an existing ESP.

If multiple drives with ESP are found, the recovery process will not proceed.

4. Windows searches for an uninitialized (raw) disk.

If multiple uninitialized disks are found, the recovery process will not proceed.

## Related topics

[Push-Button Reset Overview](#)

[ResetConfig XML Reference](#)

[Bare metal reset/recovery: create recovery media while deploying new devices](#)

[UEFI/GPT-based hard drive partitions](#)

[BIOS/MBR-based hard drive partitions](#)

# REAgentC command-line options

1/18/2019 • 3 minutes to read

You can use the REAgentC.exe tool to configure a Windows Recovery Environment (Windows RE) boot image and a push-button reset recovery image, and to administer recovery options and customizations. You can run the **REAgentC** command on an offline Windows image or on a running Windows operating system.

**Note** If you are using Windows PE 2.X, 3.X, or 4.X to configure recovery on an offline Windows 10 installation, you must use the Winrecfg.exe file from the Recovery folder of the Windows Assessment and Deployment Kit (Windows ADK). Winrecfg.exe supports only the offline operations that REAgentC.exe supports.

## REAgentC Commands

The following command-line options are available for Windows RE:

**reagentc.exe** <command> <arguments>

The following table describes these command-line options:

OPTION	ONLINE/OFFLINE	DESCRIPTION
<b>/setreimage /path</b> <path_to_Windows_RE_image> [ <b>/target</b> <path_to_offline_image>]	Both	<p>Sets the location of a Windows RE boot image. In Windows 10, Windows 8.1, Windows 8, Windows Server 2016 Technical Preview, Windows Server 2012 R2, and Windows Server 2012, <b>/path</b> supports UNC paths to locations on the local disk. For example:</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"><pre>Reagentc /setreimage /path S:\Recovery\WindowsRE</pre></div> <p>Use the <b>/target</b> option to specify the location of the Windows image when you apply the setting offline. For example:</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"><pre>Reagentc /setreimage /path T:\Recovery\WindowsRE /target W:\Windows</pre></div>

OPTION	ONLINE/OFFLINE	DESCRIPTION
<pre>/enable [/auditmode] [/osguid &lt;bcd_guid&gt;]</pre>	<p>Both</p>	<p>Enables a custom Windows RE boot image.</p> <p>The <b>/enable</b> option runs automatically during the specialize configuration pass. If you don't specify a Windows RE boot image, the computer attempts to enable Windows RE by using the default Winre.wim file from the \Windows\System32\Recovery folder.</p> <ul style="list-style-type: none"> <li>• <b>/auditmode:</b></li> </ul> <p>By default, the <b>/enable</b> option doesn't perform any actions when Windows is in audit mode. To override the default behavior and enable Windows RE from audit mode, specify the <b>/auditmode</b> option. For example:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>Reagentc /enable /auditmode</pre> </div> <p>If you generalize the image after you use the <b>/enable</b> option in audit mode, Windows RE is disabled until you use the <b>/enable</b> option again or until after the specialize configuration pass runs.</p> <ul style="list-style-type: none"> <li>• <b>/osguid &lt;bcd_guid&gt;:</b></li> </ul> <p>This option allows you to enable your custom Windows RE boot image from Windows PE. It can only be used after bcdboot.exe has been run. &lt;bcd_guid&gt; is the Boot Configuration Data (BCD) identifier of the target Windows installation, obtained by running</p> <div style="border: 1px solid black; padding: 5px; margin-top: 5px;"> <pre>bcdedit -enum -v .</pre> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>Reagentc /enable /osguid {00000000- 0000-0000-0000- 000000000000}</pre> </div>

OPTION	ONLINE/OFFLINE	DESCRIPTION
<b>/disable</b>	Online	<p>Disables any active Windows RE image that is mapped to the online image. For example:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>Reagentc /disable</pre> </div>
<b>/boottore</b>	Online	<p>Specifies that Windows RE starts automatically the next time the system starts. For example:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>Reagentc /boottore</pre> </div>
<b>/setosimage /path &lt;path_to_recovery_image&gt;</b> <b>/index &lt;image_index&gt; [/target &lt;path_to_offline_image&gt;]</b>	Both	<p>This setting is not used in Windows 10.</p> <p>Registers the location of a push-button reset image in an online or offline image. The recovery image must be in the Windows image (.wim) format.</p> <p>The <b>/index</b> option specifies the index number of the recovery image to use from within a .wim file. For example:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>Reagentc /setosimage /path R:\RecoveryImage /index 1</pre> </div> <p>Use the <b>/target</b> option to specify the location of the offline Windows image. For example:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>Reagentc /setosimage /path R:\RecoveryImage /index 1 /target W:\Windows</pre> </div>

OPTION	ONLINE/OFFLINE	DESCRIPTION
<pre>/info [/target &lt;path_to_offline_image&gt;]</pre>	Both	<p>Displays the current status of Windows RE and any available recovery image on an online or offline image. For example, the following command returns the status of the online operating system:</p> <div data-bbox="1041 422 1429 500" style="border: 1px solid black; padding: 5px;"> <pre>Reagentc /info</pre> </div> <p>Use the <b>/target</b> option to obtain configuration information about an offline image. For example:</p> <div data-bbox="1041 646 1429 725" style="border: 1px solid black; padding: 5px;"> <pre>Reagentc /info /target W:\Windows</pre> </div>
<pre>/setbootshelllink [/configfile &lt;path_to_BootShellXML&gt;] [/target &lt;path_to_offline_image&gt;]</pre>	Both	<p>Registers the link to a custom tool that appears in the Windows boot options menu. For example:</p> <div data-bbox="1041 927 1429 1096" style="border: 1px solid black; padding: 5px;"> <pre>Reagentc /setbootshelllink /configfile F:\BootMenu\AddDiagnosticsTo olToBootMenu.xml</pre> </div> <p>The BootShellXML file is an.xml file that contains the <code>&lt;BootShell&gt;</code> element and the <code>&lt;Name&gt;</code> and <code>&lt;Description&gt;</code> attributes that you want to appear in the link. For more information, see <a href="#">Customize Windows RE</a>.</p> <p>Use the <b>/target</b> option to specify the location of the offline Windows image. If this argument is not used, the running operating system is used. For example:</p> <div data-bbox="1041 1556 1429 1635" style="border: 1px solid black; padding: 5px;"> <pre>Reagentc /setbootshelllink /target W:\Windows</pre> </div>

## Related topics

[Windows RE Troubleshooting Features](#)

# ResetConfig XML reference

1/18/2019 • 3 minutes to read

This reference describes all XML elements that are used to author the ResetConfig.xml file, used to configure Windows Recovery Environment push-button reset features.

## Reset

The `Reset` XML element can contain the elements: `Run` and `SystemDisk`.

## Run

The `Run` XML element is used to add custom scripts to push-button reset features.

You can specify up to four `Run` elements in a single ResetConfig.xml file. Each `Run` element must contain a different `[ExtPoint]` value for the `Phase` attribute.

The following table describes the valid elements that can be added to the `Run` element:

ELEMENT	DESCRIPTION
<code>Run Phase="[ExtPoint]"</code>	<p>Each <code>Run</code> element defines the extensibility point to be used, the script that is executed at that extensibility point, and estimated time duration in minutes.</p> <p>The <code>Phase</code> attribute is required. It accepts only the following values for <code>[ExtPoint]</code>:</p> <ul style="list-style-type: none"><li>• <code>BasicReset_BeforeImageApply</code>. Runs the specified program at extensibility point A.</li><li>• <code>BasicReset_AfterImageApply</code>. Runs the specified program at extensibility point B</li><li>• <code>FactoryReset_AfterDiskFormat</code>. Runs the specified program at extensibility point C</li><li>• <code>FactoryReset_AfterImageApply</code>. Runs the specified program at extensibility point D</li></ul> <p>You can specify up to four <code>Run</code> sections in a single ResetConfig.xml file. However, each <code>Run</code> section must contain a different value for the phase attribute.</p>
<code>Path</code>	<p>Specifies the location of the script for a particular <code>Run</code> section.</p> <p>The path must be the relative path of the script from the folder which contains ResetConfig.xml (usually this is C:\Recovery\OEM).</p>

ELEMENT	DESCRIPTION
<code>Duration</code>	<p>Specifies the estimated time, in minutes, that you expect the custom script to run. This estimate is used to display progress information in the GUI.</p> <p>The duration must be an integer and must be between 1 and 5.</p>
<code>Param</code>	<p>Specifies the command-line parameters to use when you run the custom script or executable file. The value is treated as a string, and can contain multiple parameters.</p> <p><code>Param</code> does not support empty elements. If your script does not require parameters, then do not include this element. For examples, see <a href="#">Using ResetConfig.xml</a> later in this topic.</p>

## SystemDisk

The `SystemDisk` element customizes bare metal recovery functionality. For more information, see [Create Media to Run Push-Button Reset Features](#).

You can specify one `SystemDisk` section. Here's the required and optional elements:

ELEMENT	DESCRIPTION
<code>MinSize</code>	<p>Required. Specifies the minimum required size for the primary hard drive, in megabytes.</p> <p>Bare metal recovery won't proceed if the system disk doesn't meet this size requirement.</p>
<code>DiskpartScriptPath</code>	<p>Required. Path to Diskpart script relative to <code>C:\Recovery\OEM</code>. The script should assume that all existing partitions have been deleted, and the system disk has focus in Diskpart.</p> <p>For example, if the recovery scripts are located at <code>C:\Recovery\OEM\Scripts\RecreatePartitions.dps</code>, use the value <code>\Scripts\RecreatePartitions.dps</code>.</p>
<code>OSPartition</code>	<p>Required. The partition to which the OS should be restored. The ESP or active partition must be on the same disk as the OS.</p>
<code>WindowsREPartition</code>	<p>Required. Specifies the partition where the Windows RE boot image is installed.</p>
<code>WindowsREPath</code>	<p>Required. Specifies the folder path where the Winre.wim boot image is copied and staged, relative to the root of the partition specified in the <code>WindowsREPartition</code> element.</p>

ELEMENT	DESCRIPTION
<code>RestoreFromIndex</code>	Optional. The index of the image within install.wim to be applied during bare metal recovery. This element is optional and is only needed on manufacturer-created recovery media
<code>Compact</code>	<p>Optional. Specifies whether the recovery image should be applied with per-file compression enabled. This element is optional and is only needed on manufacturer-created recovery media.</p> <p><code>Compact</code> accepts the following values:</p> <ul style="list-style-type: none"> <li>• <code>True</code> : Files applied from the image are compressed individually.</li> <li>• <code>False</code> (default value): Compression is not used.</li> </ul>
<code>RecoveryImagePartition</code>	This setting is deprecated in Windows 10.
<code>RecoveryImagePath</code>	This setting is deprecated in Windows 10.
<code>RecoveryImageIndex</code>	This setting is deprecated in Windows 10.
<code>WIMBoot</code>	This setting is deprecated in Windows 10.

## Using ResetConfig.xml

If you use a text editor to author your .xml files, you must save the document with an .xml file name extension, and use UTF-8 encoding. You must not use ANSI coding.

These files should be placed in the folder C:\Recovery\OEM, and will automatically be detected by Push-button reset features.

## Example

This is a code example for the ResetConfig.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<Reset>
  <Run Phase="BasicReset_BeforeImageApply">
    <Path>Fabrikam\CopyFiles.cmd</Path>
    <Duration>2</Duration>
  </Run>
  <Run Phase="BasicReset_AfterImageApply">
    <Path>Fabrikam\InstallDrivers.cmd</Path>
    <Param>/allDrivers</Param>
    <Duration>2</Duration>
  </Run>
  <Run Phase="FactoryReset_AfterDiskFormat">
    <Path>Fabrikam\FixPartitions.exe</Path>
    <Duration>2</Duration>
  </Run>
  <Run Phase="FactoryReset_AfterImageApply">
    <Path>Fabrikam\InstallDrivers.cmd</Path>
    <Param>/allDrivers</Param>
    <Duration>2</Duration>
  </Run>
  <SystemDisk>
    <MinSize>75000</MinSize>
    <DiskpartScriptPath>Fabrikam>CreatePartition.txt </DiskpartScriptPath>
    <OSPartition>4</OSPartition>
    <RestoreFromIndex>2</RestoreFromIndex>
    <WindowsREPartition>1</WindowsREPartition>
    <WindowsREPath>Recovery\WindowsRE</WindowsREPath>
    <Compact>False</Compact>
  </SystemDisk>
</Reset>
```

## Related topics

[Push-Button Reset Overview](#)

[Create Media to Run Push-Button Reset Features](#)

# WinREConfig XML reference

1/18/2019 • 2 minutes to read

This reference describes all xml elements that are used to author the winreconfig.xml file, which enables you to define custom support and diagnostic tools within WinRE.

## Example WinREConfig.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<!-- WinREConfig.xml -->
<Recovery>
    <RecoveryTools>
        <RelativeFilePath>OEMDiagnostics.exe</RelativeFilePath>
        <CommandLineParam>/param1 /param2</CommandLineParam>
        <RequireLogonOnReset></RequireLogonOnReset>
    </RecoveryTools>
</Recovery>
```

## Description of available elements

ELEMENT	DESCRIPTION
RelativeFilePath	The relative path from the \Tools subfolder of the folder in which Recenv.exe resides. For example, if Recenv.exe is located in X:\WinRE, the path designated in this field is relative to X:\WinRE\tools.
CommandLineParam	Optional parameters that are used when running the custom tool.
RequireLogonOnReset	When present, "Reset this PC" -> "Remove everything" prompts users to choose an account and enter a password.

## Create WinREConfig.xml

1. Create a tools subdirectory in your working Windows RE directory. For example, in `Sources\Recovery\Tools`.
2. Create an .xml file called `winreconfig.xml`.
3. Copy the following, and paste it into `winreconfig.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- WinREConfig.xml -->
<Recovery>
    <RecoveryTools>
        <RelativeFilePath></RelativeFilePath>
        <CommandLineParam></CommandLineParam>
        <RequireLogonOnReset></RequireLogonOnReset>
    </RecoveryTools>
</Recovery>
```

4. Configure the elements in the xml file.

5. Save the file in your working Windows RE directory under `Sources\Recovery\Tools`.

# Windows RE troubleshooting features

1/18/2019 • 4 minutes to read

If a Windows device can't start, it automatically fails over to the Windows Recovery Environment (Windows RE). The Automatic Repair tool in Windows RE automates the diagnosis and repair of an unbootable Windows installation. Windows RE is also a starting point for several tools for manual system recovery. This topic describes the automatic failover behavior, manual diagnosis, and repair process in Windows RE.

## Recovering from startup failures

If the system detects a boot failure on a computer running Windows, the system automatically fails over into the on-disk Windows RE tool. At startup, the Windows loader sets a status flag to indicate that the boot process has started. Windows typically clears this flag before the Windows logon screen appears. However, if the boot attempt fails, Windows doesn't clear the flag. The next time that the computer starts, the loader detects the flag and assumes that a boot failure occurred. When this occurs, the loader starts Windows RE instead of Windows.

**Note** Boot failure detection relies on boot completion and not whether an error occurred in Windows 8. For example, a false positive may occur if power is lost during the boot process, and your user starts Windows RE even though the Windows installation is bootable.

Because the failover mechanism relies on the Windows boot manager and the Windows boot loader, some failures can make Windows RE inaccessible. In the following scenarios, your user must use the bootable Windows RE media to recover the computer:

- Corrupt disk metadata exists in the master boot record (MBR), partition table, or boot sector of a Windows RE partition.
- The boot manager is missing or corrupted.
- The Boot Configuration Data (BCD) store is missing or corrupted.

If the boot loader can't read or write to the boot status flag, Windows won't be able to automatically fail over into Windows RE. However, your user can still manually start the on-disk Windows RE tool through the **Boot Options** menu.

## Advanced troubleshooting utilities in Windows RE

Your user can manually start several system recovery tools after starting the on-disk Windows RE tool from the recovery media, or from the **Boot Options** menu. With the exception of Automatic Repair, the Windows Assessment and Deployment Kit (Windows ADK) doesn't include these tools. Push-button reset is the recommended recovery solution in Windows.

### Automatic Repair

The Automatic Repair tool automates common diagnostic and repair tasks for non-bootable operating system installations. Automatic Repair starts if the computer fails over into Windows RE because of a detected boot failure. If automatic failover to an on-disk instance of Windows RE is not available, your users can also start Automatic Repair as a manual recovery tool from a Windows RE CD or DVD.

### System Image Recovery

Use System Image Recovery for file backup and system image backup. System Image Recovery requires an external storage device. For file backup, your users can let Windows choose what to back up, or they can select individual folders, libraries, and drives. By default, backups are created on a regular schedule. Your users can

change the schedule and manually create a backup at any time. After your user sets up System Image Recovery, Windows keeps track of the new or modified files and folders, adding them to the backup.

For system image backup, your users can create a system image or an exact image of a drive. A system image includes Windows and system settings, programs, and files. Your users can use a system image to restore the contents of their computer if the hard disk drive or computer stops working.

If your users restore their computer from a system image, the restoration is a complete restoration. Your users can't choose individual items to restore. All of the current programs, system settings, and files are replaced.

If you set up a scheduled file backup, you can include a system image with only the drives Windows requires to run. You can manually create a system image if you want to include additional data drives.

**Note** Previous system image versions are copies of the files and folders saved automatically by Windows as part of the system protection process. Depending on the type of file or folder, your users can open a previous version, save the version to a different location, or restore a previous version. Your users can use these previous versions to restore accidentally modified, deleted, or damaged files or folders. However, because Windows replaces these files with new versions the files won't be available if the drive fails.

## Command Prompt

All Windows PE command-line tools are available from a command prompt window. For example, you can use Registry Editor (Regedit.exe), which includes command-line switches, to modify the Windows registry. Or, you can use the Chkdsk.exe tool to troubleshoot and fix volumes. For more information, see [Registry Editor](#), [Chkdsk](#), and [Troubleshooting Tools and Strategies](#).

## Custom Support and Recovery Tools

Computer manufacturers can provide custom support and recovery tools. These tools will vary by manufacturer. For more information, see the manufacturer-provided documentation.

## Related topics

[BCDboot Command-Line Options](#)

[REAgentC Command-Line Options](#)

# Deployment tools reference

1/18/2019 • 2 minutes to read

This section covers information about Windows deployment tools and environments.

## In this section

TOPIC	DESCRIPTION
<a href="#">DISM - Deployment Image Servicing and Management</a>	Reference information for DISM, including command-line options
<a href="#">Windows PE (WinPE)</a>	Windows PE reference
<a href="#">Windows Setup</a>	Windows Setup reference, including command-line options
<a href="#">Command-Line Tools</a>	BCDEdit, Bootsect, and Oscdimg command-line reference

# DISM - Deployment Image Servicing and Management

1/18/2019 • 2 minutes to read

Deployment Image Servicing and Management (DISM) is a command-line tool that is used to mount and service Windows images before deployment. You can use DISM image management commands to mount and get information about Windows image (.wim) files or virtual hard disks (VHD). You can also use DISM to capture, split, and otherwise manage .wim files.

You can use DISM to install, uninstall, configure, and update Windows features, packages, drivers, and international settings in a .wim file or VHD using the DISM servicing commands.

DISM commands are used on offline images, but subsets of the DISM commands are also available for servicing a running operating system.

DISM is installed with Windows, and it is also distributed in the Windows Assessment and Deployment Kit (Windows ADK). DISM replaces several deployment tools, including PEImg, Intlcfg, ImageX, and Package Manager.

## In This Section

<a href="#">DISM overview</a>	Describes how you can use DISM, where to find DISM, and what's new in DISM.
<a href="#">DISM How-to Topics (Deployment Image Servicing and Management)</a>	Provides how-to instructions on using DISM.
<a href="#">DISM Reference (Deployment Image Servicing and Management)</a>	Provides reference information for DISM, including command-line options, best practices, and supported platforms.

## Related topics

[Windows Setup Technical Reference](#)

[Device Drivers and Deployment Overview](#)

[Language Packs](#)

[Understanding Servicing Strategies](#)

# DISM Overview

1/18/2019 • 4 minutes to read

## What's New in DISM

DISM in Windows 10 supports new features:

- **Full Flash Update (.FFU):** DISM supports the Full Flash Update (.FFU) format, which captures and applies an entire drive, including partition information. This can make deployment faster and easier. See [Windows full flash update images](#) for more info.
- **Capabilities:** This new Windows package type allows you to request services like .NET or languages without specifying the version. Use DISM to search multiple sources like Windows Update or your corporate servers to find and install the latest version. For more info, see [Features on Demand](#).
- **Compress operating system and provisioning packages:** Save space on a Windows image by running the operating system and other system files from compressed files. This replaces the WIMBoot features from Windows 8.1. See `/Apply-Image /Compact` and `/Apply-CustomDataImage` in [DISM Image Management Command-Line Options](#) for more info.

## What is DISM?

Deployment Image Servicing and Management (DISM.exe) is a command-line tool that can be used to service and prepare Windows images, including those used for [Windows PE](#), [Windows Recovery Environment \(Windows RE\)](#) and [Windows Setup](#). DISM can be used to service a Windows image (.wim) or a virtual hard disk (.vhdx or .vhd).

DISM comes built into Windows and is available through the command line or from Windows PowerShell. To learn more about using DISM with PowerShell, see [Deployment Imaging Servicing Management \(DISM\) Cmdlets in Windows PowerShell](#).

### Image Requirements

DISM can be used to mount and service a Windows image from a .wim file, .ffu file, .vhdx file, or a .vhd file and also to update a running operating system. It can be used with older Windows image files (.wim files). However, it cannot be used with Windows images that are more recent than the installed version of DISM.

For a complete technical description of WIM, see the [Windows Imaging File Format \(WIM\) white paper](#).

DISM can be used to service Windows images starting with Windows 7, Windows Server 2008 R2, and their associated versions of [WinPE](#) and [WinRE](#). The commands and options that are available for servicing an image depend on which Windows operating system you are servicing, and whether the image is offline or a currently running operating system.

For a list of the supported platforms and architecture types, see [DISM Supported Platforms](#).

### Common Servicing and Management Scenarios

Image servicing and management solutions fall into two main categories:

- Managing the data or information included in the Windows image, such as enumerating or taking an inventory of the components, updates, drivers, or applications that are contained in an image, capturing or splitting an image, appending or deleting images within a .wim file, or mounting an image.
- Servicing the image itself, including adding or removing driver packages and drivers, modifying language settings, enabling or disabling Windows features, and upgrading to a higher edition of Windows.

## Limitations

**Version compatibility.** DISM can be used with target images of older Windows operating systems, but not with target images of operating systems that are more recent than the installed version of DISM. For example, DISM from Windows 10, version 1511 can service Windows 10, version 1511 and version 1507 but not version 1607. To learn more, see [DISM Supported Platforms](#).

**Remote installation.** Installing packages to a remote computer over a network is not supported.

**Answer files.** When you specify an answer file (Unattend.xml) for an image, only the settings specified in the `offlineServicing` configuration pass are applied. All other settings in the answer file are ignored. You can only specify one answer file at a time. For more information, see [DISM Unattended Servicing Command-Line Options](#)

**Use an answer file to ensure package dependencies.** Some packages require other packages to be installed first. Because of this dependency requirement, you should use an answer file if you are installing multiple packages. By applying an answer file by using DISM, multiple packages can be installed in the correct order. This is the preferred method for installing multiple packages.

**Package installation order.** Packages are installed in the order that they are listed in the command line. In the following example, 1.inf, 2.inf, and 3.inf will be installed in the order in which they are listed in the command line.

```
DISM.exe /image:"c:\images\Image1" /Add-Driver /ForceUnsigned /DriverName:"C:\Drivers\1.inf"
/DriverName:"C:\Drivers\2.inf" /DriverName:"C:\Drivers\3.inf"
```

**Multiple servicing commands are not supported.** You can specify multiple drivers (1.inf, 2.inf) or packages, but you cannot specify multiple commands (such as **/Add-Driver /Remove-Driver** or **/Add-Driver /Add-Package**) on the same command line.

**Wildcards.** Wildcards are not supported in DISM command lines.

## Where is DISM?

Deployment Image Servicing and Management (DISM.exe) is a command-line tool that can be used to service a Windows image or to prepare a Windows Preinstallation Environment (Windows PE) image. For more information about DISM see [What is DISM?](#)

### DISM in Windows 10

DISM comes with Windows 10, in the `c:\windows\system32` folder, but you can run DISM in the command prompt from any location. You have to be running from a Command Prompt as administrator to use DISM.

### DISM in the ADK

If you are running an older version of Windows, or you need a different version of DISM on your PC, download and install the Windows Assessment and Deployment Kit (Windows ADK), see [Windows Assessment and Deployment Kit \(Windows ADK\) Technical Reference](#).

DISM appears in the Windows ADK here:

```
C:\Program Files (x86)\Windows Kits\<version>\Assessment and Deployment Kit\Deployment Tools\<arch>\DISM
```

where `<version>` can be **8.0**, **8.1**, or **10**, and `<arch>` can be **x86** or **amd64**.

If you need to copy an ADK version of DISM to a PC that does not have the ADK, see [Copy DISM to another computer](#).

## Related topics

[DISM Reference \(Deployment Image Servicing and Management\)](#)

[Deployment Image Servicing and Management \(DISM\) Command-Line Options](#)

[Device Drivers and Deployment Overview](#)

[Language Packs](#)

[Understanding Servicing Strategies](#)

# Use DISM in Windows PowerShell

1/18/2019 • 2 minutes to read

The Deployment Image Servicing and Management (DISM) cmdlets can be used to perform the same functions as the DISM.exe command-line tool. In many cases, the DISM cmdlet names are similar to their corresponding [Dism.exe options](#) and the same arguments can be used. See [DISM PowerShell](#) for detailed information about the PowerShell cmdlets.

## DISM command to DISM cmdlet mapping table

DISM cmdlet names don't always correspond directly to Dism.exe options. Here is a table that maps DISM commands to their corresponding PowerShell cmdlets.

DISM.EXE COMMAND	DISM CMDLET
Dism.exe /Add-Capability	Add-WindowsCapability
Dism.exe /Append-Image	Add-WindowsImage
Dism.exe /Apply-Image	Expand-WindowsImage
Dism.exe /Capture-Image	New-WindowsImage
Dism.exe /Cleanup-MountPoints	Clear-WindowsCorruptMountPoint
Dism.exe /Commit-Image	Save-WindowsImage
Dism.exe /Export-Image	Export-WindowsImage
Dism.exe /Get-Capabilities	Get-WindowsCapability
Dism.exe /Get-ImageInfo	Get-WindowsImage
Dism.exe /Get-MountedImageInfo	Get-WindowsImage -Mounted
Dism.exe /Get-WimBootEntry	Get-WIMBootEntry
Dism.exe /List-Image	Get-WindowsImageContent
Dism.exe /Mount-Image	Mount-WindowsImage
Dism.exe /Split-Image	Split-WindowsImage
Dism.exe /Remove-Capability	Remove-WindowsCapability
Dism.exe /Remove-Image	Remove-WindowsImage
Dism.exe /Remount-Image	Mount-WindowsImage -Remount

DISM.EXE COMMAND	DISM CMDLET
Dism.exe /Unmount-Image	Dismount-WindowImage
Dism.exe /Update-WimBootEntry	Update-WIMBootEntry
Dism.exe /Image:<...> /Add-Driver	Add-WindowsDriver
Dism.exe /Image:<...> /Add-Package	Add-WindowsPackage
Dism.exe /Image:<...> /Add-ProvisionedAppxPackage	Add-AppxProvisionedPackage
Dism.exe /Image:<...> /Cleanup-Image /CheckHealth	Repair-WindowImage -CheckHealth
Dism.exe /Image:<...> /Cleanup-Image /ScanHealth	Repair-WindowImage -ScanHealth
Dism.exe /Image:<...> /Cleanup-Image /RestoreHealth	Repair-WindowImage -RestoreHealth
Dism.exe /Image:<...> /Disable-Feature	Disable-WindowsOptionalFeature
Dism.exe /Image:<...> /Enable-Feature	Enable-WindowsOptionalFeature
Dism.exe /Image:<...> /Export-Driver	Export-WindowsDriver
Dism.exe /Image:<...> /Get-CurrentEdition	Get-WindowsEdition -Current
Dism.exe /Image:<...> /Get-Driverinfo	Get-WindowsDriver -Driver
Dism.exe /Image:<...> /Get-Drivers	Get-WindowsDriver
Dism.exe /Image:<...> /Get-Featureinfo	Get-WindowsOptionalFeature -FeatureName
Dism.exe /Image:<...> /Get-Features	Get-WindowsOptionalFeature
Dism.exe /Image:<...> /Get-Packageinfo	Get-WindowsPackage -PackagePath
Dism.exe /Image:<...> /Get-Packages	Get-WindowsPackage
Dism.exe /Image:<...> /Get-ProvisionedAppxPackages	Get-AppxProvisionedPackage
Dism.exe /Image:<...> /Get-TargetEditions	Get-WindowsEdition -Target
Dism.exe /Image:<...> /Optimize-Image	Optimize-WindowImage
Dism.exe /Image:<...> /Remove-Driver	Remove-WindowsDriver
Dism.exe /Image:<...> /Remove-Package	Remove-WindowsPackage
Dism.exe /Image:<...> /Remove-ProvisionedAppxPackage	Remove-AppxProvisionedPackage
Dism.exe /Image:<...> /Set-Edition	Set-WindowsEdition

DISM.EXE COMMAND	DISM CMDLET
Dism.exe /Image:<...> /Set-ProductKey	<a href="#">Set-WindowsProductKey</a>
Dism.exe /Image:<...> /Set-ProvisionedAppxDataFile	<a href="#">Set-AppXProvisionedDataFile</a>

## Use DISM Powershell modules in WinPE

You can add Powershell support to WinPE. Visit [Adding PowerShell support to Windows PE](#) to learn how.

## Use DISM PowerShell modules in previous versions of Windows

The DISM PowerShell module is included in Windows 10 and Windows Server 2016. On other supported operating systems, you can install the Windows Assessment and Deployment Kit (ADK) which includes the DISM PowerShell module. See [the previous version of this topic for more information](#).

## Related topics

[DISM - Deployment Image Servicing and Management Technical Reference for Windows](#)

[DISM Supported Platforms](#)

# Windows 10 DISM Command-Line Options

1/18/2019 • 2 minutes to read

Deployment Image Servicing and Management (DISM.exe) mounts a Windows image (.wim) file or virtual hard disk (.vhd or .vhdx) for servicing. You can also use DISM to install, uninstall, configure, and update the features and packages in offline Windows images and offline Windows Preinstallation Environment (WinPE) images. For more information about common DISM scenarios, see [What is DISM?](#).

In addition to the command-line tool, DISM is available by using PowerShell. For more information, see [Deployment Imaging Servicing Management \(DISM\) Cmdlets in Windows PowerShell](#).

DISM replaces tools including PEImg, Intlcfg, Package Manager, and ImageX.

## In This Section

<a href="#">DISM Image Management Command-Line Options</a>	Image management commands such as capturing, applying, and mounting a Windows image.
<a href="#">DISM Global Options for Command-Line Syntax</a>	Basic command-line syntax and universal options for servicing functions.
<a href="#">DISM Operating System Package Servicing Command-Line Options</a>	Package-servicing commands for adding, removing, and enumerating .cab and .msu packages and enabling, disabling, and enumerating features.
<a href="#">DISM Provisioning Package (.ppkg) Command-Line Options</a>	Use Windows provisioning packages (.ppkg)
<a href="#">DISM Capabilities Package Servicing Command-Line Options</a>	Capabilities servicing commands for adding languages, .NET, and other Windows features.
<a href="#">DISM App Package (.appx or .appxbundle) Servicing Command-Line Options</a>	Servicing commands for adding, removing, and enumerating app packages.
<a href="#">DISM Application Servicing Command-Line Options</a>	Servicing commands that can be used to check the applicability of Windows Installer application patches (.msp files) and to query your offline image for information about installed MSI applications and application patches (.msp files).
<a href="#">DISM Default Application Association Servicing Command-Line Options</a>	Servicing commands for importing, exporting, removing, and enumerating the settings that specify which application opens a file based on file extension or protocol
<a href="#">DISM Languages and International Servicing Command-Line Options</a>	International-servicing commands for adjusting international settings and configurations.

<a href="#">DISM Driver Servicing Command-Line Options</a>	Driver-specific servicing commands for adding, removing, and enumerating driver .inf files.
<a href="#">DISM Unattended Servicing Command-Line Options</a>	Servicing commands that can be used to apply an Unattend.xml file.
<a href="#">DISM Windows PE Servicing Command-Line Options</a>	WinPE-specific servicing commands for preparing a WinPE image.
<a href="#">DISM Windows Edition-Servicing Command-Line Options</a>	Edition-servicing commands for changing the edition of your Windows image.
<a href="#">DISM Operating System Uninstall Command-Line Options</a>	Commands for working with operating system uninstalls.

## Related topics

[DISM Image Management Command-Line Options](#)

[DISM How-to Topics \(Deployment Image Servicing and Management\)](#)

[What is DISM?](#)

# DISM Image Management Command-Line Options

1/18/2019 • 20 minutes to read

Deployment Image Servicing and Management (DISM.exe) mounts a Windows image (.wim) file or virtual hard disk (.vhdx or .vhd) for servicing. You can also use the DISM image management command to list the image index numbers, to verify the architecture for the image that you are mounting, append an image, apply an image, capture an image and delete an image. After you update the image, you must unmount it and either commit or discard the changes that you have made.

This topic discusses DISM commands related to image management. To see other command-line options, see [Deployment Image Servicing and Management \(DISM\) Command-Line Options](#). For more information about common DISM scenarios, see [What is DISM?](#).

In addition to the command-line tool, DISM is available by using Windows PowerShell. For more information, see [Deployment Imaging Servicing Management \(DISM\) Cmdlets in Windows PowerShell](#).

The following commands can be used to mount, unmount, capture, append, and delete and query .wim, .vhdx and .vhd files. These options are not case sensitive.

## /Append-Image

Adds an additional image to a .wim file. **/Append-Image** compares new files to the resources in the existing .wim file specified by the **/ImageFile** argument, and stores only a single copy of each unique file so that each file is only captured once. The .wim file can have only one assigned compression type. Therefore, you can only append files with the same compression type.

This command-line option does not apply to virtual hard disk (VHD) files.

### IMPORTANT

Ensure that you have enough disk space for the **/Append-Image** option to run. If you run out of disk space while the image is being appended, you might corrupt the .wim file.

Syntax:

```
DISM.exe /Append-Image /ImageFile:<path_to_image_file> /CaptureDir:<source_directory> /Name:<image_name> [/Description:<image_description>] [/ConfigFile:<configuration_file.ini>] [/Bootable] [/WIMBoot] [/CheckIntegrity] [/Verify] [/NoRpFix]
```

PARAMETER	DESCRIPTION
/WIMBoot	Use /WIMBoot to append the image with Windows image file boot (WIMBoot) configuration. This only applies to Windows 8.1 images that have been captured or exported as a WIMBoot file. This feature isn't supported in Windows 10.

PARAMETER	DESCRIPTION
/ConfigFile	specifies the location of a configuration file that lists exclusions for image capture and compress commands. For more information, see <a href="#">DISM Configuration List and WimScript.ini Files</a> .
/Bootable	Marks a volume image as being a bootable image. This argument is available only for Windows Preinstallation Environment (WinPE) images. Only one volume image can be marked as bootable in a .wim file.
/CheckIntegrity	Detects and tracks .wim file corruption when used with capture, unmount, export, and commit operations. /CheckIntegrity stops the operation if DISM detects that the .wim file is corrupted when used with apply and mount operations.
/Verify	Checks for errors and file duplication.
/NoRpFix	Disables the reparse point tag fix. A reparse point is a file that contains a link to another file on the file system. If /NoRpFix is not specified, reparse points that resolve to paths outside of the value specified by /ImageFile will not be captured.

Example:

```
Dism /Append-Image /ImageFile:install.wim /CaptureDir:D:\ /Name:Drive-D
```

## /Apply-FFU

For FFU, this command applies a Full Flash Utility (FFU) or split FFU (SFU) to a specified physical drive.

Syntax:

```
/Apply-Ffu /ImageFile:<path_to_image_file> /ApplyDrive:<physical_drive_path> [/SFUFile:<pattern>]
```

PARAMETER	DESCRIPTION
/ImageFile	The path and name of the FFU image file that will be applied
/ApplyDrive	The path to the physical drive that will be imaged
/SFUfile<pattern>	Optional, for split FFUs that are captured with no compression. Use /SFUFile to reference split FFU files (SFUs). <i>Pattern</i> is the naming pattern and location of split files. Use a wildcard character when specifying the naming pattern. For example, "E:\image\install*.sfu" will apply all of the split files in the E:\image directory named install1.sfu, install2.sfu, and so on.

Example:

```
DISM.exe /Apply-Ffu /ImageFile:flash.ffu /ApplyDrive:\\.\PhysicalDrive0
```

## /Apply-Image

For WIM, this command applies a Windows image file (.wim) or a split Windows image (.swm) files to a specified partition. Beginning with Windows 10, version 1607, DISM can apply and capture extended attributes (EA).

For FFU, this command applies a full flash update (.ffu) image to a specified drive. It doesn't support applying an image from a virtual hard disk (.vhdx) file, though you can use this command to apply a full image to a VHD. FFU applies to Windows 10 only.

This option doesn't support applying an image from a virtual hard disk (VHD), though you can use this command to apply images to a .vhdx file that's been attached, partitioned, and formatted.

If Dism /Apply-Image fails with error code 5 and you are using Windows 10 version 1607 with Windows Subsystem for Linux (WSL) feature, see [KB article 319598](#).

Arguments for WIM:

```
DISM.exe /Apply-Image /ImageFile:<path_to_image_file> [/SWMFile:<pattern>] /ApplyDir:<target_directory>
{/Index:< image_index> | /Name:<image_name>} [/CheckIntegrity] [/Verify] [/NoRpFix]
[/ConfirmTrustedFile] [/WIMBoot (deprecated)] [/Compact] [/EA]
```

Arguments for FFU

```
DISM.exe /Apply-Image /ImageFile:<path_to_image_file> /ApplyDrive:<target_drive> [/SFUFile:<pattern>]
/index:1
```

PARAMETER	DESCRIPTION
/CheckIntegrity	Detects and tracks .wim file corruption when used with capture, unmount, export, and commit operations. /CheckIntegrity stops the operation if DISM detects that the .wim file is corrupted when used with apply and mount operations.
/Verify	Checks for errors and file duplication.
/NoRpFix	Disables the reparse point tag fix. A reparse point is a file that contains a link to another file on the file system. If /NoRpFix is not specified, reparse points that resolve to paths outside the value specified by /ImageFile will not be captured.
/SWMFile	Enables you to reference split .wim files (SWMs). <i>Pattern</i> is the naming pattern and location of split files. Use a wildcard character when specifying the naming pattern. For example, "E:\image\install*.swm" will apply all of the split files in the E:\image directory named install1.swm, install2.swm, and so on.

PARAMETER	DESCRIPTION
/ConfirmTrustedFile	Validates the image for Trusted Desktop on a Windows 10, Windows 8.1, or Windows 8. This option can only be run on a computer running at least WinPE 4.0. When using /Apply-Image with the /ConfirmTrustedFile option in WinPE, always specify the /ScratchDir option pointed to a physical media location. This ensures that short file names will always be available. See <a href="#">DISM Global Options for Command-Line Syntax</a> for more information about the default behavior of the /ScratchDir option. Beginning with Windows 10, version 1607, you can use /EA to apply extended attributes.
/WIMBoot	Use /WIMBoot to append the image with Windows image file boot (WIMBoot) configuration. This only applies to Windows 8.1 images that have been captured or exported as a WIMBoot file. This feature isn't supported in Windows 10.
/Compact	<p>Applies an image in compact mode, saving drive space. Replaces WIMBoot. For Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) only.</p> <p><b>Note:</b> If you're applying an image in compact mode with the /ScratchDir option, make sure your ScratchDir folder is not on a FAT32-formatted partition. Using a FAT32 partition could result in unexpected reboots during OOBE.</p>
/EA	New in Windows 10, version 1607. Applies extended attributes.
/ApplyDrive	Specifies the logical drive, using the DeviceID. to get the device ID from the command line, type "wmic diskdrive list brief". Note: a VHD may appear with the name "PhysicalDrive" in the description, for example, .\PhysicalDrive2.
/SFUFile	Use /SFUFile to reference split FFU files (SFUs). <i>Pattern</i> is the naming pattern and location of split files.

Examples:

```
Dism /apply-image /imagefile:install.wim /index:1 /ApplyDir:D:\
```

```
Dism /apply-image /imagefile:install.swm /swmfile:install.swm /index:1 /applydir:D:\
```

```
DISM.exe /Apply-image /ImageFile:flash.ffu /ApplyDrive:\\.\PhysicalDrive0 /index:1
```

```
DISM.exe /Apply-image /ImageFile:flash.sfu /SFUFile:flash*.sfu /ApplyDrive:\\.\PhysicalDrive0 /index:1
```

## /Capture-CustomImage

Captures the incremental file changes based on the specific install.wim file to a new file, custom.wim for a WIMBoot image. You can't capture an empty directory. The captured files are converted to pointer files. The custom.wim is placed in the same folder next to the install.wim.

### Important

- /Capture-CustomImage only captures the customization files. It can't be used to capture installation files into a new WIM.
- Keep the install.wim and custom.wim files together. Don't switch out either the custom.wim file or the install.wim file.
- You can only capture the custom image once. Don't remove or recapture a custom.wim after capturing the incremental file changes.

Syntax:

```
Dism /Capture-CustomImage /CaptureDir:<source_directory> [/ConfigFile:<configuration_file.ini>]  
[/CheckIntegrity] [/Verify] [/ConfirmTrustedFile]
```

PARAMETER	DESCRIPTION
/CaptureDir	Specifies the directory to which the image was applied and customized.
/ConfigFile	Specifies the location of a configuration file that lists exclusions for image capture and compress commands. For more information, see <a href="#">DISM Configuration List and WimScript.ini Files</a> .
/CheckIntegrity	Detects and tracks .wim file corruption when used with capture, unmount, export, and commit operations. /CheckIntegrity stops the operation if DISM detects that the .wim file is corrupted when used with apply and mount operations.
/Verify	Checks for errors and file duplication.
[/ConfirmTrustedFile	Validates the image for Trusted Desktop on a Windows 10, Windows 8.1, or Windows 8. This option can only be run on a computer running at least WinPE 4.0.

Example:

```
Dism /Capture-CustomImage /CaptureDir:D:\
```

## /Capture-FFU

Captures an image of a physical drive's partitions to a new .ffu file.

You can capture the image as a full flash utility image (.ffu) file or a set of split ffu (.sfu) files;

Syntax:

```
Dism /Capture-Ffu /ImageFile:<path_to_image_file> /CaptureDrive:<physical_drive_path> /Name:<image_name> [/Description:<image_description>] [/PlatformIds:<platform_ids>] [/Compress:{default|none}]
```

PARAMETER	DESCRIPTION
/CaptureDrive	The physical drive to be captured. You can <a href="#">use diskpart to get drive number information</a> . Uses the format <code>\.\PhysicalDriveX</code> , where X is the disk number that diskpart provides.
/PlatformIds	Not needed for desktop capture. Specifies one or more platform ids (separated with semicolon) to be added to the image. If not specified, platform id will be '*'.
/Compress	Specifies the type of compression used for when capturing. If you'll be splitting the FFU, specify <code>none</code> , as DISM doesn't support splitting compressed FFUs.

Examples:

Capture a desktop FFU:

```
DISM.exe /Capture-Ffu /ImageFile:install.ffd /CaptureDrive:\.\PhysicalDrive0 /Name:Drive0
```

Capture a desktop FFU that will be split:

```
DISM.exe /Capture-Ffu /ImageFile:install.ffd /CaptureDrive:\.\PhysicalDrive0 /Name:Drive0
/Compress:none
```

## /Capture-Image

Captures an image of a drive to a new .wim file. Captured directories include all subfolders and data. You cannot capture an empty directory. A directory must contain at least one file.

You can capture the image as a Windows image (.wim) file or a set of split Windows image (.swm) files; this option doesn't support capturing a virtual hard disk (.vhd/.vhdx) file or a full flash update (.ffd) image.

Beginning with Windows 10, version 1607, DISM can apply and capture extended attributes (EA).

Syntax:

```
Dism /Capture-Image /ImageFile:<path_to_image_file> /CaptureDir:<source_directory> /Name:<image_name>
[/Description:<image_description>]
[/ConfigFile:<configuration_file.ini>] {[/Compress:{max|fast|none}] [/Bootable] | [/WIMBoot]}
[/CheckIntegrity] [/Verify] [/NoRpFix] [/EA]
```

PARAMETER	DESCRIPTION
/ConfigFile	Specifies the location of a configuration file that lists exclusions for image capture and compress commands. For more information, see <a href="#">DISM Configuration List and WimScript.ini Files</a> .

PARAMETER	DESCRIPTION
/Compress	Specifies the type of compression used for the initial capture operation. The <b>maximum</b> option provides the best compression, but takes more time to capture the image. The <b>fast</b> option provides faster image compression, but the resulting files are larger than those compressed by using the maximum option. This is also the default compression type that is used if you do not specify the argument. The <b>none</b> option does not compress the captured image at all.
/Bootable	Marks a volume image as being a bootable image. This argument is available only for WinPE images. Only one volume image can be marked as bootable in a .wim file.
/CheckIntegrity	Detects and tracks .wim file corruption when used with capture, unmount, export, and commit operations. /CheckIntegrity stops the operation if DISM detects that the .wim file is corrupted when used with apply and mount operations.
/Verify	Checks for errors and file duplication.
/NoRpFix	Disables the reparse point tag fix. A reparse point is a file that contains a link to another file on the file system. If /NoRpFix is not specified, reparse points that resolve to paths outside of the value specified by /ImageFile will not be captured.
/WIMBoot	Use /WIMBoot to append the image with Windows image file boot (WIMBoot) configuration. This only applies to Windows 8.1 images that have been captured or exported as a WIMBoot file. This feature isn't supported in Windows 10.
/EA	New in Windows 10, version 1607. Captures extended attributes. The switch must be explicitly specified to capture extended attributes. DISM will capture extended attribute bits if they are set in the components to be captured in the WIM image. If the bits are not set, DISM won't set them. Only the inbox components of CAB packages and drivers will have these extended attribute bits, not the AppX package components or Win32 application components. Extended attributes with prefix "\$Kernel." in name will be skipped because only user mode extended attributes are captured. If you use DISM in Windows 10, version 1607 to capture extended attributes and use an earlier version of DISM to apply the image, the operation will succeed but the extended attributes will not be set to the applied image.

Examples:

```
Dism /Capture-Image /ImageFile:install.wim /CaptureDir:D:\ /Name:Drive-D
```

```
dism /Capture-Image /CaptureDir:C:\ /ImageFile:"C:\WindowsWithOffice.wim" /Name:"Chinese Traditional" /ea
```

## /Cleanup-Mountpoints

Deletes all of the resources associated with a mounted image that has been corrupted. This command will not unmount images that are already mounted, nor will it delete images that can be recovered using the **/Remount-Image** command.

Example:

```
Dism /Cleanup-Mountpoints
```

To learn more, see [Repair a Windows Image](#)

## /Commit-Image

Applies the changes that you have made to the mounted image. The image remains mounted until the **/Unmount-Image** option is used.

Syntax:

```
Dism /Commit-Image /MountDir:<path_to_mount_directory> [/CheckIntegrity] [/Append]
```

PARAMETER	DESCRIPTION
/CheckIntegrity	Detects and tracks .wim file corruption when used with capture, unmount, export, and commit operations. /CheckIntegrity stops the operation if DISM detects that the .wim file is corrupted when used with apply and mount operations.
/Append	Adds the modified image to the existing .wim file instead of overwriting the original image. The /CheckIntegrity and /Append arguments do not apply to virtual hard disk (VHD) files.

Example:

```
Dism /Commit-Image /MountDir:C:\test\offline
```

## /Delete-Image

Deletes the specified volume image from a .wim file that has multiple volume images. This option deletes only the metadata entries and XML entries. It does not delete the stream data and does not optimize the .wim file.

This command-line option does not apply to virtual hard disk (VHD) files.

Syntax:

```
Dism /Delete-Image /ImageFile:<path_to_image_file> {/Index:<image_index> | /Name:<image_name>} [/CheckIntegrity]
```

PARAMETER	DESCRIPTION
/CheckIntegrity	Detects and tracks .wim file corruption when used with capture, unmount, export, and commit operations. /CheckIntegrity stops the operation if DISM detects that the .wim file is corrupted when used with apply and mount operations.

Example:

```
Dism /Delete-Image /ImageFile:install.wim /Index:1
```

## /Export-Image

Exports a copy of the specified image to another file. The source and destination files must use the same compression type. You can also optimize an image by exporting to a new image file. When you modify an image, DISM stores additional resource files that increase the overall size of the image. Exporting the image will remove unnecessary resource files.

This command-line option does not apply to virtual hard disk (VHD) files.

Syntax:

```
Dism /Export-Image /SourceImageFile:<path_to_image_file> {/SourceIndex:<image_index> | /SourceName:<image_name>} /DestinationImageFile:<path_to_image_file> [/DestinationName:<Name>] [/Compress:{fast|max|none|recovery}] [/Bootable] [/WIMBoot] [/CheckIntegrity]
```

PARAMETER	DESCRIPTION
/SWMFile	Enables you to reference split .wim files. pattern is the naming pattern and location of split files. You can also specify wildcard characters. For example, "E:\image\install*.swm" will export the split files in the E:\image directory named install1.swm, install2.swm, and so on.
/Compress	Specifies the type of compression used for the initial capture operation. The /Compress argument does not apply when you export an image to an existing .wim file, you can only use this argument when you export an image to a new .wim file. The <b>maximum</b> option provides the best compression, but takes more time to capture the image. The <b>fast</b> option provides faster image compression, but the resulting files are larger than those compressed by using the <b>maximum</b> option. This is also the default compression type that is used if you do not specify the argument. Use the <b>recovery</b> option to export push-button reset images. The resulting files are much smaller in size, which in turn, greatly reduce the amount of disk space needed for saving the push-button reset image on a recovery drive. The destination file must be specified with an .esd extension. The <b>none</b> option does not compress the captured image at all.
/Bootable	Marks a volume image as being a bootable image. This argument is available only for WinPE images. Only one volume image can be marked as bootable in a .wim file.

PARAMETER	DESCRIPTION
/WIMBoot	Use /WIMBoot to append the image with Windows image file boot (WIMBoot) configuration. This only applies to Windows 8.1 images that have been captured or exported as a WIMBoot file. This feature isn't supported in Windows 10.
/CheckIntegrity	Detects and tracks .wim file corruption when used with capture, unmount, export, and commit operations. /CheckIntegrity stops the operation if DISM detects that the .wim file is corrupted when used with apply and mount operations.

Example:

```
Dism /Export-Image /SourceImageFile:install.wim /SourceIndex:1 /DestinationImageFile:install2.wim
```

## /Get-ImageInfo

Displays information about the images that are contained in a .wim, .ffu, .vhdx or .vhd file. When used with the /Index or /Name argument, information about the specified image is displayed, which includes if an image is a WIMBoot image, if the image is Windows 8.1, see [Take Inventory of an Image or Component Using DISM](#). The /Name argument does not apply to VHD files. You must specify /Index:1 for FFU and VHDX files.

Syntax:

```
Dism /Get-ImageInfo /ImageFile:<path_to_image.wim> [{/Index:<Image_index> | /Name:<Image_name>}]
```

Examples:

```
Dism /Get-ImageInfo /ImageFile:C:\test\offline\install.wim
```

```
Dism /Get-ImageInfo /ImageFile:C:\test\images\myimage.vhd /Index:1
```

## /Get-MountedImageInfo

Returns a list of .ffu, .vhdx, .vhd, and .wim images that are currently mounted, as well as information about the mounted image such as whether the image is valid, read/write permissions, mount location, mounted file path, and mounted image index.

Example:

```
Dism /Get-MountedImageInfo
```

## /Get-WIMBootEntry

Use /Get-WIMBootEntry to display WIMBoot configuration entries for the specified disk volume.

For more information about how to display WIMBoot configuration entries, see [Take Inventory of an](#)

Image or Component Using DISM.

This only applies to Windows 8.1; this feature isn't supported in Windows 10.

Syntax:

```
Dism /Get-WIMBootEntry /Path:<volume_path>
```

Example:

```
Dism /Get-WIMBootEntry /Path:C:\
```

## /List-Image

Displays a list of the files and folders in a specified image.

This command-line option does not apply to virtual hard disk (VHD) files.

Syntax:

```
Dism /List-Image /ImageFile:<path_to_image_file> {/Index:<image_index> | /Name:<image_name>}
```

Example:

```
Dism /List-Image /ImageFile:install.wim /Index:1
```

## /Mount-Image

Mounts an image from a .ffu, .wim, .vhdx or .vhd file to the specified directory so that it is available for servicing.

When mounting an image, note the following:

- The mount directory must be created, but empty.
- An index or name value is required for all image types. WIMs can contain more than one image. For FFU and VHD, use `index:1`.

Syntax:

```
Dism /Mount-Image /ImageFile:<path_to_image_file> {/Index:<image_index> | /Name:<image_name>}  
/MountDir:<path_to_mount_directory> [/ReadOnly] [/Optimize] [/CheckIntegrity]
```

PARAMETER	DESCRIPTION
/ReadOnly	Sets the mounted image with read-only permissions. Optional.
/Optimize	Reduces initial mount time.

PARAMETER	DESCRIPTION
/CheckIntegrity	Detects and tracks .wim file corruption when used with capture, unmount, export, and commit operations. /CheckIntegrity stops the operation if DISM detects that the .wim file is corrupted when used with apply and mount operations.

Examples:

```
Dism /Mount-Image /ImageFile:C:\test\images\myimage.wim /index:1 /MountDir:C:\test\offline
```

```
Dism /Mount-Image /ImageFile:C:\test\images\myimage.vhd /index:1 /MountDir:C:\test\offline /ReadOnly
```

```
Dism /Mount-Image /ImageFile:C:\test\images\WinOEM.ffa /MountDir:C:\test\offline /index:1
```

## /Optimize-Image /WIMBoot

Performs specified configurations to an offline image.

PARAMETER	DESCRIPTION
/WIMBoot	configure an offline image for installing on a Windows image file boot (WIMBoot) system.
/Optimize	Reduces initial mount time. /Optimize-Image /WIMBoot only applies to Windows 8.1 images that have been captured or exported as a WIMBoot file. Only use /Optimize-Image with images that will be used for WIMBoot supported systems. If /Optimize-Image is used with a non-WIMBoot supported system image, Windows may not work as expected, after installation on a non-WIMBoot supported device.

Example:

```
Dism /Image:C:\test\offline /Optimize-Image /WIMBoot
```

## /Remount-Image

Remounts a mounted image that has become inaccessible and makes it available for servicing.

Syntax:

```
Dism /Remount-Image /MountDir:<path_to_mount_directory>
```

Example:

```
Dism /Remount-Image /MountDir:C:\test\offline
```

## /Split-FFU

For FFU, this command splits an existing full-flash update (.ffu) file into multiple read-only split .sfu files. DISM doesn't support splitting compressed FFUs. If you are splitting FFUs, make sure that your FFU was captured with the `/compress:none` option specified.

This option creates the .sfu files in the specified directory, naming each file the same as the specified /SFUFile, but with an appended number. For example, if you use `c:\flash.sfu`, you'll get a flash.sfu file, a flash2.ffu file, a flash3.sfu file, and so on, defining each portion of the split .sfu file and saving it to the C:\ directory.

Syntax for FFU:

```
Dism /Split-FFu /ImageFile:<path_to_image_file> /SFUFile:<pattern> /FileSize:<MB-Size>
[/CheckIntegrity]
```

PARAMETER	DESCRIPTION
<code>/FileSize</code>	Specifies the maximum size in megabytes (MB) for each created file. If a single file is larger than the value specified in the <code>/FileSize</code> option, one of the split .swm files that results will be larger than the value specified in the <code>/FileSize</code> option, in order to accommodate the large file.
<code>/CheckIntegrity</code>	Detects and tracks .wim file corruption when used with capture, unmount, export, and commit operations. <code>/CheckIntegrity</code> stops the operation if DISM detects that the .wim file is corrupted when used with apply and mount operations.
<code>/ImageFile</code>	Specifies the path of a .FFU file, example: flash.ffu.
<code>/SFUFile</code>	References split FFU files (SFUs). <i>Pattern</i> is the naming pattern and location of split files.

Example:

```
DISM.exe /Split-FFu /ImageFile:flash.ffu /SFUFile:flash.sfu /FileSize:650
```

## /Split-Image

For WIM, this command splits an existing .wim file into multiple read-only split .swm files.

This option creates the .swm files in the specified directory, naming each file the same as the specified *path\_to\_swm*, but with an appended number. For example, if you set *path\_to\_swm* as `c:\Data.swm`, this option creates a Data.swm file, a Data2.swm file, a Data3.swm file, and so on, defining each portion of the split .wim file and saving it to the C:\ directory.

This command-line option does not apply to virtual hard disk (VHD) files.

Syntax for WIM:

```
Dism /Split-Image /ImageFile:<path_to_image_file> /SWMFile:<path_to_swm> /FileSize:<MB-Size>
[/CheckIntegrity]
```

PARAMETER	DESCRIPTION
/FileSize	Specifies the maximum size in megabytes (MB) for each created file. If a single file is larger than the value specified in the /FileSize option, one of the split .swm files that results will be larger than the value specified in the /FileSize option, in order to accommodate the large file.
/CheckIntegrity	Detects and tracks .wim file corruption when used with capture, unmount, export, and commit operations. /CheckIntegrity stops the operation if DISM detects that the .wim file is corrupted when used with apply and mount operations.
/ImageFile	Specifies the path of an image file, example: install.wim.

Example:

```
Dism /Split-Image /ImageFile:install.wim /SWMFile:split.swm /FileSize:650
```

## /Unmount-Image

Unmounts the .ffu, .wim, .vhdx or .vhd file and either commits or discards the changes that were made when the image was mounted.

You must use either the /commit or /discard argument when you use the /Unmount-Image option.

Syntax:

```
Dism /Unmount-Image /MountDir:<path_to_mount_directory> {/Commit | /Discard} [/CheckIntegrity] [/Append]
```

PARAMETER	DESCRIPTION
/CheckIntegrity	Detects and tracks .wim file corruption when used with capture, unmount, export, and commit operations. /CheckIntegrity stops the operation if DISM detects that the .wim file is corrupted when used with apply and mount operations.
/Append	Adds the modified image to the existing .wim file instead of overwriting the original image. The /CheckIntegrity and /Append arguments do not apply to virtual hard disk (VHD, VHDX), or FFU files.

Examples:

```
Dism /Unmount-Image /MountDir:C:\test\offline /commit
```

```
Dism /Unmount-Image /MountDir:C:\test\offline /discard
```

## /Update-WIMBootEntry

Updates the WIMBoot configuration entry, associated with the specified data source ID, with the renamed image file or moved image file path.

**Note:** /Update-WIMBootEntry requires a restart in order for any updates to take effect.

Syntax:

```
Dism /Update-WIMBootEntry /Path:<Volume_path> /DataSourceID:<Data_source_id> /ImageFile:<Renamed_image_path>
```

PARAMETER	DESCRIPTION
/Path	Specifies the disk volume of the WIMBoot configuration.
/DataSourceID	Specifies the data source ID as displayed by /Get-WIMBootEntry.

Example:

```
DISM.exe /Update-WIMBootEntry /Path:C:\ /DataSourceID:0 /ImageFile:R:\Install.wim
```

## /Apply-SiloedPackage

Applies one or more siloed provisioning packages (SPPs) to a specified image. This option is only available after running CopyDndl.cmd from the ADK for Windows 10, Version 1607, and running

```
dism.exe /Apply-SiloedPackage
```

 from the target folder created by CopyDndl.cmd.

**Note:** /Apply-SiloedPackage can only be run once against a Windows image, but /PackagePath can be used more than once in the same command to apply multiple SPPs. SPPs will be applied in the specified order, so a dependency should be specified before the SPP that depends on it.

For more information about siloed provisioning packages, and how to use CopyDndl.cmd, see [Siloed provisioning packages](#).

To find out how to work with siloed provisioning packages, see [Lab 10: Add desktop applications and settings with siloed provisioning packages \(SPPs\)](#).

```
/Apply-SiloedPackage /PackagePath:<package_path> /ImagePath:<applied_image_path>
```

PARAMETER	DESCRIPTION
/PackagePath	Specifies the path of a siloed provisioning package file.
/ImagePath	Specifies the path of the Windows image where you are applying the SPP.

Example:

```
Dism.exe /apply-SiloedPackage /PackagePath:C:\test\Word.spp /PackagePath:C:\test\spp2.spp /ImagePath:C:\
```

## Related topics

## [DISM - Deployment Image Servicing and Management Technical Reference for Windows](#)

[What is DISM?](#)

[DISM Global Options for Command-Line Syntax](#)

[Deploy Windows using Full Flash Update \(FFU\)](#)

[WIM vs. VHD vs. FFU: comparing image file formats](#)

# DISM Global Options for Command-Line Syntax

1/18/2019 • 5 minutes to read

Global options can be added to most of the servicing and imaging options in the Deployment Image Servicing and Management (DISM) tool. These options can be used to access the command-line help, specify the location of files to use, and control logging.

## Basic Syntax for Servicing Commands

After you have mounted or applied a Windows® image so that it is available offline as a flat file structure, you can specify any DISM global options, the servicing option that will update your image, and the location of the offline image. You can use only one servicing option per command line.

If you are servicing a running computer, you can use the **/Online** option instead of specifying the location of the offline Windows image. The commands and options that are available for servicing an image depend on which Windows operating system you are servicing. They also depend on whether the image is offline or a running operating system. All commands work on an offline Windows image. Subsets of the commands are available for servicing a running operating system.

The base syntax for DISM servicing commands is:

```
DISM.exe {/Image:<path_to_image> | /Online} {dism_global_options} {servicing_option}  
[<servicing_argument>]
```

For more information about servicing commands, see [Deployment Image Servicing and Management \(DISM\) Command-Line Options](#).

## Basic Syntax for Imaging Commands

Many of the global options are also available for imaging commands. The base syntax for DISM imaging commands is:

```
DISM.exe [dism_global_options] {servicing_option} [<servicing_argument>]
```

For more information about using DISM for image management, such as applying or mounting an image, see [DISM Image Management Command-Line Options](#).

## Global Options for Servicing and Imaging Commands

The following DISM global options are available for an offline image.

```
DISM.exe /image:<path_to_offline_image_directory> [/WinDir:<path_to_%WINDIR%>]  
[/LogPath:<path_to_log_file.log>] [/LogLevel:<n>] [/SysDriveDir:<path_to_bootMgr_file>] [/Quiet]  
[/NoRestart] [/ScratchDir:<path_to_scratch_directory>] [/English] [/Format:<output_format> ]
```

The following DISM global options are available for a running operating system.

```
DISM.exe /online [/LogPath:<path_to_log_file>] [/LogLevel:<n>] [/SysDriveDir:<path_to_bootMgr_file>]  
[/Quiet] [/NoRestart] [/ScratchDir:<path_to_scratch_directory>] [/English] [/Format:<output_format> ]
```

The following table provides a description of how each DISM global option can be used. These options are not case sensitive.

GLOBAL OPTION	DESCRIPTION
<p><b>/Get-Help</b></p> <p><b>/?</b></p>	<p>Displays information about available DISM command-line options and arguments.</p> <p>Use the <b>/?</b> or <b>/Get-Help</b> option without specifying an image file to get help on image management commands such as <b>/Mount-Image</b>.</p> <p>Example:</p> <p><b>Dism /?</b></p> <p>Specify an image file with the <b>/Image:&lt;path_to_an_image&gt;</b> option or use the <b>/Online</b> option to get help on the servicing command in the image, such as <b>/Get-Packages</b>. The options that are available for servicing an image depend on the servicing technology that is available in your image.</p> <p>Example:</p> <p><b>Dism /image:C:\test\offline /?</b></p> <p><b>Dism /online /?</b></p> <p>You can display additional Help by specifying a command-line option.</p> <p>Example:</p> <p><b>Dism /image:C:\test\offline /Add-Driver /?</b></p> <p><b>Dism /image:C:\test\offline /Add-Package /?</b></p> <p><b>Dism /online /Get-Drivers /?</b></p>
<p><b>/LogPath:&lt;path to log file.log&gt;</b></p>	<p>Specifies the full path and file name to log to. If not set, the default is: %WINDIR%\Logs\Dism\dism.log</p> <div data-bbox="817 1257 1428 1572" style="border: 1px solid black; padding: 10px;"> <p><b>Important</b></p> <p>In Windows PE, the default directory is the RAMDISK scratch space which can be as low as 32 MB.</p> <p>The log file will automatically be archived. The archived log file will be saved with .bak appended to the file name and a new log file will be generated. Each time the log file is archived the .bak file will be overwritten.</p> </div> <p>When using a network share that is not joined to a domain, use the <b>net use</b> command together with domain credentials to set access permissions before you set the log path for the DISM log.</p> <p>Example:</p> <p><b>Dism /image:C:\test\offline /LogPath:AddPackage.log /Add-Package /PackagePath:C:\packages\package.cab</b></p>

GLOBAL OPTION	DESCRIPTION
<b>/LogLevel:&lt;n&gt;</b>	<p>Specifies the maximum output level shown in the logs. The default log level is 3. The accepted values are as follows:</p> <ul style="list-style-type: none"> <li>1 = Errors only</li> <li>2 = Errors and warnings</li> <li>3 = Errors, warnings, and informational</li> <li>4 = All of the information listed previously, plus debug output</li> </ul> <p>Example:</p> <pre>Dism /image:C:\test\offline /LogPath:AddPackage.log /LogLevel:1 /Add- Package /PackagePath:C:\packages\package.cab</pre>
<b>/Image:&lt;path_to_offline_image_directory&gt;</b>	<p>This is the full path to the root directory of the offline Windows image that you will service. If the directory named Windows is not a subdirectory of the root directory, <b>/WinDir</b> must be specified.</p> <p>This option cannot be used with <b>/Online</b>.</p> <p>Example:</p> <pre>Dism /image:C:\test\offline /LogPath:AddPackage.log /LogLevel:1 /Add- Package /PackagePath:C:\packages\package.cab</pre>
<b>/WinDir:&lt;path_to_%WINDIR%&gt;</b>	<p>Used with the <b>/Image</b> option to specify the path to the Windows directory relative to the image path. This cannot be the full path to the Windows directory; it should be a relative path. If not specified, the default is the Windows directory in the root of the offline image directory.</p> <p>This option cannot be used with the <b>/Online</b> option.</p> <p>Example:</p> <pre>Dism /image:C:\test\offline /WinDir:WinNT /Add- Package /PackagePath:C:\packages\package.cab</pre>
<b>/Online</b>	<p>Specifies that the action is to be taken on the operating system that is currently running.</p> <p>This option cannot be used with the <b>/Image</b> or the <b>/WinDir</b> option. When <b>/Online</b> is used the Windows directory for the online image is automatically detected.</p> <p>Example:</p> <pre>Dism /online /Get-Packages</pre>

GLOBAL OPTION	DESCRIPTION
<b>/SysDriveDir:&lt;path_to_sysdrive_directory&gt;</b>	<p>Use <b>/SysDriveDir</b> to service an installed Windows image from a Windows PE environment.</p> <p>The <b>/SysDriveDir</b> option specifies the path to the location of the BootMgr files. This is necessary only when the BootMgr files are located on a partition other than the one that you are running the command from.</p> <p>For example, at a Windows PE command prompt, type:</p> <p style="padding-left: 20px;"><b>Dism /image:C:\Windows /SysDriveDir:C:&lt;/strong&gt;</b></p>
<b>/Quiet</b>	<p>Turns off information and progress output to the console. Only error messages will be displayed.</p> <p>To run in quiet mode, this option must be set every time that the command-line utility is run.</p> <div data-bbox="817 729 1436 887" style="border: 1px solid black; padding: 10px;"> <p><b>Note</b></p> <p>Do not use the <b>/Quiet</b> option with <b>/Get</b> commands. No information will be displayed.</p> </div> <p>Example:</p> <p style="padding-left: 20px;"><b>Dism /image:C:\test\offline /Add-Package /PackagePath:C:\packages\package.cab /quiet</b></p>
<b>/NoRestart</b>	<p>Suppresses reboot. If a reboot is not required, this command does nothing. This option will keep the application from prompting for a restart (or keep it from restarting automatically if the <b>/Quiet</b> option is used).</p> <p>Example:</p> <p style="padding-left: 20px;"><b>Dism /online /Add-Package /PackagePath:C:\packages\package.cab /NoRestart /quiet</b></p>
<b>/ScratchDir:&lt;path_to_scratchdirectory&gt;</b>	<p>Specifies a temporary directory that will be used when extracting files for temporary use during servicing. The directory must exist locally. If not specified, the \Windows&lt;em&gt;%Temp%&lt;/em&gt; directory will be used, with a subdirectory name of randomly generated hexadecimal value for each run of DISM. Items in the scratch directory are deleted after each operation.</p> <p>You should not use a network share location as a scratch directory to expand a package (.cab or .msu file) for installation. The directory used for extracting files for temporary usage during servicing should be a local directory.</p> <p>Example:</p> <p style="padding-left: 20px;"><b>Dism /image:C:\test\offline /ScratchDir:C:\Scratch /Add-Package /PackagePath:C:\packages\package.cab</b></p>

GLOBAL OPTION	DESCRIPTION
<b>/English</b>	<p>Displays command-line output in English.</p> <div data-bbox="822 233 1434 444" style="border: 1px solid black; padding: 10px;"> <p><b>Note</b></p> <p>Some resources cannot be displayed in English.</p> <p>This option is not supported when you use the <b>DISM /?</b> command.</p> </div> <p>Example:</p> <pre>Dism /Get-ImageInfo /ImagePath:C:\test\offline\install.wim /index:1 /English</pre>
<b>/Format:{Table   List}</b>	<p>Specifies the report output format.</p> <p>Example:</p> <pre>Dism /Image:C:\test\offline /Get-Apps /Format:table</pre>

## Related topics

[Deployment Image Servicing and Management \(DISM\) Command-Line Options](#)

[DISM Application Servicing Command-Line Options](#)

[DISM Windows Edition-Servicing Command-Line Options](#)

[DISM Languages and International Servicing Command-Line Options](#)

[DISM Operating System Package Servicing Command-Line Options](#)

[DISM Driver Servicing Command-Line Options](#)

[DISM Unattended Servicing Command-Line Options](#)

[DISM Windows PE Servicing Command-Line Options](#)

# DISM Operating System Package (.cab or .msu) Servicing Command-Line Options

1/18/2019 • 10 minutes to read

Use DISM with Windows cabinet (.cab) or Windows Update Stand-alone Installer (.msu) files to install or remove updates, service packs, language packs, and to enable or disable Windows features. Features are optional components for the core operating system.

## Syntax

```
DISM.exe {/Image:<path_to_image_directory> | /Online} [dism_global_options] {servicing_option}  
[<servicing_argument>]
```

The following operating system package-servicing options are available for an offline image:

```
DISM.exe /Image:<path_to_image_directory> [/Get-Packages | /Get-PackageInfo | /Add-Package | /Remove-  
Package ] [/Get-Features | /Get-FeatureInfo | /Enable-Feature | /Disable-Feature ] [/Cleanup-Image]
```

The following operating system package-servicing options are available for a running operating system:

```
DISM.exe /Online [/Get-Packages | /Get-PackageInfo | /Add-Package | /Remove-Package ] [/Get-Features |  
/Get-FeatureInfo | /Enable-Feature | /Disable-Feature ] [/Cleanup-Image]
```

## Operating system package-servicing options

This section describes how you can use each operating system package-servicing option. These options are not case sensitive.

### /Get-Help /?

When used immediately after a package-servicing command-line option, information about the option and the arguments is displayed.

Additional topics might become available when an image is specified.

Syntax:

```
Dism /Get-Help
```

Examples:

```
Dism /Image:C:\test\offline /Add-Package /?
```

```
Dism /Online /Get-Packages /?
```

### /Get-Packages

Displays basic information about all packages in the image. Use the /Format:Table or /Format>List argument to

display the output as a table or a list.

Syntax:

```
Dism /Get-Packages [/Format:{Table | List}]
```

Examples:

```
Dism /Image:C:\test\offline /Get-Packages
```

```
Dism /Image:C:\test\offline /Get-Packages /Format:Table
```

```
Dism /Online /Get-Packages
```

### /Get-PackageInfo

Displays detailed information about a package provided as a .cab file. Only .cab files can be specified. You cannot use this command to obtain package information for .msu files. /PackagePath can point to either a .cab file or a folder.

You can use the /Get-Packages option to find the name of the package in the image, or you can specify the path to the .cab file. The path to the .cab file should point to the original source of the package, not to where the file is installed on the offline image.

Syntax:

```
Dism /Get-PackageInfo {/PackageName:<name_in_image> | /PackagePath:<path_to_cabfile>}
```

Examples:

```
Dism /Image:C:\test\offline /Get-PackageInfo /PackagePath:C:\packages\package.cab
```

```
Dism /Image:C:\test\offline /Get-PackageInfo  
/PackageName:Microsoft.Windows.Calc.Demo~6595b6144ccf1df~x86~en~1.0.0.0
```

### /Add-Package

Installs a specified .cab or .msu package in the image. An .msu package is supported only when the target image is offline, either mounted or applied.

Multiple packages can be added on one command line. The applicability of each package will be checked. If the package cannot be applied to the specified image, you will receive an error message. Use the /IgnoreCheck argument if you want the command to process without checking the applicability of each package.

Use the /PreventPending option to skip the installation of the package if the package or Windows image has pending online actions. (Introduced in Windows 8/Windows PE 4.0).

/PackagePath can point to:

- A single .cab or .msu file.
- A folder that contains a single expanded .cab file.
- A folder that contains a single .msu file.

- A folder that contains multiple .cab or .msu files.

## Notes

- If /PackagePath points to a folder that contains a .cab or .msu files at its root, any subfolders will also be recursively checked for .cab and .msu files.
- /Add-Package doesn't run a full check for a package's applicability and dependencies. If you're adding a package with dependencies, make sure that all dependencies are installed when you add the package.

Syntax:

```
Dism /Add-Package /PackagePath:<path_to_cabfile> [/IgnoreCheck] [/PreventPending]
```

Examples:

```
Dism /Image:C:\test\offline /LogPath:AddPackage.log /Add-Package /PackagePath:C:\packages\package.msu
```

```
Dism /Image:C:\test\offline /Add-Package /PackagePath:C:\packages\package1.cab  
/PackagePath:C:\packages\package2.cab /IgnoreCheck
```

```
Dism /Image:C:\test\offline /Add-Package /PackagePath:C:\test\packages\package.cab /PreventPending
```

## /Remove-Package

Removes a specified .cab file package from the image. Only .cab files can be specified. You cannot use this command to remove .msu files.

**Note** Using this command to remove a package from an offline image will not reduce the image size.

You can use the /PackagePath option to point to the original source of the package, specify the path to the CAB file, or you can specify the package by name as it is listed in the image. Use the /Get-Packages option to find the name of the package in the image.

Syntax:

```
/Remove-Package {/PackageName:<name_in_image> | /PackagePath:<path_to_cabfile>}
```

Examples:

```
Dism /Image:C:\test\offline /LogPath:C:\test\RemovePackage.log /Remove-Package  
/PackageName:Microsoft.Windows.Calc.Demo~6595b6144ccf1df~x86~en~1.0.0.0
```

```
Dism /Image:C:\test\offline /LogPath:C:\test\RemovePackage.log /Remove-Package  
/PackageName:Microsoft.Windows.Calc.Demo~6595b6144ccf1df~x86~en~1.0.0.0 /PackageName:Microsoft-Windows-  
MediaPlayer-Package~31bf3856ad364e35~x86~~6.1.6801.0
```

```
Dism /Image:C:\test\offline /LogPath:C:\test\RemovePackage.log /Remove-Package  
/PackagePath:C:\packages\package1.cab /PackagePath:C:\packages\package2.cab
```

## /Get-Features

Displays basic information about all features (operating system components that include optional Windows foundation features) in a package. You can use the /Get-Features option to find the name of the package in the image, or you can specify the path to the original source of the package. If you do not specify a package name or path, all features in the image will be listed. /PackagePath can point to either a .cab file or a folder.

Feature names are case sensitive if you are servicing a Windows image other than Windows 8.

Use the /Format:Table or /Format>List argument to display the output as a table or a list.

Syntax:

```
/Get-Features {/PackageName:<name_in_image> | /PackagePath:<path_to_cabfile>} [/Format:{Table | List}]
```

Examples:

```
Dism /Image:C:\test\offline /Get-Features
```

```
Dism /Image:C:\test\offline /Get-Features /Format>List
```

```
Dism /Image:C:\test\offline /Get-Features  
/PackageName:Microsoft.Windows.Calc.Demo~6595b6144ccf1df~x86~en~1.0.0.0
```

```
Dism /Image:C:\test\offline /Get-Features /PackagePath:C:\packages\package1.cab
```

### **/Get-FeatureInfo**

Displays detailed information about a feature. You must use /FeatureName. Feature names are case sensitive if you are servicing a Windows image other than Windows 10 or Windows 8.x. You can use the /Get-Features option to find the name of the feature in the image.

/PackageName and /PackagePath are optional and can be used to find a specific feature in a package.

Syntax:

```
/Get-FeatureInfo /FeatureName:<name_in_image> [{/PackageName:<name_in_image> | /PackagePath:  
<path_to_cabfile>}]
```

Examples:

```
Dism /Image:C:\test\offline /Get-FeatureInfo /FeatureName:Hearts
```

```
Dism /Image:C:\test\offline /Get-FeatureInfo /FeatureName:Hearts /PackagePath:C:\packages\package.cab
```

### **/Enable-Feature**

Enables or updates the specified feature in the image. You must use the /FeatureName option. Feature names are case sensitive if you are servicing a Windows image other than Windows 8. Use the /Get-Features option to find the name of the feature in the image.

You can specify the /FeatureName option multiple times in one command line for features that share the same parent package.

You do not have to specify the package name using the /PackageName option if the package is a Windows Foundation Package. Otherwise, use /PackageName to specify the parent package of the feature.

You can restore and enable a feature that has previously been removed from the image. Use the /Source argument to specify the location of the files that are required to restore the feature. The source of the files can be the Windows folder in a mounted image, for example c:\test\mount\Windows. You can also use a Windows side-by-side folder as the source of the files, for example z:\sources\SxS.

If you specify multiple /Source arguments, the files are gathered from the first location where they are found and the rest of the locations are ignored. If you do not specify a /Source for a feature that has been removed, the default location in the registry is used or, for online images, Windows Update (WU) is used.

Use /LimitAccess to prevent DISM from contacting WU for online images.

Use /All to enable all parent features of the specified feature.

The /Source, /LimitAccess, and /All arguments can be used with Windows 10, Windows 8.x, and Windows PE images above 4.0.

Syntax:

```
/Enable-Feature /FeatureName:<name_in_image> [/PackageName:<name_in_image>] [/Source: <source>]  
[/LimitAccess] [/All]
```

Examples:

```
Dism /Online /Enable-Feature /FeatureName:Hearts /All
```

```
Dism /Online /Enable-Feature /FeatureName:Calc /Source:c:\test\mount\Windows /LimitAccess
```

```
Dism /Image:C:\test\offline /Enable-Feature /FeatureName:Calc  
/PackageName:Microsoft.Windows.Calc.Demo~6595b6144ccf1df~x86~en~1.0.0.0
```

## **/Disable-Feature**

Disables the specified feature in the image. You must use the /FeatureName option. Feature names are case sensitive if you are servicing a Windows image other than Windows 8. Use the /Get-Features option to find the name of the feature in the image.

You can specify /FeatureName multiple times in one command line for features in the same parent package.

You do not have to specify the package name using the /PackageName option if it the package is a Windows Foundation Package. Otherwise, use /PackageName to specify the parent package of the feature.

Use /Remove to remove a feature without removing the feature's manifest from the image. This option can only be used with Windows 10, Windows 8.x, and Windows PE images above 4.0. The feature will be listed as "Removed" when you use /Get-FeatureInfo to display feature details and can be restored and enabled using /Enable-Feature with the /Source option.

Syntax:

```
/Disable-Feature /FeatureName:<name_in_image> [/PackageName:<name_in_image>] [/Remove]
```

Examples:

```
*Dism /Online /Disable-Feature /FeatureName:Hearts
```

```
Dism /Image:C:\test\offline /Disable-Feature /FeatureName:Calc  
/PackageName:Microsoft.Windows.Calc.Demo~6595b6144ccf1df~x86~en~1.0.0.0
```

## /Cleanup-Image

Performs cleanup or recovery operations on the image. /AnalyzeComponentStore and /ResetBase can be used with Windows 10, Windows 8.1, and Windows PE images above 5.0. Beginning with Windows 10, version 1607, you can specify /Defer with /ResetBase. But we highly recommend you **only** use /Defer as an option in the factory where DISM /Resetbase requires more than 30 minutes to complete. /StartComponentCleanup can be used with Windows 10, Windows 8.x, and Windows PE images above 4.0. /CheckHealth, /ScanHealth, /RestoreHealth, /Source, and /LimitAccess can be used with Windows 10, Windows 8.x, and Windows PE images above 4.0. /HideSP and /SPSuperseded can't be used when servicing a version of Windows that is earlier than Windows 7 Service Pack 1 (SP1) image.

**Tip** To determine when the /ResetBase option was last run, check the LastResetBase\_UTC registry entry under this registry path:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing

Syntax:

```
/Cleanup-Image {/RevertPendingActions | /SPSuperseded [/HideSP] | /StartComponentCleanup [/ResetBase  
[/Defer]] | /AnalyzeComponentStore | /CheckHealth | /ScanHealth | /RestoreHealth [/Source: <filepath>]  
[/LimitAccess]}
```

PARAMETER	DESCRIPTION
/RevertPendingActions	If you experience a boot failure, you can use the /RevertPendingActions option to try to recover the system. The operation reverts all pending actions from the previous servicing operations because these actions might be the cause of the boot failure. The /RevertPendingActions option is not supported on a running operating system or a Windows PE or Windows Recovery Environment (Windows RE) image. Important: You should use the /RevertPendingActions option only in a system-recovery scenario on a Windows image that did not boot.
/SPSuperseded	Removes any backup files created during the installation of a service pack. Use /HideSP to prevent the service pack from being listed in the Installed Updates Control Panel. The service pack cannot be uninstalled after the /SPSuperseded operation is completed.
/StartComponentCleanup	Cleans up the superseded components and reduces the size of the component store. Use /ResetBase to reset the base of superseded components, which can further reduce the component store size. Installed Windows updates can't be uninstalled after running /StartComponentCleanup with the /ResetBase option. Use /Defer with /ResetBase to defer long-running cleanup operations to the next automatic maintenance.

PARAMETER	DESCRIPTION
/AnalyzeComponentStore	Creates a report of the component store. For more information about the report and how to use the information provided in the report, see <a href="#">Determine the Actual Size of the WinSxS Folder</a> .
/CheckHealth	Checks whether the image has been flagged as corrupted by a failed process and whether the corruption can be repaired.
/ScanHealth	Scans the image for component store corruption. This operation will take several minutes.
/RestoreHealth	Scans the image for component store corruption, and then performs repair operations automatically. This operation will take several minutes.
/Source	Used with /RestoreHealth to specify the location of known good versions of files that can be used for the repair, such as a path to the Windows directory of a mounted image.
/LimitAccess	Prevents DISM from contacting Windows Update for repair of online images.

Examples:

```
Dism /Image:C:\test\offline /Cleanup-Image /RevertPendingActions
```

```
Dism /Image:C:\test\offline /Cleanup-Image /SPSuperseded /HideSP
```

```
Dism /Online /Cleanup-Image /ScanHealth
```

```
Dism /Online /Cleanup-Image /RestoreHealth /Source:c:\test\mount\windows /LimitAccess
```

To learn more, see [Repair a Windows Image](#).

## Limitations

- When you are installing a package in an offline image, the package state is “install pending” because of pending online actions. In other words, the package will be installed when the image is booted and the online actions are processed. If subsequent actions are requested, they cannot be processed until the previous pending online action is completed. You can use the /PreventPending option when you add a package with /AddPackage to skip the installation of a package when there are pending online actions.
- Some packages require other packages to be installed first. You should not assume that dependencies will be satisfied. If there are dependency requirements, you should use an answer file to install the necessary packages. By passing an answer file to DISM, multiple packages can be installed in the correct order. This is the preferred method for installing multiple packages. For more information, see [Add or Remove Packages Offline Using DISM](#).
- Packages are installed in the order that they are listed in the command line.
- When using DISM to list the optional components in a Windows PE image, the optional components will always be listed as pending even when the servicing operation was successful. This is by design and requires

no additional action from you.

## Related topics

[What is DISM?](#)

[DISM Image Management Command-Line Options](#)

[Deployment Image Servicing and Management \(DISM\) Command-Line Options](#)

# DISM Provisioning Package (.ppkg) Command-Line Options

1/24/2019 • 2 minutes to read

Use DISM to work with Provisioning Packages (.ppkg) files. For example, you can add settings and Windows desktop applications to Windows 10, or reduce the size of your Windows installation.

## /Add-ProvisioningPackage

Adds applicable payload of provisioning package to the specified image.

Syntax:

```
DISM.exe /Add-ProvisioningPackage /PackagePath:<package_path> [/CatalogPath:<path>]
```

Example:

```
DISM.exe /Image=C:\ /Add-ProvisioningPackage /PackagePath:C:\oem.ppkg
```

## /Get-ProvisioningPackageInfo

Get the information of provisioning package.

Syntax:

```
DISM.exe /Get-ProvisioningPackageInfo /PackagePath:<package_path>
```

Example:

```
DISM.exe /Image=C:\ /Get-ProvisioningPackageInfo /PackagePath:C:\oem.ppkg
```

## /Apply-CustomDataImage

Dehydrates files contained in the custom data image to save space. For client editions, this package is used by the push-button recovery tools.

Syntax:

```
/Apply-CustomDataImage /CustomDataImage:<path_to_image_file> /ImagePath:<target_drive> /SingleInstance
```

PARAMETER	DESCRIPTION
/CustomDataImage	Specifies where the provisioning package is stored.

PARAMETER	DESCRIPTION
/ImagePath	Specifies the drive that contains the Windows image. DISM scans this drive for any non-system files on this drive and incorporates them into the provisioning package.
/SingleInstance	After DISM captures the non-system files to a compressed provisioning package, DISM adds pointers on the drive to the new compressed provisioning package, and removes the original files. As a result, the files are still visible to the system, but take up less space on the drive.

Example:

```
DISM.exe /Apply-CustomDataImage /CustomDataImage:C:\oem.pkg /ImagePath:C:\ /SingleInstance
```

Applies to: Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) only.

# DISM App Package (.appx or .appxbundle) Servicing Command-Line Options

1/18/2019 • 9 minutes to read

You can use app package-servicing commands to add, remove, and list provisioned app packages (.appx or .appxbundle) in a Windows image. An .appxbundle, new for Windows 10, is a collection of app and resource packages used together to enrich the app experience, while minimizing the disk footprint on a given PC. For detailed documentation about .appxbundle packages and the Microsoft Store pipeline, see [App packaging](#). Only a subset of the packages within an .appxbundle might be added to the image when a bundle is provisioned using DISM. For more information, see [Understanding How DISM Adds .appxbundle Resource Packages to an Image](#).

Provisioned app packages are added to a Windows image and are then installed for every new or existing user profile the next time the user logs on. For more information, including requirements for app package provisioning, see [Sideload Apps with DISM](#).

You can also use PowerShell to add, remove, and list app packages (.appx or .appxbundle) per image or per user in a Windows installation. For more information, see [Deployment Imaging Servicing Management \(DISM\) Cmdlets in Windows PowerShell](#) and [App Installation Cmdlets in Windows PowerShell](#).

The base syntax for servicing a Windows image using DISM is:

```
DISM.exe {/Image:<path_to_image_directory> | /Online} [dism_global_options] {servicing_option} [<servicing_argument>]
```

The following app package (.appx or .appxbundle) servicing options are available for an offline image.

```
DISM.exe /Image:<path_to_image_directory> [/Get-ProvisionedAppxPackages | /Add-ProvisionedAppxPackage | /Remove-ProvisionedAppxPackage | /Set-ProvisionedAppxDataFile]
```

The following app package (.appx or .appxbundle) servicing options are available for a running operating system.

```
DISM.exe /Online [/Get-ProvisionedAppxPackages | /Add-ProvisionedAppxPackage | /Remove-ProvisionedAppxPackage | /Set-ProvisionedAppxDataFile]
```

## App package servicing options

This table describes how you can use each app servicing option. These options are not case sensitive.

SERVICING OPTION	DESCRIPTION	EXAMPLE
/Get-Help /?	When used immediately after an app package servicing command-line option, information about the option and the arguments is displayed. Additional topics might become available when an image is specified.	<pre>Dism /image:C:\test\offline /Add-ProvisionedAppxPackage /? Dism /online /Get-ProvisionedAppxPackages /?</pre>
/Get-ProvisionedAppxPackages	Displays information about app packages (.appx or .appxbundle), in an image, that are set to install for each new user.	<pre>Dism /Image:C:\test\offline /Get-ProvisionedAppxPackages</pre>
/Add-ProvisionedAppxPackage	<p>Adds one or more app packages to the image. The app will be added to the Windows image and registered for each existing or new user profile the next time the user logs in. If the app is added to an online image, the app will not be registered for the current user until the next time the user logs in.</p> <p>It is recommended to provision apps in an online operating system in audit mode so that appropriate hard links can be created for apps that contain the exact same files (to minimize disk space usage) while also ensuring no apps are running for a successful installation.</p> <p><b>Syntax:</b></p> <pre>Dism /Image:C:\test\offline /Add-ProvisionedAppxPackage /FolderPath:C:\Test\Apps\MyUnpackedApp /CustomDataFolderPath:c:\Test\Apps\CustomData.xml Dism /Online /Add-ProvisionedAppxPackage /PackagePath:C:\Test\Apps\MyPackedApp\MainPackage.appx /DependencyPackagePath:C:\Test\Apps\MyPackedApp\FranchiseX64.appx /DependencyPackagePath:C:\Test\Apps\MyPackedApp\FranchiseX64.appx /LicensePath:C:\Test\Apps\MyLicense.xml Dism /Online /Add-ProvisionedAppxPackage /FolderPath:C:\Test\Apps\MyUnpackedApp /SkipLicense Dism /Image:C:\test\offline /Add-ProvisionedAppxPackage /PackagePath:C:\Test\Apps\MyPackedApp\MainPackage.appx /SkipLicense Dism /Online /Add-ProvisionedAppxPackage /PackagePath:C:\Test\Apps\MyPackedApp\MainPackage.appx /Region:"all" Dism /Online /Add-ProvisionedAppxPackage /PackagePath:C:\Test\Apps\MyPackedApp\MainPackage.appx /Region:"US;GB"</pre>	<pre>Dism /Image:C:\test\offline /Add-ProvisionedAppxPackage /CustomDataFolderPath:c:\Test\Apps\CustomData.xml Dism /Online /Add-ProvisionedAppxPackage /DependencyPackagePath:C:\Test\Apps\MyPackedApp\FranchiseX64.appx /LicensePath:C:\Test\Apps\MyLicense.xml Dism /Online /Add-ProvisionedAppxPackage /FolderPath:C:\Test\Apps\MyUnpackedApp /SkipLicense Dism /Image:C:\test\offline /Add-ProvisionedAppxPackage /SkipLicense Dism /Online /Add-ProvisionedAppxPackage /Region:"all" Dism /Online /Add-ProvisionedAppxPackage /Region:"US;GB"</pre>

SERVICING OPTION	DESCRIPTION	EXAMPLE
	<pre>dism.exe /Add-ProvisionedAppxPackage [/FolderPath:&lt;App_folder_path&gt; [/<b>SkipLicense</b>] [/CustomDataPath: &lt;custom_file_path&gt;] /PackagePath: &lt;main_package_path&gt; [/<b>DependencyPackagePath:</b> &lt;dependency_package_path&gt;] {[/<b>licenseFile</b>:&lt;license_file_path&gt;] [/<b>SkipLicense</b>\]} [/CustomDataPath: &lt;custom_file_path&gt;]] [/Region:&lt;region&gt;]</pre> <p>Use <b>/FolderPath</b> to specify a folder of unpacked app files containing a main package, any dependency packages, and the license file. This is only supported for an unpacked app package.</p> <p>Use <b>/PackagePath</b> to specify an app package (.appx or .appxbundle). You can use <b>/PackagePath</b> when provisioning a line-of-business app online.</p> <p><b>Important:</b> Use the <b>/PackagePath</b> parameter to provision .appxbundle packages. Also, dependency packages cannot be provisioned with <b>/PackagePath</b>, they must be provisioned with the <b>/DependencyPackagePath</b> parameter for an app.</p> <p><b>/PackagePath</b> is not supported from a host PC that is running Windows Preinstallation Environment (WinPE) 4.0, Windows Server 2008 R2, or an earlier version of Windows.</p> <p>Use <b>/Region</b> to specify what regions an app package (.appx or .appxbundle) must be provisioned in. The region argument can either be "all", indicating that the app should be provisioned for all regions, or it can be a semi-colon delimited list of regions. The regions will be in the form of <a href="#">ISO 3166-1 Alpha-2</a> or <a href="#">ISO 3166-1 Alpha-3 codes</a>. For example, the United States can be specified as either "US" or "USA" (case-insensitive). When a list of regions is not specified, the package will be provisioned only if it is pinned to start layout.</p> <p>Use <b>/DependencyPackagePath</b> to specify each dependency package needed for the app to be provisioned. The necessary dependency packages of an app can be found by looking at the elements in the AppxManifest.xml in the root of the .appx package of the app. If multiple apps all share the same dependency, the latest minor version of each major version of the dependency package should be installed. For example, App1, App2, and App3 all have a dependency on Microsoft.NET.Native.Framework. App1 specifies Microsoft.NET.Native.Framework.1.6 with minor version 25512.0, App2 specifies Microsoft.NET.Native.Framework.1.6 with minor version 25513.0, and App3 specifies Microsoft.NET.Native.Framework.1.3 with minor version 24202.0. Because both App1 and App2 both specify the same major version of the dependency package, only the latest minor version 25513.0 should be installed, while App3 specifies a different major version of the dependency package, so it must also be installed. So the dependency packages that should be installed are Microsoft.NET.Native.Framework.1.6 with minor version 25513.0 and Microsoft.NET.Native.Framework.1.3 with minor version 24202.0.</p> <p>If the package has dependencies that are architecture-specific, you must install all of the applicable architectures for the dependency on the target image. For example, on an x64 target image, include a path to both the x86 and x64 dependency packages or include them both in</p>	

SERVICING OPTION	<p>the folder of unpacked app files. If the ARM <b>DESCRIPTION</b> package is also specified or included, DISM will ignore it since it does not apply to the target x64 image.</p> <table border="1" data-bbox="599 197 1001 467"> <thead> <tr> <th data-bbox="599 197 806 265">COMPUTER ARCHITECTURE</th><th data-bbox="806 197 1001 265">DEPENDENCIES TO INSTALL:</th></tr> </thead> <tbody> <tr> <td data-bbox="599 265 806 332">x64</td><td data-bbox="806 265 1001 332">x64 and x86</td></tr> <tr> <td data-bbox="599 332 806 399">x86</td><td data-bbox="806 332 1001 399">x86</td></tr> <tr> <td data-bbox="599 399 806 467">ARM</td><td data-bbox="806 399 1001 467">Windows RT (ARM) only</td></tr> </tbody> </table>	COMPUTER ARCHITECTURE	DEPENDENCIES TO INSTALL:	x64	x64 and x86	x86	x86	ARM	Windows RT (ARM) only	EXAMPLE
COMPUTER ARCHITECTURE	DEPENDENCIES TO INSTALL:									
x64	x64 and x86									
x86	x86									
ARM	Windows RT (ARM) only									
	<p>Use <b>/CustomDataPath</b> to specify an optional custom data file for an app. You can specify any file name. The file will be renamed to Custom.dat when it is added to the image.</p> <p>Use <b>/LicensePath</b> with the <b>/PackagePath</b> option to specify the location of the .xml file containing your application license.</p> <p>Only use <b>/SkipLicense</b> with apps that do not require a license on a sideloading-enabled computer. Using <b>/SkipLicense</b> in other scenarios can compromise an image.</p>									
<b>/Remove-ProvisionedAppxPackage</b>	<p>Removes provisioning for app packages (.appx or .appxbundle) from the image. App packages will not be registered to new user accounts that are created.</p> <p><b>Syntax:</b></p> <div style="border: 1px solid black; padding: 2px;"> <pre>/Remove-ProvisionedAppxPackage /PackageName:&lt;PackageName&gt;</pre> </div> <p><b>Important:</b> This option will only remove the provisioning for a package if it is registered to any user profile. Use the <a href="#">Remove-AppxPackage</a> cmdlet in PowerShell to remove the app for each user that it is already registered to in order to fully remove the app from the image.</p> <p>If the app has not been registered to any user profile, the <b>/Remove-ProvisionedAppxPackage</b> option will remove the package completely.</p> <p>To remove app packages from a Windows Server 2012 image that has the Desktop Experience installed, you must remove the app packages before you remove the Desktop Experience. The Desktop Experience is a requirement of the <b>/Remove-ProvisionedAppxPackage</b> option for Server Core installations of Windows Server 2012.</p>	<pre>Dism /Image:C:\test\offline /Remove-ProvisionedAppxPackage /PackageName:microsoft.devx.appx.app1_1.0.0.0_neutri</pre>								
<b>/Optimize-ProvisionedAppxPackages</b>	<p>Optimizes the total file size of provisioned packages on the image by replacing identical files with hardlinks. Once an image with provisioned AppX packages is online, <b>/optimize-provisionedappxpackages</b> won't be able to optimize provisioned AppX packages. If you take an image offline and add packages, only the packages that are provisioned after the image was brought back offline will be optimized.</p> <p>This command is not supported against an online image.</p>	<pre>DISM.exe /Image:C:\test\offline /Optimize-ProvisionedAppxPackages</pre>								

SERVICING OPTION	DESCRIPTION	EXAMPLE
/Set-ProvisionedAppxDataFile	<p>Adds a custom data file into the specified app package (.appx or .appxbundle).</p> <p><b>Syntax:</b></p> <pre>/Set-ProvisionedAppxDataFile [ /CustomDataPath:&lt;custom_file_path&gt;   /PackageName:&lt;PackageName&gt;</pre> <p>The specified app (.appx or .appxbundle) package must already be added to the image prior to when you add the custom data file with this option. You can also add a custom data file when you use the <b>/Add-ProvisionedAppxPackage</b> option.</p> <p>Use <b>/CustomDataPath</b> to specify an optional custom data file for an app. You can specify any file name. The file will be renamed to Custom.dat when it is added to the image. If a Custom.dat file already exists, it will be overwritten.</p> <p>Use <b>/PackageName</b> to specify an app package (.appx or .appxbundle).</p>	<pre>DISM.exe /Image:C:\test\offline /Set-ProvisionedAppxDataFile /CustomDataPath:c:\Test\Apps\Custom.dat /PackageName:microsoft.appx.app1_1.0.0.0_neutral_ae4138d7.appx</pre>

## Understanding How DISM Adds .appxbundle Resource Packages to an Image

When an .appxbundle is added to the image, not all resource packages within the bundle are applicable. For example, if an app is being added to a Windows image with a Spanish (Spain) default language, French (France) resources should not be included. To determine what resources are added to the image, the package applicability is determined using:

- **Language Resource Packs:** If an operating system language is not present, the corresponding app language resource pack is not added. For example, you might have an image that is a Windows 10 with English (US) as the default language, and a Spanish (Spain) language pack included. English (US) and Spanish (Spain) app resource packs will be added to the image. If a French (France) resource pack (or any other language) is available in the app bundle, it will not be added.
- **Scale and DirectX (DXFL) Resource Packs:** Scale and DirectX (DXFL) resource packs depend upon the hardware configuration of the Windows device. Because the type of target hardware can't be known at the time the DISM commands are run, all scale and DXFL resource packages are added to the image at provisioning time. For more information about developing an app with scaling resources, see [Guidelines for scaling to pixel density \(Microsoft Store apps\)](#).

For an image containing multiple language packs, app resource packages will be added to the image for each language. Once the first user has signed in to the PC with the deployed image and the user has chosen a language during OOBE, the inapplicable resource packages, (language resource packs, scale resource packs and DXFL resource packages) that do not match the user profile settings are removed.

For example, an app might support English (US), French (France), and Spanish (Spain) languages. If the app is added to an image with English (US) and Spanish (Spain) language packs present, only English (US) and Spanish (Spain) resource packs will be added to the image. Then, if a user signs in for the first time and, during OOBE, selects English (US) as their operating system language, the Spanish (Spain) resource packages will be removed after sign in completes.

**Important** If you add or remove a language pack from an image, you change the applicability context which may result in leaving an incorrect or incomplete set of resource packages in the image. When a language pack is added or removed, you must, once again, add all .appxbundle packages (including any dependency packages and Microsoft Store license file) to the image. This will ensure that the correct set of resource packages is provisioned.

## Limitations

- You cannot install an app package (.appx) on an operating system that does not support Windows 8 apps. You can't install an app bundle package (.appxbundle) on an operating system that does not support at least Windows 8.1 apps. Apps aren't supported on WinPE 4.0, the Windows Server 2012 Server Core installation option, or on any versions of Windows older than Windows 8 and Windows Server 2012.

To install and run apps on Windows Server 2012, you must install the [Desktop Experience](#).

- The **/FolderPath** option is only supported for app packages based on the .appx format.
- **/PackagePath** must always be used for .appxbundle packages.

## Related topics

[What is DISM?](#)

[DISM Image Management Command-Line Options](#)

[Deployment Image Servicing and Management \(DISM\) Command-Line Options](#)

[Sideloading Apps with DISM](#)

# DISM Application Servicing (.msp) Command-Line Options

1/18/2019 • 2 minutes to read

Application servicing command-line options can be used on an offline image to check the applicability of Windows Installer application patches (.msp files) and to query your offline image for information about installed Windows Installer applications and application patches (.msp files).

For information about using Deployment Image Servicing and Management (DISM) with app packages, see [DISM App Package \(.appx or .appxbundle\) Servicing Command-Line Options](#).

The base syntax for servicing a Windows image using DISM is:

```
DISM.exe /Image:<path_to_image_directory> [dism_global_options] {servicing_option}
[<servicing_argument>]
```

The following servicing options are available to list Windows Installer applications and .msp application patches, and to check the applicability of an application patch for an offline Windows image:

```
DISM.exe /Image:<path_to_directory> [/Check-AppPatch | /Get-AppPatchInfo: | /Get-AppPatches | /Get-AppInfo | /Get-Apps]
```

## Application servicing options

This section describes how you can use each application servicing option. These options are not case sensitive.

**/Get-Help /?**

When used immediately after a package servicing command-line option, information about the option and the arguments is displayed. Additional topics might become available when an image is specified.

Example:

```
Dism /image:C:\test\offline /Check-AppPatch /?
```

```
/Check-AppPatch /PatchLocation:< path_to_patch.msp>
```

Displays information only if the MSP patches apply to the offline image. The path to the MSP patch file must be specified. Multiple patch files can be specified.

Example:

```
Dism /image:C:\test\offline /Check-AppPatch /PatchLocation:C:\test\MSIPatches\MsiTestPatch1.msp
/PatchLocation:C:\test\MSIPatches\MsiTestPatch2.msp
```

```
/Get-AppPatchInfo: [/PatchCode:< patch_code_GUID>] [/ProductCode:< product_code_GUID>]
```

Displays detailed information about installed MSP patches filtered by <patch\_code\_GUID> and <product\_code\_GUID>.

If the **/PatchCode** option is specified, detailed information is displayed for all Windows Installer applications that the patch is applied to.

If the **/ProductCode** option is specified, information about all MSP patches in the specified application is displayed.

If the **/PatchCode** and **/ProductCode** options are specified, information is displayed only if that specific patch is

applied to the specified Windows Installer application.

Use the **/Get-AppPatches** option to find the patch code GUID and the product code GUID specific to the patch.

Use the **/Get-Apps** option to list all product code GUIDs for an installed Windows Installer applications.

If **/PatchCode** and **/ProductCode** are not specified, all installed Windows Installer packages and MSP patches are displayed.

Example:

```
Dism /image:C:\test\offline /Get-AppPatchInfo
```

```
Dism /image:C:\test\offline /Get-AppPatchInfo: /PatchCode:{B0B9997C-GUID-GUID-GUID-74D866BBDFFF}
```

```
Dism /image:C:\test\offline /Get-AppPatchInfo: /ProductCode:{B0F9497C-GUID-GUID-GUID-74D866BBDF59}
```

```
Dism /image:C:\test\offline /Get-AppPatchInfo: /PatchCode:{B0B9997C-GUID-GUID-GUID-74D866BBDFFF} /ProductCode:{B0F9497C-GUID-GUID-GUID-74D866BBDF59}
```

**/Get-AppPatches: [/ProductCode:< product\_code\_GUID >]**

Displays basic information about all applied MSP patches for all applications installed on the offline image. If a product code GUID is specified, information is displayed about all patches in the specified Windows Installer application.

Examples:

```
Dism /image:C:\test\offline /Get-AppPatches
```

```
Dism /image:C:\test\offline /Get-AppPatches /ProductCode:{B0F9497C-GUID-GUID-GUID-74D866BBDF59}
```

**/Get-AppInfo: [/ProductCode:< product\_code\_GUID >]**

Displays detailed information about a specific installed Windows Installer application.

Use the **/Get-Apps** option to find the GUID for an installed Windows Installer application. If a product code GUID is not specified, information is displayed for all Windows Installer applications installed in the offline image.

Examples:

```
Dism /image:C:\test\offline /Get-AppInfo
```

```
Dism /image:C:\test\offline /Get-AppInfo /ProductCode:{B0F9497C-GUID-GUID-GUID-74D866BBDF59}
```

**/Get-Apps**

Displays basic information about all Windows Installer applications in the offline image.

Example:

```
Dism /image:C:\test\offline /Get-Apps
```

## Limitations

**/Get-AppPatches** and **/Get-AppPatchInfo** apply only to installed patches (.msp files).

When you determine the applicability of an MSP patch, only the Windows Installer applications for which the patch is applicable will be displayed. One patch can apply to many installed applications and many patches can apply to one application.

## Related topics

[What is DISM?](#)

[DISM Image Management Command-Line Options](#)

[Deployment Image Servicing and Management \(DISM\) Command-Line Options](#)

[DISM App Package \(.appx or .appxbundle\) Servicing Command-Line Options](#)

# DISM Default Application Association Servicing Command-Line Options

1/18/2019 • 2 minutes to read

You can use the default application association-servicing commands to import, export, list, and remove the settings that specify which application opens a file based on the file name extension or protocol.

The base syntax for servicing a Windows image using DISM is:

**DISM.exe {/Image:<path\_to\_image\_directory> | /Online} [dism\_global\_options] {servicing\_option} [<servicing\_argument>]**

The following default application servicing options are available for an offline image.

**DISM.exe /image:<path\_to\_image\_directory> [/Get-DefaultAppAssociations | /Import-DefaultAppAssociations | /Remove-DefaultAppAssociations]**

The following default application association servicing options are available for a running operating system.

**DISM.exe /Online [/Export-DefaultAppAssociations | /Get-DefaultAppAssociations | Import-DefaultAppAssociations | Remove-DefaultAppAssociations]**

The following table provides a description of how each default application association servicing option can be used. These options are not case sensitive.

OPTION	DESCRIPTION
<b>/Get-Help</b> <b>/?</b>	When used immediately after a default application association servicing command-line option, information about the option and the arguments is displayed. Additional topics might become available when an image is specified.  Examples:  <b>Dism /image:C:\test\offline /Import-DefaultAppAssociations /?</b>  <b>Dism /online /Get-DefaultAppAssociations /?</b>
<b>/Export-DefaultAppAssociations:&lt;path_to_export_file&gt;</b>	Exports the default application associations from a running operating system to an .xml file.  Example:  <b>Dism.exe /Online /Export-DefaultAppAssociations:C:\AppAssoc.xml</b>

OPTION	DESCRIPTION
<b>/Get-DefaultAppAssociations</b>	<p>Displays the list of default application associations that have been set in the specified Windows image. You can use this option to verify that default application associations were successfully imported to the image.</p> <p>Examples:</p> <p><b>Dism.exe /Image:C:\test\offline /Get-DefaultAppAssociations</b></p> <p><b>Dism.exe /Online /Get-DefaultAppAssociations</b></p>
<b>/Import-DefaultAppAssociations:&lt;path_to_xml_file&gt;</b>	<p>Imports a set of default application associations to a specified Windows image from an .xml file. The default application associations will be applied for each user during their first logon.</p> <p>Examples:</p> <p><b>Dism.exe /Image:C:\test\offline /Import-DefaultAppAssociations:C:\AppAssoc.xml</b></p> <p><b>Dism.exe /Online /Import-DefaultAppAssociations:C:\AppAssoc.xml</b></p>
<b>/Remove-DefaultAppAssociations</b>	<p>Removes the default application associations from the specified Windows image.</p> <p>Examples:</p> <p><b>Dism.exe /Image:C:\test\offline /Remove-DefaultAppAssociations</b></p> <p><b>Dism.exe /Online /Remove-DefaultAppAssociations</b></p>

## Related topics

[What is DISM?](#)

[DISM Image Management Command-Line Options](#)

[Deployment Image Servicing and Management \(DISM\) Command-Line Options](#)

# DISM Languages and International Servicing Command-Line Options

1/18/2019 • 9 minutes to read

The international commands can be used to change international settings in Windows and Windows Preinstallation Environment (WinPE) images. You can also query existing settings in an offline or online Windows image.

The base syntax for servicing a Windows image using the Deployment Image Servicing and Management (DISM.exe) tool is:

```
DISM.exe {/Image:<path_to_offline_image_directory> | /Online} [dism_global_options]
{servicing_option} [<servicing_argument>]
```

There are three types of international servicing commands:

- **Get commands.** Retrieves a report of the international settings for an offline image or a running operating system.
- **Set commands.** Sets the different international settings for an offline image.
- **Gen-LangIni commands.** Generates the Lang.ini file that is used during Setup.

The following international servicing options are available for an offline image:

```
DISM.exe /Image:<path_to_offline_image_directory> [/Get-Intl] [/Set-UILang | /Set-UILangFallback |
/Set-SysLocale | /Set-UserLocale | /Set-InputLocale | /Set-AllIntl | /Set-Timezone | /Set-
SKUIntlDefaults | /Set-LayeredDriver] [/Gen-Langini | /Set-SetupUILang | /Distribution]
```

The following international servicing options are available for a running operating system:

**DISM.exe /Online /Get-Intl**

The following table provides a description of how each international servicing option can be used. These options are not case-sensitive.

OPTION/ARGUMENT	DESCRIPTION
Option: <b>/Get-Help /?</b>	<p>When used immediately after an international servicing command-line option, information about the option and the arguments is displayed. Additional topics might become available when an image is specified.</p> <p>Examples:</p> <p><b>Dism /image:C:\test\offline /Set-UILang /?</b></p> <p><b>Dism /online /Get-intl /?</b></p>

OPTION/ARGUMENT	DESCRIPTION
<p>Option: <b>/Get-Intl</b></p>	<p>Displays information about international settings and languages.</p> <p>Use the <b>/Online</b> option to display information about international settings and languages in the running operating system.</p> <p>Use the <b>/Image:&lt;path_to_offline_image_directory&gt;</b> option to display information about international settings and languages in the offline image.</p> <p>When used with the <b>/Distribution</b> options, information about international settings and languages in the distribution is displayed. The name of the folder in the distribution share is not validated. It will be reported as ...\\Langpacks&amp;lt;locale_name&gt;\\Lp.cab. Where &lt;locale_name&gt; is the name of the folder.</p> <div data-bbox="826 698 1429 893" style="border: 1px solid black; padding: 10px;"> <p><b>Note</b></p> <p>The user locale is reported only for offline images. The report does not include this setting for running operating systems.</p> </div> <p>Examples:</p> <p><b>Dism /online /Get-Intl</b></p> <p><b>Dism /image:C:\\test\\offline /Get-Intl</b></p> <p><b>Dism /image:C:\\test\\offline /distribution:C:\\windows_distribution /Get-Intl</b></p>
<p>Option: <b>/Set-UILang:</b></p> <p>Argument: &lt;<i>language_name</i>&gt;</p>	<p>Sets the default system user interface (UI) language. If the language is not installed in the Windows image, the command will fail.</p> <p>&lt;<i>language_name</i>&gt; specifies the name of the language to set as the default; for example, ja-JP.</p> <div data-bbox="826 1394 1429 1814" style="border: 1px solid black; padding: 10px;"> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>● This option doesn't support setting the UI to a language that's been installed as a .appx-based Language Experience Pack (LXP).</li> <li>● If you install a LIP by using a .cab file and specify it as the default system UI language, the LIP language will be set as the default UI language (or system locale) and the parent language will be set as the default system UI language (or install language).</li> </ul> </div> <p>Example:</p> <p><b>Dism /image:C:\\test\\offline /Set-UILang:fr-FR</b></p>

OPTION/ARGUMENT	DESCRIPTION
Option: <b>/Set-UILangFallback:</b> Argument: < <i>language_name</i> >	<p>Sets the fallback default language for the system UI in the offline Windows image. This setting is used only when the language specified by the <b>/Set-UILang</b> option is a partially localized language.</p> <p>&lt;<i>language_name</i>&gt; specifies the name of the language to set as the default fallback; for example, en-US.</p> <p>Example:</p> <pre>Dism /image:C:\test\offline /Set-UILangFallback:fr-FR</pre>
Option: <b>/Set-SysLocale:</b> Argument: < <i>locale_name</i> >	<p>Sets the language for non-Unicode programs (also called system locale) and font settings in the offline Windows image.</p> <p>&lt;<i>locale_name</i>&gt; specifies the name of the language and locale to set as the default language for non-Unicode; for example, en-US.</p> <div style="border: 1px solid black; padding: 10px;"> <p><b>Important</b></p> <p>You cannot set Unicode-only languages as the system locale. If you try, the <b>/Set-SysLocale</b> option will fail and the language for non-Unicode programs will not be changed.</p> </div> <p>Example:</p> <pre>Dism /image:C:\test\offline /Set-SysLocale:fr-FR</pre>
Option: <b>/Set-UserLocale:</b> Argument: < <i>locale_name</i> >	<p>Sets the "standards and formats" language (also called user locale) in the offline Windows image. The "standards and formats" language is a per-user setting that determines default sort order and the default settings for formatting dates, times, currency, and numbers.</p> <p>Example:</p> <pre>Dism /image:C:\test\offline /Set-UserLocale:fr-FR</pre>

OPTION/ARGUMENT	DESCRIPTION
<p>Option: <b>/Set-InputLocale:</b>            Argument: &lt;input_locale&gt;:&lt;keyboard_layout&gt;</p>	<p>Sets the input locales and keyboard layouts to use in the offline Windows image.</p> <p>The value of the &lt;input_locale&gt;:&lt;keyboard_layout&gt; pair can be one of the following:</p> <ul style="list-style-type: none"> <li>• &lt;language_id:keyboard_layout&gt;            For example, 0409:00000409</li> <li>• &lt;locale_name&gt;            For example, if you specify en-US as the locale name, The <b>Set-InputLocale:</b> option also sets the default keyboard layout defined for this locale.</li> </ul> <p>You can specify more than one value by using semicolons as separators. This is useful when you want to include support for multiple keyboards on a single computer. The first value will be set as the default keyboard.</p> <p>The valid keyboard layouts that can be configured on your computer are listed in the following registry key.</p> <p><b>HKEY_LOCAL_MACHINE    \SYSTEM\CurrentControlSet\Control\Keyboard    Layouts</b></p> <p>For a list of the values, see <a href="#">Default Input Locales</a> and <a href="#">Default Keyboard Settings</a>.</p> <p>Use the hexadecimal value of the language ID and keyboard layout that you intend to configure.</p> <p>This parameter is optional.</p> <p>Example:</p> <pre>Dism /image:C:\test\offline /Set-  InputLocale:fr-fr</pre> <pre>Dism /image:C:\test\offline /Set-  InputLocale:0410:00010410</pre>

OPTION/ARGUMENT	DESCRIPTION
<p>Option: <b>/Set-AllIntl:</b>            Argument: &lt;<i>language_name</i>&gt;</p>	<p>Sets the default system UI language, the language for non-Unicode programs, the "standards and formats" language, and the input locales and keyboard layouts to the specified language in the offline Windows image. This option specifies the language value for the following:</p> <ul style="list-style-type: none"> <li>• UI language</li> <li>• System locale</li> <li>• User locale</li> <li>• Input locale</li> </ul> <p>If used with any of the options that specify the individual language or locales, then the individual settings take precedence.</p> <p>&lt;<i>language_name</i>&gt; specifies the language name and locale code; for example, en-US, es-ES, or fr-FR.</p> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <ul style="list-style-type: none"> <li>• This option doesn't support setting the UI to a language that's been installed as a .appx-based Language Experience Pack (LXP).</li> <li>• If you install a LIP by using a .cab file and specify it as the default system UI language, the LIP language will be set as the default UI language (or system locale) and the parent language will be set as the default system UI language (or install language).</li> </ul> </div>
<p>Option: <b>/Set-TimeZone:</b>            Argument: &lt;<i>timezone_name</i>&gt;</p>	<p>Sets the default time zone in a Windows image. Before setting the time zone, DISM verifies that the specified time zone string is valid for the image.</p> <p>&lt;<i>timezone_name</i>&gt; specifies the name of the time zone to use; for example, Pacific Standard Time. For a complete list of time-zone strings, see the Windows® Unattended Setup Reference. On a computer that is running Windows 7, you can use the tzutil command-line tool to list the time zone for that computer. The tzutil tool is installed by default on Windows 7.</p> <p>The name of the time zone must exactly match the name of the time zone settings in the registry in <b>HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Time Zones</b>.</p> <p>If you add a custom time zone to your computer, you can specify that custom time-zone string.</p> <p>Example:</p> <pre>Dism /image:C:\test\offline /Set-TimeZone:"W. Europe Standard Time"</pre>

OPTION/ARGUMENT	DESCRIPTION
<p>Option: <b>/Set-SKUIntlDefaults:</b> Argument: &lt;language_name&gt;</p>	<p>Sets the default system UI language, the language for non-Unicode programs, the "standards and formats" language, and the input locales, keyboard layouts, and time zone values in an offline Windows image to the default value specified by &lt;language_name&gt;. The <b>/Set-SKUIntlDefaults</b> option does not change the keyboard driver for Japanese and Korean keyboards. You must use the <b>/Set-LayeredDriver</b> option to change this.</p> <p>Use <b>/Set-SKUIntlDefaults</b> to change all the international settings in an offline Windows image to match the default values that are set during retail installations. For more information about the default values of each language pack, see <a href="#">Default Input Locales for Windows Language Packs</a>.</p> <p>This parameter is optional. If combined with one of the settings earlier in this section, the individual setting takes priority.</p> <p>If the language passed matches a Unicode-only locale setting, the system locale will not be changed but the command will not fail.</p> <p>Example: <b>Dism /image:C:\test\offline /Set-SKUIntlDefaults:fr-FR</b></p>
<p>Option: <b>/Set-LayeredDriver:</b> Arguments: &lt;1-6&gt;</p>	<p>Specifies a keyboard driver to use for Japanese or Korean keyboards.</p> <p>In Japan, many retail users have 106-key keyboards, whereas others have 101- or 102-key keyboards. In Korea, there are several different types of keyboards, some with different numbers of keys.</p> <p>The possible values for these settings are [1-6]:</p> <ol style="list-style-type: none"> <li>1. Specifies the PC/AT Enhanced Keyboard (101/102-Key).</li> <li>2. Specifies the Korean PC/AT 101-Key Compatible Keyboard/MS Natural Keyboard (Type 1).</li> <li>3. Specifies the Korean PC/AT 101-Key Compatible Keyboard/MS Natural Keyboard (Type 2).</li> <li>4. Specifies the Korean PC/AT 101-Key Compatible Keyboard/MS Natural Keyboard (Type 3).</li> <li>5. Specifies the Korean Keyboard (103/106 Key).</li> <li>6. Specifies the Japanese Keyboard (106/109 Key).</li> </ol> <p>Example: <b>Dism /image:C:\test\offline /Set-LayeredDriver:1</b></p>

OPTION/ARGUMENT	DESCRIPTION
<p>Option: <b>/Gen-LangINI:</b></p> <p>Argument: &lt;<i>language_name</i>&gt;</p>	<p>Generates a new Lang.ini file, which is used by Setup to define the language packs inside the image and outside in the distribution. It also defines the default UI language for Setup.</p> <p>The new Lang.ini file will be added to the Sources folder of the Windows distribution.</p> <div style="border: 1px solid black; padding: 5px;"> <p><b>Note</b></p> <p>You will not be prompted for permission to overwrite an existing Lang.ini file. The existing Lang.ini file will be overwritten automatically.</p> </div> <p>You must specify an offline Windows image (<b>/Image:&lt;path_to_offline_image.wim&gt;</b>) and a distribution (<b>/Distribution:&lt;path_to_distribution_directory&gt;</b>).</p> <p>Example:</p> <pre>Dism /image:C:\test\offline /Gen-LangINI /distribution:C:\windows_distribution</pre>
<p>Option: <b>/Set-SetupUILang:</b></p> <p>Argument: &lt;<i>language_name</i>&gt;</p>	<p>Defines the default language that will be used by Setup. If this language cannot be used, Setup automatically uses English.</p> <p>This is an optional command. If not used, the default UI language in the image will be used. If the language is not present, the first language in the list of present languages will be used.</p> <p>Example:</p> <pre>Dism /image:C:\test\offline /Set-SetupUILang:fr-FR /distribution:C:\windows_distribution</pre>
<p>Option: <b>/Distribution:</b></p> <p>Argument: &lt;<i>path_to-distribution_directory</i>&gt;</p>	<p>Specifies the path to the Windows distribution. The Windows distribution is a copy of the content that releases on the Windows product DVD. This option is only for use with the <b>/Get-Intl</b> and <b>/Gen-LangINI</b> option if there are external language packs.</p> <p>Example:</p> <pre>Dism /image:C:\test\offline /Gen-LangINI /distribution:C:\windows_distribution</pre>

## Limitations

- The DISM International servicing commands cannot be used on a Windows Vista or a Windows Server 2008 image. For information about servicing Windows Vista and Windows Server 2008 images, see the Windows Vista SP1 release of the Windows OEM Preinstallation Kit (Windows OPK) or Windows Automated Installation Kit (Windows AIK).
- You cannot use other servicing commands on the same command line with international servicing commands.

- You cannot set a Unicode-only language as the system locale.

The following languages are Unicode-only:

LANGUAGE	COUNTRY/REGION
Amharic	Ethiopia
Armenian	Armenia
Assamese	India
Bangla	Bangladesh
Bangla	India (Bengali Script)
Divehi	Maldives
Georgian	Georgia
Gujarati	India (Gujarati Script)
Hindi	India
Inuktitut (Syllabics)	Canada
Kannada	India (Kannada Script)
Kazakh	Kazakhstan
Khmer	Cambodia
Konkani	India
Lao	Lao PDR
Malayalam	India (Malayalam Script)
Maltese	Malta
Maori	New Zealand
Marathi	India
Mongolian (Mongolian)	PRC
Nepali	Federal Democratic Republic of Nepal
Odia	India (Odia Script)
Pashto	Afghanistan
Punjabi	India (Gurmukhi Script)

LANGUAGE	COUNTRY/REGION
Sanskrit	India
Sinhala	Sri Lanka
Syriac	Syria
Tamil	India
Telugu	India (Telugu Script)
Tibetan	PRC
Yi	PRC

- Do not install a language pack after an update.

If you install an update (hotfix, general distribution release [GDR], or service pack [SP]) that contains language-dependent resources before you install a language pack, the language-specific changes contained in the update are not applied. Always install language packs before installing updates.

- When specifying a time zone by using **/Set-TimeZone:<timezone\_name>** you must use straight quotation marks for multiple words. For example, **/Set-TimeZone:"Pacific Standard Time"**. If you copy and paste the time zone name, including quotation marks, from a Microsoft® Word document, the quotation marks might not be recognized and the command line might fail.
- If you are servicing an international image, and your host environment does not support the language in that image, you might not be able to read an error message that originates from the international image.

## Related topics

[What is DISM?](#)

[DISM Image Management Command-Line Options](#)

[Deployment Image Servicing and Management \(DISM\) Command-Line Options](#)

# DISM Capabilities Package Servicing Command-Line Options

1/18/2019 • 2 minutes to read

Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) only. Use Deployment Image Servicing and Management (DISM.exe) to service Windows capabilities. Capabilities are a Windows package type allows you to request services like .NET or languages without specifying the version. Use DISM to search multiple sources like Windows Update or your corporate servers to find and install the latest version.

To see the available capabilities, go to [Features On Demand](#).

## DISM Command-Line Options

Here's how each DISM option can be used. These options are not case sensitive.

Note, each of these commands requires either the **/Online** or **/Image:<path\_to\_offline\_image\_file>** argument.

OPTIONS	DESCRIPTION
<b>/Add-Capability</b> <b>/CapabilityName:&lt;capability_name&gt;</b> <b>[/Source:&lt;source&gt;] [/LimitAccess]</b>	<p>Adds a capability to an image.</p> <p>Example:</p> <p><b>Dism /Online /Add-Capability /CapabilityName:Language.Basic~~~en-US~0.0.1.0</b></p> <div style="border: 1px solid black; padding: 10px;"><p><b>Note</b> DISM checks for the source files in the following order:</p><ol style="list-style-type: none"><li>1. If <b>/Source</b> is specified, DISM looks in the specified locations first.</li><li>2. If <b>/Source</b> is not specified, or if the source files are not found in the specified locations, DISM checks to see if a group policy is set. If it is, DISM checks the locations specified by the group policy.</li><li>3. If the files still aren't found, and if DISM is working against an online image, and if <b>/LimitAccess</b> is not specified, it looks for the files on Windows Update.</li></ol></div> <p><b>/Source:</b> Allows you to choose a location, such as a server, where the capability source files are located. You can use multiple <b>/Source</b> arguments.</p> <p>Example:</p> <p><b>Dism /Online /Add-Capability /CapabilityName:Language.Basic~~~en-US~0.0.1.0 /Source:\server\share /Source:\server2\share</b></p> <p><b>/LimitAccess:</b> Tells DISM to not check Windows Update or Windows Server Update Services for the capability source files.</p> <p>Example:</p> <p><b>Dism /Online /Add-Capability /CapabilityName:Language.Basic~~~en-US~0.0.1.0 /Source:\server\share /LimitAccess</b></p>

OPTIONS	DESCRIPTION
<b>/Get-Capabilities</b>  	Get capabilities in the image. Example: <b>DISM /Online /Get-Capabilities</b>
<b>/Get-CapabilityInfo</b>  <b>/CapabilityName:&lt;capability_name&gt;</b>	Get information about a specific capability. Example: <b>DISM /Online /Get-CapabilityInfo</b> <b>/CapabilityName:Language.Basic~en-US~0.0.1.0</b>
<b>/Remove-Capability</b>  <b>/CapabilityName:&lt;capability_name&gt;</b>	Example: <b>Dism /Online /Remove-Capability</b> <b>/CapabilityName:Language.Basic~~~en-US~0.0.1.0</b> Example: <b>Dism /Image:C:\test\offline /Remove-Capability</b> <b>/CapabilityName:Language.Basic~en-US~0.0.1.0</b>

## Related topics

[Features On Demand](#)

[DISM - Deployment Image Servicing and Management Technical Reference for Windows](#)

[What is DISM?](#)

[DISM Global Options for Command-Line Syntax](#)

[DISM Operating System Package Servicing Command-Line Options](#)

[DISM Languages and International Servicing Command-Line Options](#)

# DISM Windows Edition-Servicing Command-Line Options

1/18/2019 • 3 minutes to read

You can use the Windows edition-servicing commands to change one edition of Windows to a higher edition in the same edition family. The edition packages for each potential target edition are staged in a Windows image. This is referred to as an edition-family image. Because the target editions are staged, you can service a single image, and the updates will be applied appropriately to each edition in the image. This can help reduce the number of images that you have to manage, but it might increase the factory time or end-user time that must be spent in the **specialize** configuration pass.

Offline changes do not require a product key. If you change to a higher edition using offline servicing, you can add the product key using one of the following methods:

- Enter the product key during the out-of-box experience (OOBE).
- Use an unattended answer file to enter the product key during the **specialize** configuration pass.
- Use Deployment Image Servicing and Management (DISM) and the Windows edition-servicing command-line option **/Set-ProductKey** after you set the edition offline.

## Command-line Syntax

The base syntax for servicing a Windows image using DISM is:

```
DISM.exe {/Image:<path_to_image_directory> | /Online} [dism_global_options] {servicing_option} [<servicing_argument>]
```

You can use the following edition-servicing options on an offline image to list editions or to change a Windows image to a higher edition:

```
DISM.exe /Image:<path_to_image_directory> {/Get-CurrentEdition | /Get-TargetEditions | /Optimize-Image | /WIMBoot | /Set-Edition | /Set-ProductKey:<product_key>}
```

The following edition-servicing options are available for a running Windows operating system:

```
DISM.exe /Online {/Get-CurrentEdition | /Get-TargetEditions | /Set-ProductKey:<product_key> | /Set-Edition:<target_edition> {/GetEula:< path> | /AcceptEula | /ProductKey:<product_key>}}
```

The following table provides a description for how each edition-servicing option can be used. These options are not case-sensitive.

OPTION	DESCRIPTION
<b>/Get-Help</b> <b>/?</b>	When used immediately after an edition-servicing command-line option, information about the option and the arguments is displayed. Additional Help topics might become available when an image is specified.  Examples:  <b>Dism /Image:C:\test\offline /Get-CurrentEdition /?</b> <b>Dism /Online /Get-CurrentEdition /?</b>

OPTION	DESCRIPTION
<b>/Get-CurrentEdition</b>	<p>Displays the edition of the specified image.</p> <p>Examples:</p> <p><b>Dism /Image:C:\test\offline /Get-CurrentEdition</b></p> <p><b>Dism /Online /Get-CurrentEdition</b></p>
<b>/Get-TargetEditions</b>	<p>Displays a list of Windows editions that an image can be changed to.</p> <p>Examples:</p> <p><b>Dism /Image:C:\test\offline /Get-TargetEditions</b></p> <p><b>Dism /Online /Get-TargetEditions</b></p>
<p><b>/Set-Edition:&lt;target_edition_ID&gt; [&lt;/GetEula:&lt;path   /AcceptEula /ProductKey:&lt;product_key&gt;&gt;]</b></p>	<p>Use the <b>/Set-Edition</b> option with no arguments to change an offline Windows image to a higher edition.</p> <p>To change an online Windows Server operation system to a higher edition, you must use the <b>/Set-Edition</b> option with the <b>/AcceptEula</b> and <b>/ProductKey</b> arguments.</p> <div style="border: 1px solid black; padding: 10px;"> <p><b>Important</b></p> <p>You should not use the <b>/Set-Edition</b> option on an image that has already been changed to a higher edition. It is recommended that you use this option on the lowest edition available in the edition family.</p> </div> <p>Use <b>/GetEula</b> on an online image to copy the end-user license agreement to a specified path.</p> <p>The <b>/AcceptEula</b> argument accepts the end-user license agreement and is required in order to change the Windows edition on an online image.</p> <p>Example:</p> <p><b>Dism /Image:C:\test\offline /Set-Edition:&lt;edition name&gt;</b></p> <p>On a running Windows Server operating system only:</p> <p><b>Dism /online /Set-Edition:&lt;edition name&gt; /GetEula:c:\eulapath</b></p> <p><b>Dism /online /Set-Edition:&lt;edition name&gt; /AcceptEula /ProductKey:12345-67890-12345-67890-12345</b></p> <p>Where <i>&lt;edition name&gt;</i> is the higher edition that you want to change to.</p>

OPTION	DESCRIPTION
<b>/Set-ProductKey:&lt;productKey&gt;</b>	<p>The <b>/Set-ProductKey</b> option can only be used to enter the product key for the current edition in an offline Windows image after you change an offline Windows image to a higher edition using the <b>/Set-Edition</b> option.</p> <p>Example:</p> <pre>Dism /Image:C:\test\offline /Set-ProductKey:12345-67890-12345-67890-12345</pre>

## Limitations

- If you do not enter the product key when you set the edition of your offline image, you must either enter the product key during OOBE, or use an unattended answer file to enter the product key during the **specialize** configuration pass.
- You cannot use edition-servicing commands on a Windows Preinstallation Environment (Windows PE) image.
- To maintain edition-specific customizations, you should apply edition-specific answer files after the edition upgrade.
- If you want to run the **/Set-Edition** option against a 64-bit image with more than 30 language packs, you must run it from a 64-bit computer. Otherwise, you might receive an out-of-memory error. This limitation only exists if you are manipulating a 64-bit image from a 32-bit computer. This limitation does not exist when you run this option on a computer that matches the architecture of the image.
- You cannot set a Windows image to a lower edition. The lowest edition will not appear when you run the **/Get-TargetEditions** option.
- You should not use the **/Set-Edition** option on an image that has already been changed to a higher edition.

## Related topics

[What is DISM?](#)

[DISM Image Management Command-Line Options](#)

[Deployment Image Servicing and Management \(DISM\) Command-Line Options](#)

[Change the Windows Image to a Higher Edition Using DISM](#)

# DISM Driver Servicing (.inf) Command-Line Options

1/18/2019 • 3 minutes to read

Use DISM with INF-style drivers to add, remove, or list drivers to an online or offline Windows image (.wim). Microsoft Windows Installer or other driver package types (such as .exe files) are not supported.

You can specify a directory where the driver INF files are located, or you can point to a driver by specifying the name of the INF file.

The base syntax for servicing a Windows image using DISM is:

**DISM.exe {/Image:<path\_to\_image\_directory> | /Online} [dism\_global\_options] {servicing\_option} [<servicing\_argument>]**

The following driver servicing options are available for an offline image.

**DISM.exe /image:<path\_to\_image\_directory> [/Get-Drivers | /Get-DriverInfo | /Add-Driver | /Remove-Driver | /Export-Driver]**

The following driver servicing options are available for a running operating system.

**DISM.exe /Online [/Get-Drivers | /Get-DriverInfo | /Export-Driver]**

The following table provides a description of how each driver servicing option can be used. These options are not case sensitive.

OPTION/ARGUMENT	DESCRIPTION
Option: <b>/Get-Help /?</b>  Arguments: <b>/All</b> <b>/Format:{Table   List}</b>	When used immediately after a driver servicing command-line option, information about the option and the arguments is displayed. Additional topics might become available when an image is specified.  Examples: <b>Dism /image:C:\test\offline /Add-Driver /?</b> <b>Dism /online /Get-Drivers /?</b>
Option: <b>/Get-Drivers</b>  Arguments: <b>/All</b> <b>/Format:{Table   List}</b>	Displays basic information about driver packages in the online or offline image.  By default, only third-party drivers will be listed. Use the <b>/all</b> argument to display information about default drivers and third-party drivers. Use the <b>/Format:Table</b> or <b>/Format:List</b> argument to display the output as a table or a list.  If you point to an image, you can determine what drivers are in the image, in addition to the state of the drivers (installed or staged).  Example: <b>Dism /image:C:\test\offline /Get-Drivers</b> <b>Dism /online /Get-Drivers</b>

OPTION/ARGUMENT	DESCRIPTION
<p>Option: <b>/Get-DriverInfo</b></p> <p>Arguments:</p> <p><b>/Driver:&lt;installed_INF_FileName&gt;</b></p> <p><b>/Driver:&lt;path_to_driver.inf&gt;</b></p>	<p>Displays detailed information about a specific driver package.</p> <p>You can point to an INF file installed in the image, or one that is not yet installed. You can specify the name of the uninstalled driver or the third-party driver in the device driver store. Installed third-party drivers in the driver store will be named Oem0.inf, Oem1.inf, and so on. This is referred to as the published name.</p> <p>You can specify multiple drivers on the command line by using the <b>/driver</b> option multiple times.</p> <p>Example:</p> <p>First, use the <b>/Get-DriverInfo</b> option so that you can identify a driver INF file. Then run the following command:</p> <pre>Dism /image:C:\test\offline /Get-DriverInfo /driver:&lt;path_to_driver.inf&gt;</pre> <pre>Dism /online /Get-DriverInfo /driver:C:\test\drivers\usb\usb.inf</pre>
<p>Option: <b>/Add-Driver</b></p> <p>Arguments:</p> <p><b>/Driver:&lt;folder_containing_INF&gt;</b></p> <p><b>/Driver:&lt;path_to_driver.inf&gt;</b></p> <p><b>/Recurse</b></p> <p><b>/ForceUnsigned</b></p>	<p>Adds third-party driver packages to an offline Windows image.</p> <p>When you use the <b>/Driver</b> option to point to a folder, INF files that are not valid driver packages are ignored. These files are reported on the console when the command runs, and a warning is included in the log file. You will not receive an error message.</p> <p>If you point to a path and use the <b>/Recurse</b> option, all subfolders are queried for drivers to add.</p> <p>For testing purposes you can use <b>/ForceUnsigned</b> to add unsigned drivers and override the requirement that drivers installed on X64-based computers must have a digital signature. For more information about driver signing requirements, see <a href="#">Device Drivers and Deployment Overview</a>.</p> <p>Examples:</p> <pre>Dism /image:C:\test\offline /Add-Driver /driver:C:\test\drivers&lt;/strong&gt;</pre> <pre>Dism /image:C:\test\offline /Add-Driver /driver:C:\test\drivers /recurse</pre> <pre>Dism /image:C:\test\offline /Add-Driver /driver:C:\test\drivers\mydriver.inf</pre> <pre>Dism /image:C:\test\offline /Add-Driver /driver:C:\test\drivers\mydriver.inf /ForceUnsigned</pre>

OPTION/ARGUMENT	DESCRIPTION
<p>Option: <b>/Remove-Driver</b></p> <p>Arguments:</p> <p><b>/Driver:&lt;published_name&gt;</b></p>	<p>Removes third-party drivers from an offline image.</p> <p>When third-party drivers are added, they are named Oem0.inf, Oem1.inf, and so on. You must specify the &lt;published name&gt; (for example, Oem1.inf) to remove the driver. You cannot remove default drivers.</p> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p><b>Warning</b></p> <p>Removing a boot-critical driver package can make the offline Windows image unbootable.</p> </div> <p>You can specify multiple drivers on the command line by using the <b>/Driver</b> option multiple times.</p> <p>Examples:</p> <pre>Dism /image:C:\test\offline /Remove-Driver /driver:oem1.inf</pre> <pre>Dism /image: C:\test\offline /Remove-Driver /driver:oem1.inf /driver:oem2.inf</pre>
<p>Option: <b>/Export-Driver</b></p> <p>Arguments:</p> <p><b>/Destination:&lt;path_to_destination_folder&gt;</b></p>	<p>Exports all third-party driver packages from a Windows image to a destination path. The exported drivers can then be injected to an offline image by running the <b>DISM Add-Driver</b> command. This command is new for Windows 8.1 Update.</p> <p>Examples:</p> <pre>DISM /Online /Export-Driver /Destination:C:\destpath</pre> <pre>DISM /Image:C:\test\offline /Export-Driver /Destination:C:\destpath</pre>

## Limitations

- The driver servicing command supports only .inf files. Windows Installer or other driver package types (such as .exe files) are not supported.
- Drivers are installed in the order that they are listed in the command line. In the following example, 1.inf, 2.inf, and 3.inf will be installed in the order that they are listed in the command line.

```
Dism /Image:C:\test\offline /Add-Driver /Driver:C:\test\drivers\1.inf /Driver:C:\test\drivers\2.inf
/Driver:C:\test\drivers\3.inf
```

## Related topics

[What is DISM?](#)

[DISM Image Management Command-Line Options](#)

[Deployment Image Servicing and Management \(DISM\) Command-Line Options](#)

# DISM Unattended Servicing Command-Line Options

1/18/2019 • 2 minutes to read

If you are installing multiple packages to a Windows® image, use DISM to apply an unattend.xml answer file to the image. Some packages require other packages to be installed first. If there is a dependency requirement, the best way to ensure the correct order of the installation is by using an answer file. When you use DISM to apply an unattend.xml answer file to an image, the unattended settings in the **offlineServicing** configuration pass are applied to the Windows image.

The base syntax for servicing a Windows image using DISM is:

**DISM.exe {/Image:<path\_to\_image\_directory> | /Online} [dism\_global\_options] {servicing\_option} [<servicing\_argument>]**

The following servicing options are available to apply an unattend.xml answer file to an offline Windows image:

**DISM.exe /Image:<path\_to\_image\_directory> /Apply-Unattend:<path\_to\_unattend.xml>**

The following servicing options are available to apply an unattend.xml answer file to a running operating system:

**DISM.exe /Online /Apply-Unattend:<path\_to\_unattend.xml>**

The following table provides a description of how an unattended servicing option can be used. These options are not case sensitive.

OPTION	DESCRIPTION
<b>/Get-Help</b> <b>/?</b>	When used immediately after an unattended servicing command-line option, information about the option and the arguments is displayed. Additional topics might become available when an image is specified.  Examples:  <b>Dism /online /Apply-Unattend /?</b> <b>Dism /image:C:\test\offline /Apply-Unattend /?</b>

OPTION	DESCRIPTION
<b>/Apply-Unattend:</b> <path_to_unattend.xml>	<p>Applies an Unattend.xml file to an image.</p> <p>If you are updating device drivers using an unattended answer file, you must apply the answer file to an offline image and specify the settings in the <b>offlineServicing</b> configuration pass.</p> <p>If you are updating packages or other settings using an unattended answer file, you can apply the answer file to an offline or online image. Specify the settings in the <b>offlineServicing</b> configuration pass.</p> <p>Example:</p> <pre>Dism /image:C:\test\offline /Apply-Unattend:C:\test\answerfiles\myunattend.xml</pre> <pre>Dism /online /Apply-Unattend:C:\test\answerfiles\myunattend.xml</pre>

## Limitations

- You cannot use other servicing commands on the same command line with unattended servicing commands.
- Only a single unattend.xml answer file can be specified on any command line.
- When you add packages to an image using an unattended answer file, the applicability of the package will not be checked. The answer file will be applied, and the operation will complete even if there are packages specified in the answer file which do not apply to the image. If you have to check the applicability of a package when you add it to an image, use the **DISM** command together with the **/Add-Package** option without the **/ignorecheck** option. For more information, see [DISM Operating System Package Servicing Command-Line Options](#).
- If you are updating device drivers using an unattended answer file, you must apply the answer file to an offline image.
- When you use DISM.exe to apply an answer file to a running operating system, the answer file should only contain elements in the **offlineServicing** configuration pass. This is because some settings in the Specialize configuration pass might be applied to the operating system. We recommend that the answer file that you use with DISM only contain settings in the **offlineServicing** configuration pass.
- The recommended way to author answer files is to create them in Windows System Image Manager (Windows SIM). However, if you use a manually authored answer file, you must validate the answer file in Windows SIM to verify that it works. For more information, see [Best Practices for Authoring Answer Files](#).
- When you apply an answer file by using DISM, the answer file is not cached on the target computer.

## Related topics

[What is DISM?](#)

[DISM Image Management Command-Line Options](#)

[DISM Languages and International Servicing Command-Line Options](#)

[DISM Operating System Package Servicing Command-Line Options](#)

[DISM Windows Edition-Servicing Command-Line Options](#)

## [DISM Driver Servicing Command-Line Options](#)

# DISM Windows PE Servicing Command-Line Options

1/18/2019 • 2 minutes to read

You can update Windows Preinstallation Environment (WinPE) by adding drivers, language packs, list packages and prepare the Windows PE image for deployment.

You must use these commands on a mounted WinPE image. See [WinPE: Mount and Customize](#). These options cannot be used with an online, running version of Windows PE.

In addition to the [DISM global options](#), the following Windows PE servicing options are available. These options are not case sensitive.

**DISM.exe /Image: <path\_to\_image\_directory> [/Get-PESettings | /Get-ScratchSpace | /Get-TargetPath | /Set-ScratchSpace:<size\_of\_ScratchSpace> | /Set-TargetPath :<target\_path> ]**

OPTION	DESCRIPTION
<b>/Get-PESettings</b>	Displays a list of Windows PE settings in the Windows PE image. The list includes scratch space settings and target path settings. For example:  <b>Dism /image:C:\test\offline /Get-PESettings</b>
<b>/Get-ScratchSpace</b>	Retrieves the configured amount of Windows PE system volume scratch space. This setting represents the amount of writeable space available on the Windows PE system volume when booted in ramdisk mode. For example:  <b>Dism /image:C:\test\offline /Get-ScratchSpace</b>
<b>/Get-TargetPath</b>	Retrieves the target path of the Windows PE image. The target path represents a path to the root of the Windows PE image at boot time. For example:  <b>Dism /image:C:\test\offline /Get-TargetPath</b>
<b>/Set-ScratchSpace:&lt;size_of_ScratchSpace&gt;</b>	Sets the available scratch space, in megabytes. Valid values are 32, 64, 128, 256 and 512. For example:  <b>Dism /image:C:\test\offline /set-ScratchSpace:128</b>

OPTION	DESCRIPTION
<b>/Set-TargetPath :&lt;target_path&gt;</b>	<p>For hard disk boot scenarios, this option sets the location of the Windows PE image on the disk.</p> <p>Note the following limitations when setting the target path:</p> <ul style="list-style-type: none"> <li>• The path must be at least three characters and no longer than 32 characters</li> <li>• The path must start with a letter (any letter from C to Z)</li> <li>• The drive letter must be followed by :</li> <li>• The remainder of the path must not contain any invalid characters, such as Unicode characters</li> <li>• The path must be absolute, no "." or ".." elements</li> <li>• The path must not contain any blank spaces or "&amp;quot;"</li> </ul> <p>For example:</p> <p><b>Dism /image:C:\test\offline /Set-TargetPath:X:</b></p>

## Related topics

[Windows PE for Windows 10](#)

[Wpeutil Command-Line Options](#)

[What is DISM?](#)

[DISM Image Management Command-Line Options](#)

[Deployment Image Servicing and Management \(DISM\) Command-Line Options](#)

# DISM operating system uninstall command-line options

6/4/2019 • 2 minutes to read

## IMPORTANT

OEMs shouldn't use this setting in imaging or manufacturing scenarios. This setting is for IT administrators.

Windows gives a user the ability to uninstall and roll back to a previous version of Windows. You can use DISM to:

- Find out how many days after an upgrade that an OS can be uninstalled
- Initiate an uninstall
- Remove the ability for a user to uninstall a Windows upgrade
- Set the number of days that a user has to uninstall a Windows upgrade

## /Get-OSUninstallWindow

Run this command against an online image to see how many days after an upgrade that an uninstall can be initiated.

Syntax:

```
DISM /Online /Get-OSUninstallWindow
```

## /Initiate-OSUninstall

Run this command against an online image to revert a PC to a previous Windows installation.

Syntax:

```
DISM /Online /Initiate-OSUninstall
```

## /Remove-OSUninstall

Run this command against an online image to remove the ability to roll back a PC to a previous installation of Windows.

Syntax:

```
DISM /Online /Remove-OSUninstall
```

## /Set-OSUninstallWindow

Run this command against an online image to set the number of days after an upgrade that an uninstall can be initiated.

Syntax:

```
DISM /Online /Set-OSUninstallWindow /Value:<days>
```

**TIP**

If value passes anything <2 or >60, the default value of 10 will be set

Example:

```
DISM /Online /Set-OSUninstallWindow /Value:30
```

## Related topics

[DISM Image Management Command-Line Options](#)

[Deployment Image Servicing and Management \(DISM\) Command-Line Options](#)

# DISM Reference (Deployment Image Servicing and Management)

1/18/2019 • 2 minutes to read

Deployment Image Servicing and Management (DISM) is a command-line tool that is used to service Windows® images offline before deployment. You can use it to install, uninstall, configure, and update Windows features, packages, drivers, and international settings. Subsets of the DISM servicing commands are also available for servicing a running operating system. For more information, see [What is DISM?](#).

## In This Section

<a href="#">Deployment Image Servicing and Management (DISM) Command-Line Options</a>	Lists the command-line options for managing and servicing a Windows image with the Dism.exe tool.
<a href="#">DISM Configuration List and WimScript.ini Files</a>	Describes how to create a configuration list to exclude files and folders from an image capture or compression.
<a href="#">Deployment Image Servicing and Management (DISM) Best Practices</a>	Describes some best practices related to servicing a Windows image. We recommend that you implement these practices wherever possible.
<a href="#">Service a Windows PE Image with DISM</a>	Describes information specific to using DISM to configure a Windows PE image.
<a href="#">DISM Supported Platforms</a>	Describes the different operating systems and architectures supported by DISM.
<a href="#">Configure a Windows Repair Source</a>	Describes how to configure and maintain a Windows image repair source to use within your network. The repair source can be used to restore Windows features or to repair a corrupted Windows image.

## Related topics

[What is DISM?](#)

[DISM How-to Topics \(Deployment Image Servicing and Management\)](#)

# DISM Configuration List and WimScript.ini Files

1/18/2019 • 2 minutes to read

The Deployment Image Servicing and Management (DISM) tool is a command-line tool that you can use to capture and apply Windows images. You can create a configuration list file to determine the following:

- Which files and folders must be excluded from the capture process when you use the **/Capture-Image** option with the DISM tool.
- Which folders, files, and file types must be excluded from the compression process when you use the **/Compress** argument.

The **/ConfigFile** argument enables you to customize specific compression, capture, and boundary alignment actions for each file and folder when you capture an image using DISM.exe. You can create a configuration list (.ini) file by using a text editor, such as Notepad.

## Creating a Configuration List File

The following sections appear in the DISM configuration list file.

SECTION	DESCRIPTION
[ExclusionList]	Enables you to define the files and folders to exclude when you use the <b>/Capture-Image</b> option.
[ExclusionException]	Enables you to override the default exclusion list when you use the <b>/Capture-Image</b> option.
[CompressionExclusionList]	Enables you to define the specific files and folders, and also to specify file types, to exclude when you use the <b>/Compress</b> argument. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"><p><b>Note</b></p><p>You can use file or folder matching to exclude a file from compression. You can provide a full path match, or you can use wildcard characters (.). For example, you can use <code>\WINDOWS\inf.pnf</code> to match a specific type of file, or <code>\WINDOWS\inf*</code> to match a whole folder.</p></div>

### Default Exclusion List

By default, the DISM.exe tool will exclude the following files.

```
[ExclusionList]
\$ntfs.log
\hiberfil.sys
\pagefile.sys
\swapfile.sys
\System Volume Information
\RECYCLER
\Windows\CSC

[CompressionExclusionList]
*.mp3
*.zip
*.cab
\WINDOWS\inf\*.pnf
```

### Exclusion List Guidelines

- You can only use wildcard characters in the last component in a file path that does not begin with a backslash. For example:

```
myfolder\*.txt
```

- You can use a preceding backslash to limit file-matching and directory-matching relative to the root directory. For example, you can use this exclusion list:

```
\myfolder
\folder\subfolder
```

This list will exclude the following files and directories when you capture the "C:\" drive:

```
C:\myfolder
C:\folder\subfolder
```

However, DISM will not exclude files or directories that are contained in the following example.

```
C:\main\myfolder
C:\data\folder\subfolder
```

- You can override the default exclusion list by using the [ExclusionException] section. For example:

```
[ExclusionException]
\pagefile.sys
\System Volume Information
```

- If an explicit [ExclusionException] section is provided in the WIM configuration file, it will always take precedence over the [Exclusion List] section.
- You cannot override the default compression exclusion list by using the [ExclusionException] section.

## Using the Configuration File

If you create a custom-named configuration file and store it outside the DISM directory, you can use the DISM command to run the file. At a command prompt, open the DISM directory. For example:

```
Dism /Capture-Image /ImageFile:install.wim /CaptureDir:D:\ /Name:Drive-D /ConfigFile:<configuration list>
```

or

```
Dism /Append-Image /ImageFile:install.wim /CaptureDir:D:\ /Name:Drive-D /ConfigFile:<configuration list>
```

where *<configuration list>* provides the complete directory location for the configuration file. For example, `c:\imaging\configuration_list.ini`. You must use either the **/Capture-Image** option to create a new .wim file or the **/Append-Image** option to append an existing .wim file.

## Related topics

[DISM Image Management Command-Line Options](#)

# Deployment Image Servicing and Management (DISM) Best Practices

1/18/2019 • 6 minutes to read

This section describes some best practices related to servicing a Windows image. We recommend that you implement these practices wherever possible.

## Elevate Permissions for Command-Line Tools

Many deployment command-line tools, including Deployment Image Servicing and Management (DISM), require elevated permissions.

Make sure that you have elevated permissions. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.

This must be done even if you are logged on as an administrator.

## Disable Antivirus Tools

Some DISM commands may be blocked by antivirus or antimalware tools. Before servicing an image, disable antivirus or antimalware tools on the technician computer.

## Servicing an Image

The best way to service a Windows image is offline with the DISM tool. DISM can be used to install, uninstall, configure, and update drivers, features, and packages in Windows images and Windows Preinstallation Environment (WinPE) images without booting the image. For more information, see [DISM - Deployment Image Servicing and Management Technical Reference for Windows](#).

You can use the **/Commit-Image** option at any point during servicing to save the changes that you have made so far. You can recover a corrupted image more easily with the **/Cleanup-Image /RestoreHealth** option if you have committed your changes often.

You can mount and modify multiple images on a single computer. However, performance may slow down on some functions, such as **/Unmount-Image**, depending on the memory available on the computer. As a best practice, you should not mount more than 20 images at the same time.

**Note** If you have split a .wim file into smaller files for spanning across multiple media, you cannot mount the image for servicing.

## Changing International Settings

To change the international settings in Windows 10, Windows 8.1, Windows 8, Windows Server 2016 Technical Preview, Windows Server 2012 R2, Windows Server 2012, Windows 7, and Windows Server 2008 R2 images, you must use DISM. For more information, see [DISM Languages and International Servicing Command-Line Options](#).

## Use Log Files

DISM will log verbose information to %WINDIR%\Logs\Dism\Dism.log by default. You can also specify a name and location of your choice for the log file, and set the **/LogLevel** parameters so that only the information you are

interested in is logged. When an error occurs, the console will display the error code, error message, and the location of the log file.

**Important** If you specify a log path on a network share from a computer that is not joined to a domain, use net-use with domain credentials to set access permissions before you set the log path for the DISM log.

The log file will automatically be archived. The archived log file will be saved with .bak appended to the file name, and a new log file will be generated. Each time the log file is archived, the .bak file will be overwritten.

The log file gives you the history of the operations that have been performed, which can help you troubleshoot problems.

## Package Locations

Do not put a package that you intend to install directly at the root of a partition on a Windows installation.

## Storing Files on a Network Share

Although DISM supports network paths for images and packages, most operations will perform faster on files that are copied to the local hard-drive.

## Servicing a Windows Image from WinPE

You can service Windows images from WinPE. However, you must consider certain factors while planning your servicing strategy. Review the following requirements for servicing an image from WinPE.

### Booting WinPE from a Hard Drive

For better performance, you can allocate additional memory when you boot WinPE from a hard disk drive. You can also create temporary folders to store update files to accommodate large updates.

### Add Page-File Support to Your WinPE Image

Make sure you have sufficient memory to load and run your custom WinPE image. In addition to the image size, you should have at least 256 MB of available working memory. If you have limited memory, define a page file (Pagefile.sys) to improve memory management. For more information on implementing a page file, see [Wpeutil Command-Line Options](#).

### Create a Temporary Directory in Which to Store Update Files

You should use the **/ScratchDir** option with DISM to create a temporary directory on a different drive when you create or service a Windows image. A temporary directory is used for many DISM operations including capturing an image, installing language packs, installing updates, or installing or removing Windows features in a Windows image. Some files are expanded to this temporary directory before they are applied to a Windows image.

There must be sufficient space in the partition to accommodate large updates. The specific size of the free space that is required depends on the size of the updates that you intend to install. When adding a language pack, the scratch directory must have at least 1 GB of space for temporary files.

If you do not set a temporary directory path using the **/ScratchDir** option, WinPE creates a 32-MB temporary directory by default. You can allocate additional temporary storage to this default location using the DISM **/Set-ScratchSpace** option. Valid sizes include 32, 64, 128, 256, and 512 MB. This feature is available only offline and you cannot adjust this setting while a WinPE session is running. As a best practice, you should use the **/ScratchDir** option to instead specify a directory on another partition that has sufficient space to support any image management and servicing operations you perform.

After installation is complete, the contents of this directory are no longer needed and can be deleted. For more information, see [DISM Image Management Command-Line Options](#).

## Booting WinPE from a CD-ROM/DVD

Servicing a Windows image requires additional temporary storage space. For WinPE RAM disks, you might need additional RAM. In addition to the RAM requirements of your WinPE image, additional RAM is required to process updates. The amount of RAM that is required depends on the size of the updates that you intend to apply. Ensure that your computer has sufficient RAM.

## Scan for Corruption and Verify the Integrity of System Files

Before you deliver a computer to an end user, you should verify the integrity of Windows system files. You can use the **/Cleanup-Image** option to identify file corruption and perform repair operations on the image. For more information about the **/Cleanup-Image** option in DISM, see [DISM Operating System Package Servicing Command-Line Options](#).

You can also use System File Checker (Sfc.exe) on an online or offline reference image. System File Checker is released with all versions of Windows. System File Checker requires elevated permissions, and you must be an Administrator to run it. It scans all protected files to verify the file versions. To verify only the integrity of the Windows system files, run the **sfc.exe /verifyonly** option. For complete command-line syntax, at an elevated command prompt, type **sfc.exe /?**.

Running Sfc.exe can take a significant amount of time. The expected result is that there are no system integrity violations. However, if there are problems with Windows system files, you should investigate the issues. We do not recommend that you use the Sfc.exe scan options to automatically fix Windows system files.

## Improving Security for Windows Images

Your Windows images contain custom configuration data, custom applications, and other intellectual property. There are several ways to improve the security of your Windows images, both online and offline.

- **Restrict access to Windows images.** Depending on your environment, you can edit the access control lists (ACLs) or permissions on a file. Only approved accounts can have access to Windows images.
- **Update your Windows images with the latest fixes and software updates.** There are many ways you can service a Windows image. After servicing your Windows image, test the validity and stability of the computer.
- **During Windows installation, configure the computer to automatically download and install Windows updates.** This extends installation time, but ensures that the Windows image that you are installing contains the latest updates. For more information, see the `DynamicUpdate` setting in the Microsoft-Windows-Setup component in the [Unattended Windows Setup Reference](#).

## Related topics

[DISM - Deployment Image Servicing and Management Technical Reference for Windows](#)

[Understanding Servicing Strategies](#)

# DISM Supported Platforms

1/18/2019 • 3 minutes to read

The Windows 10 version of Deployment Image Servicing and Management (DISM) is available in Windows 10 for desktop editions (Home, Pro, Enterprise, and Education), Windows Server 2016, and Windows Preinstallation Environment (WinPE) for Windows 10.

To service Windows 10 images, you'll need the Windows 10 version of DISM, otherwise the image may become corrupted.

To use the Windows 10 version of DISM onto a previous version of Windows, install the Windows Assessment and Deployment Kit (ADK) [from this website](#), and install the **Deployment Tools**. Then, start the **Deployment and Imaging Tools Environment** to run DISM commands.

To use the Windows 10 version of DISM with a previous version of Windows PE, see [Install Windows 10 using a previous version of Windows PE](#).

Note, newer DISM features don't always work when servicing images of previous versions of Windows. To learn more, see the [DISM Reference](#).

## Supported Platforms

The host deployment environment is the operating system where DISM runs. The target image is the image that is being serviced.

HOST DEPLOYMENT ENVIRONMENT	TARGET IMAGE: WINDOWS 10 OR WINPE FOR WINDOWS 10	TARGET IMAGE: WINDOWS 8.1, WINDOWS SERVER 2016, WINDOWS SERVER 2012 R2, OR WINPE 5.0 (X86 OR X64)	TARGET IMAGE: WINDOWS 8, WINDOWS SERVER 2012, OR WINPE 4.0 (X86 OR X64)	TARGET IMAGE: WINDOWS 7, WINDOWS SERVER 2008 R2, OR WINPE 3.0 (X86 OR X64)
Windows 10 (x86 or x64)	Supported	Supported	Supported	Supported
Windows Server 2016 (x86 or x64)	Supported	Supported	Supported	Supported
Windows 8.1 (x86 or x64)	Supported, using the Windows 10 version of DISM	Supported	Supported	Supported
Windows Server 2012 R2 (x86 or x64)	Supported, using the Windows 10 version of DISM	Supported	Supported	Supported

<b>HOST DEPLOYMENT ENVIRONMENT</b>	<b>TARGET IMAGE: WINDOWS 10 OR WINPE FOR WINDOWS 10</b>	<b>TARGET IMAGE: WINDOWS 8.1, WINDOWS SERVER 2016, WINDOWS SERVER 2012 R2, OR WINPE 5.0 (X86 OR X64)</b>	<b>TARGET IMAGE: WINDOWS 8, WINDOWS SERVER 2012, OR WINPE 4.0 (X86 OR X64)</b>	<b>TARGET IMAGE: WINDOWS 7, WINDOWS SERVER 2008 R2, OR WINPE 3.0 (X86 OR X64)</b>
Windows 8(x86 or x64)	Supported, using the Windows 10 version of DISM	Supported, using the Windows 8.1 version of DISM or later	Supported	Supported
Windows Server 2012 (x86 or x64)	Supported, using the Windows 10 version of DISM	Supported, using the Windows 8.1 version of DISM or later	Supported	Supported
Windows 7 (x86 or x64)	Supported, using the Windows 10 version of DISM	Supported, using the Windows 8.1 version of DISM or later	Supported, using the Windows 8 version of DISM or later	Supported
Windows Server 2008 R2 (x86 or x64)	Supported, using the Windows 10 version of DISM	Supported, using the Windows 8.1 version of DISM or later	Supported, using the Windows 8 version of DISM or later	Supported
Windows Server 2008 SP2 (x86 or x64)	Not supported	Supported, using the Windows 8.1 version of DISM or later	Supported, using the Windows 8 version of DISM or later	Supported
WinPE for Windows 10 x86	Supported	Supported	Supported	Supported
WinPE for Windows 10 x64	Supported: X64 target image only	Supported: X64 target image only	Supported: X64 target image only	Supported: X64 target image only
WinPE 5.0 x86	Supported, using the Windows 10 version of DISM	Supported	Supported	Supported
WinPE 5.0 x64	Supported, using the Windows 10 version of DISM: X64 target image only	Supported: X64 target image only	Supported: X64 target image only	Supported: X64 target image only

HOST DEPLOYMENT ENVIRONMENT	TARGET IMAGE: WINDOWS 10 OR WINPE FOR WINDOWS 10	TARGET IMAGE: WINDOWS 8.1, WINDOWS SERVER 2016, WINDOWS SERVER 2012 R2, OR WINPE 5.0 (X86 OR X64)	TARGET IMAGE: WINDOWS 8, WINDOWS SERVER 2012, OR WINPE 4.0 (X86 OR X64)	TARGET IMAGE: WINDOWS 7, WINDOWS SERVER 2008 R2, OR WINPE 3.0 (X86 OR X64)
WinPE 4.0 x86	Supported, using the Windows 10 version of DISM	Supported, using the Windows 8.1 version of DISM or later	Supported	Supported
WinPE 4.0 x64	Supported, using the Windows 10 version of DISM: X64 target image only	Supported, using the Windows 8.1 version of DISM or later: X64 target image only	Supported: X64 target image only	Supported: X64 target image only
WinPE 3.0 x86	Supported, using the Windows 10 version of DISM	Supported, using the Windows 8.1 version of DISM or later	Supported, using the Windows 8 version of DISM or later	Supported
WinPE 3.0 x64	Supported, using the Windows 10 version of DISM: X64 target image only	Supported, using the Windows 8.1 version of DISM or later: X64 target image only	Supported, using the Windows 8 version of DISM or later: X64 target image only	Supported: X64 target image only

Resilient File System (REFS) is not supported.

## Related topics

[Install the Windows 10 Assessment and Deployment Kit \(ADK\)](#)

[DISM Reference \(Deployment Image Servicing and Management\)](#)

[Install Windows 10 using a previous version of Windows PE](#)

# Deployment Image Servicing and Management (DISM) API

1/18/2019 • 2 minutes to read

## Purpose

The Deployment Image Servicing and Management (DISM) API allows you to build customized solutions on the DISM platform. You can use the DISM API to install, uninstall, configure, and update Windows features, packages, and drivers in a Windows image.

## Developer Audience

The DISM API is designed for use by C/C++ programmers.

## Run-Time Requirements

DISM API can be used on any operating system supported by the Windows® Assessment and Deployment Kit (Windows ADK). For more information, see the [Windows ADK Technical Reference](#).

For more information, see [Using the DISM API](#).

## Additional Reference

The DISM platform also includes a command-line tool and Windows PowerShell cmdlets. For more information about the DISM tool, see [DISM Platform Technical Reference](#). For more information about DISM PowerShell cmdlets, see [DISM PowerShell Reference](#).

## In This Section

TOPIC	DESCRIPTION
<a href="#">Using the DISM API</a>	Review requirements, best practices, and other considerations for using the DISM API.
<a href="#">Creating a DISM Application</a>	Set up a development environment and learn about required functions.
<a href="#">DISM API Troubleshooting</a>	Use the DISM API log file to troubleshoot your custom application.
<a href="#">DISM API Reference</a>	Find a function or object defined by the DISM API.
<a href="#">DISM API Samples</a>	Look at sample usage of DISM API functions.

## Related topics

[DISM Platform Technical Reference](#)

[DISM PowerShell Reference](#)

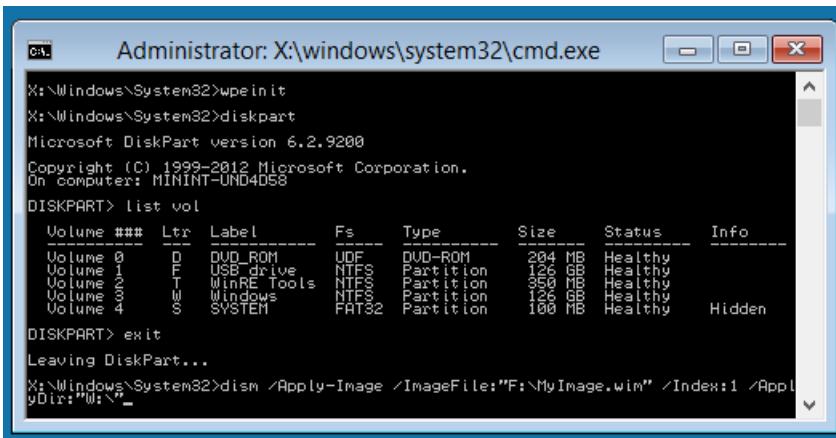


# Windows PE (WinPE)

1/18/2019 • 4 minutes to read

Windows PE (WinPE) for Windows 10 is a small operating system used to install, deploy, and repair Windows 10 for desktop editions (Home, Pro, Enterprise, and Education), Windows Server, and other Windows operating systems. From Windows PE, you can:

- Set up your hard drive before installing Windows.
- Install Windows by using apps or scripts from a network or a local drive.
- Capture and apply Windows images.
- Modify the Windows operating system while it's not running.
- Set up automatic recovery tools.
- Recover data from unbootable devices.
- Add your own custom shell or GUI to automate these kinds of tasks.



```
X:\Windows\System32>wpeinit
X:\Windows\System32>diskpart
Microsoft DiskPart version 6.2.9200
Copyright (C) 1999-2012 Microsoft Corporation.
On computer: MININT-UNDO58
DISKPART> list vol
 Volume ### Ltr Label Fs Type Size Status Info
 Volume 0 D DUD_ROM UDF DUD-ROM 284 MB Healthy
 Volume 1 F USB Drive NTFS Partition 126 GB Healthy
 Volume 2 T WinRE Tools NTFS Partition 350 MB Healthy
 Volume 3 W Windows NTFS Partition 126 GB Healthy
 Volume 4 S SYSTEM FAT32 Partition 100 MB Healthy Hidden
DISKPART> exit
Leaving DiskPart...
X:\Windows\System32>dism /Apply-Image /ImageFile:"F:\MyImage.wim" /Index:1 /Appl yDir:"W:\\"
```

## Where do I download it?

Starting with Windows 10, version 1809, WinPE is an [add-on to the Windows Assessment and Deployment Kit \(ADK\)](#). Previously it was included in the ADK. To get Windows PE, install the ADK, and then install the WinPE addon. For more info, see [WinPE: Create USB Bootable drive](#), [WinPE: Create a Boot CD, DVD, ISO, or VHD](#), or see the [Demo: Installing Windows PE on a USB Drive](#).

## Support for many Windows features

Windows PE runs the Windows command line environment, and supports these Windows features:

- **Batch files and scripts**, including support for Windows Script Host (WSH), and ActiveX Data Objects (ADO), and optional support for PowerShell.
- **Applications**, including Win32 application programming interfaces (APIs) and optional support for HTML Applications (HTA).
- **Drivers**, including a generic set of drivers that can run networking, graphics, and mass storage devices.
- **Image capturing and servicing**, including Deployment Image Servicing and Management (DISM).
- **Networking**, including connecting to file servers using TCP/IP and NetBIOS over TCP/IP via LAN.
- **Storage**, including NTFS, DiskPart, and BCDBoot.
- **Security tools**, including optional support for BitLocker and the Trusted Platform Module (TPM), Secure Boot, and other tools.

- **Hyper-V**, including VHD files, mouse integration, mass storage and network drivers that allow Windows PE to run in a hypervisor.

## Hardware requirements

Windows PE has the same requirements as Windows with these exceptions:

- No hard drive is required. You can run Windows PE entirely from memory.
- The base version requires only 512MB of memory. (If you add drivers, packages, or apps, you'll need more memory.)
- In order to boot Windows PE directly from memory (also known as RAM disk boot), a contiguous portion of physical memory (RAM) which can hold the entire Windows PE (WIM) image must be available. To optimize memory use, manufacturers should ensure that their firmware reserves memory locations either at the beginning or at the end of the physical memory address space.

The 32-bit version of Windows PE can boot 32-bit UEFI and BIOS PCs, and 64-bit BIOS PCs.

The 64-bit version of Windows PE can boot 64-bit UEFI and BIOS PCs.

## Limitations

Windows PE is not a general-purpose operating system. It may not be used for any purpose other than deployment and recovery. It should not be used as a thin client or an embedded operating system. There are other Microsoft products, such as Windows Embedded CE, which may be used for these purposes.

To prevent its use as a production operating system, Windows PE automatically stops running the shell and restarts after 72 hours of continuous use. This period is not configurable.

When Windows PE reboots, all changes are lost, including changes to drivers, drive letters, and the Windows PE registry. To make lasting changes, see [WinPE: Mount and Customize](#).

The default Windows PE installation uses the FAT32 file format, which poses its own limitations, including a maximum 4GB file size and maximum 32GB drive size. To learn more, see [WinPE: Use a single USB key for WinPE and a WIM file \(.wim\)](#).

Windows PE does not support any of the following:

- File server or Terminal Server use.
- Joining to a network domain.
- Connecting to an IPv4 network from Windows PE on an IPv6 network.
- Remote Desktop.
- .MSI installation files.
- Booting from a path that contains non-English characters.
- Running 64-bit apps on the 32-bit version of Windows PE.
- Adding bundled app packages through DISM (.appxbundle packages).

### TIP

In general, use the latest version of WinPE to deploy Windows. If you are using customized WinPE for Windows 10 images, you may prefer to continue using your existing Windows PE image and run the latest version of DISM from a network location. To learn more, see [Copy DISM to Another Computer](#).

### Notes on running Windows Setup in Windows PE:

- You can use the 32-bit versions of Windows PE and Windows Setup to install 64-bit versions of Windows. For more information, see [Windows Setup Supported Platforms and Cross-Platform Deployments](#).
- Although Windows PE supports dynamic disks, Windows Setup does not. If you install Windows to a dynamic disk created in Windows PE, the dynamic disks won't be available in Windows.
- For UEFI-based PCs that support both UEFI and legacy BIOS modes, Windows PE needs to be booted in the correct mode in order to correctly install Windows. For more info, see [WinPE: Boot in UEFI or legacy BIOS mode](#).

## See also

CONTENT TYPE	REFERENCES
<b>Product evaluation</b>	<a href="#">What's new in WinPE</a>
<b>Deployment</b>	<a href="#">Create a USB Bootable drive</a>   <a href="#">Demo: Installing Windows PE on a USB drive</a>   <a href="#">Create a boot CD, DVD, ISO, or VHD</a>   <a href="#">Install on a hard drive (Flat boot or Non-RAM)</a>   <a href="#">Boot in UEFI or legacy BIOS mode</a>   <a href="#">Boot to UEFI Mode or legacy BIOS mode</a>   <a href="#">Use a single USB key for WinPE and a WIM file (.wim)</a>
<b>Operations</b>	<a href="#">Mount and customize</a>   <a href="#">Add drivers</a>   <a href="#">Storage Area Network (SAN) policy</a>   <a href="#">Create apps</a>   <a href="#">Optimize and shrink the image</a>
<b>Troubleshooting</b>	<a href="#">Network drivers: Initializing and adding drivers</a>   <a href="#">Debug apps</a>
<b>Tools and settings</b>	<a href="#">Wpeinit and Startnet.cmd: Using WinPE Startup Scripts</a>   <a href="#">Identify drive letters with a script</a>   <a href="#">Add packages (Optional Components Reference)</a>
<b>Command-line options</b>	<a href="#">CopyPE</a>   <a href="#">Drvload</a>   <a href="#">Makewinpemedia</a>   <a href="#">Wpeutil</a>
<b>Technologies based on Windows PE</b>	<a href="#">Windows Setup</a>   <a href="#">Windows Recovery Environment</a>   <a href="#">Diagnostic and Recovery Toolset (DaRT)</a>

# What's New in Windows PE

1/24/2019 • 4 minutes to read

This topic describes the new and changed functionality of the Windows Preinstallation Environment (Windows PE/WinPE) and compares it with previous versions of Windows PE and MS-DOS.

## New and Changed Functionality

This table compares the features and functionality with those of previous versions of Windows PE:

FEATURE	WINDOWS PE FOR WINDOWS 10	WINDOWS PE 5.0	WINDOWS PE 4.0	WINDOWS PE 3.X	WINDOWS PE 2.X
Operating systems deployed	Windows 10, Windows 8.1, Windows Server 2012 R2, Windows 8, Windows Server 2012, Windows 7, or Windows Server 2008 R2.  Doesn't support: Windows Vista or Windows Server 2008.	Windows 8.1, Windows Server 2012 R2, Windows 8, Windows Server 2012, Windows 7, or Windows Server 2008 R2.  Doesn't support: Windows Vista or Windows Server 2008.	Windows 8, Windows Server 2012, Windows 7, Windows Server 2008 R2,  Windows Vista or Windows Server 2008.	Windows 7, Windows Server 2008 R2,  Windows Vista or Windows Server 2008.	Windows Vista or Windows Server 2008
Scripts used to deploy Windows PE	No change.	No change.	CopyPE updated for use with the Windows ADK.  MakeWinPE Media added to make creation of USB flash drives or ISO files easier.	CopyPE and Oscdimg tools included.	CopyPE and Oscdimg tools included.  Windows PE 2.1: Oscdimg tool updated to support larger images.

FEATURE	WINDOWS PE FOR WINDOWS 10	WINDOWS PE 5.0	WINDOWS PE 4.0	WINDOWS PE 3.X	WINDOWS PE 2.X
Scripting tools	No change.	.NET Framework optional component renamed to WinPE_NetFx.  PowerShell optional component renamed to WinPE_PowerShell.  Winpeshl.ini allows you to launch apps with command-line parameters in quotes. For more info, see <a href="#">Winpeshl.ini Reference: Launching an app when WinPE starts.</a>	.NET Framework 4.5 optional component added (WinPE_NetFx 4).  PowerShell 3.0 optional component added (WinPE_PowerShell3).	Command-line scripting tools included.	Command-line scripting tools included.

FEATURE	WINDOWS PE FOR WINDOWS 10	WINDOWS PE 5.0	WINDOWS PE 4.0	WINDOWS PE 3.X	WINDOWS PE 2.X
Image capturing and servicing tools	DISM supports Windows 10 and Windows Imaging and Configuration Designer (ICD) features.	DISM supports Windows 8.1 and Windows Server 2012 R2 images but doesn't support Windows Vista or Windows Server 2008 images. For more info, see <a href="#">DISM - Deployment Image Servicing and Management Technical Reference for Windows</a> .	Image capturing tools included with new <code>dism /Capture-image</code> and <code>dism /Apply-image</code> commands.  Doesn't support servicing Windows 8.1 or Windows Server 2012 R2 images.	DISM - Deployment Image Servicing and Management Technical Reference for Windows added. DISM is a command-line tool that you can use to customize a Windows or a Windows PE image.  The PEImg and Pkgmgr tools are not supported in Windows PE 3.0.  ImageX available as an optional application for capturing and applying images.	<b>PEImg</b> is used to service Windows PE images.  After you run <b>PEImg /prep</b> against the Windows PE 2.0 image, the image can't be modified.  ImageX is available as an optional application for capturing and applying images.  <b>Pkgmgr</b> is used to install, remove, or update Windows packages in offline images.  Doesn't support servicing Windows 8.1 or Windows Server 2012 R2 images.

FEATURE	WINDOWS PE FOR WINDOWS 10	WINDOWS PE 5.0	WINDOWS PE 4.0	WINDOWS PE 3.X	WINDOWS PE 2.X
Optimizing Windows PE	No change.	<p>The profiling feature is removed.</p> <p>The default amount of scratch space is 512 MB for PCs that have more than 1 GB of RAM.</p>	No change.	<p>Smaller default size.</p> <p>The Windows PE 3.0 default image contains only the minimum resources to support most deployment scenarios.</p> <p>You can add optional components by using Deployment Image Servicing and Management (DISM).</p> <p>The new <code>dism /apply-profiles</code> command allows you to further reduce the contents of a Windows PE 3.0 image to only those files necessary to support a given set of apps.</p>	<p>Windows PE 2.1: Supports booting directly from the hard disk, not into RAM disk.</p> <p>Windows PE 2.1: Writable RAM drive: when booting from read-only media, Windows PE automatically creates a writable RAM disk (drive X) and allocates 32 megabytes (MB) of the RAM disk for general-purpose storage. You can customize the size, in megabytes, by using <b>PEImg /scratchsize</b>. Valid values are 32, 64, 128, 256, and 512.</p>
File management	No change.	No change.	File Management optional component added for discovering and restoring deleted files from unencrypted volumes.	Windows PE 3.1: base image contains improvements that are related to 4k/512e drive support.	No 4k/512e drive support.

FEATURE	WINDOWS PE FOR WINDOWS 10	WINDOWS PE 5.0	WINDOWS PE 4.0	WINDOWS PE 3.X	WINDOWS PE 2.X
Memory	No change.	<p>Maximum supported:</p> <ul style="list-style-type: none"> <li>● x86: 64 GB</li> <li>● x64: 4 TB</li> </ul>	No change.	No change.	<p>Maximum supported:</p> <ul style="list-style-type: none"> <li>● x86: 4 GB</li> <li>● x64: 128 GB</li> </ul>
Virtualization	No change.	No change.	No change.	<p>Windows PE 3.0 includes all Hyper-V drivers except display drivers. This enables Windows PE to run in Hypervisor. Supported features include mass storage, mouse integration, and network adapters.</p>	Not supported.
Networking	No change.	No change.	<p>Optional Remote Network Driver Interface Specification (RNDIS) feature added for enabling network devices that implement the RNDIS specification over USB.</p>	<p>The Windows PE 3.1 base image contains RNDIS binaries. Windows PE 3.0: <a href="#">Hotfix</a> available for 802.1X (LAN) support.</p> <p>Windows PE 3.1 includes 802.1X binaries as an optional component. The file name of this package is WinPE-Dot3Svc.cab.</p>	<p>Supports IPv4 and IPv6. Doesn't support other protocols, like Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX).</p>

FEATURE	WINDOWS PE FOR WINDOWS 10	WINDOWS PE 5.0	WINDOWS PE 4.0	WINDOWS PE 3.X	WINDOWS PE 2.X
Recovery	No change.	No change.	WinRE Configuration utility added (winrecfg.exe) to support configuring Windows RE in an offline operating system.	No change.	Supports Windows Recovery Environment (Windows RE).
Security	No change.	No change.	Secure Startup optional component added for provisioning and managing BitLocker and the Trusted Platform Module.	No change.	Supports BitLocker and Trusted Platform Module.
Architectures	No change.	No change.	Supports x86, x64, and ARM-based PCs.	No change.	Supports x86, x64, and Itanium-based PCs.

To see which version of Windows PE you're running, type `regedit` and locate this registry key:

**HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\WinPE**.

## Comparison with MS-DOS

Windows PE is similar to MS-DOS. It also includes support for the following features:

- The NTFS 5.x file system, including dynamic volume creation and management.
- TCP/IP networking and file sharing (client only).
- 32-bit or 64-bit Windows device drivers.
- A subset of the Windows application programming interface (API).
- CD, DVD, and USB flash drives.
- Windows Deployment Services server.
- Image management and servicing (DISM).
- Hyper-V drivers (all drivers except for display drivers). This enables Windows PE to run in a hypervisor. Supported features include mass storage, mouse integration, and network adapters.
- Optional support for PowerShell, Windows Management Instrumentation (WMI), Windows Data Access Components (Windows DAC), and HTML Applications (HTAs).

## Where is WinPE.wim?

In Windows 7, the main Windows PE boot file was renamed from winpe.wim to boot.wim. This file is in Windows PE in the \sources folder. It can be modified in the same way as WinPE.wim.

## Related topics

[WinPE for Windows 10](#)

# WinPE Optional Components (OC) Reference

1/25/2019 • 12 minutes to read

Add feature packages, also known as optional components, to Windows PE (WinPE).

## Where to get WinPE Optional Components

WinPE optional components become available when you install the **Windows Preinstallation Environment** with the **Windows Assessment and Deployment Kit** (ADK). Starting with Windows 10, version 1809, WinPE is an addon to the ADK that [you download](#) and install after you install the ADK.

Optional components are available in 32- and 64-bit architectures. The OCs you add to your WinPE image must be from the same ADK build and have the same architecture as your WinPE image. You can find WinPE optional components in the following locations after you install the ADK:

- **64-bit**

```
C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE\_OCs\
```

- **32-bit**

```
C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\x86\WinPE\_OCs\
```

## How to add Optional Components

WinPE OCs are .cab packages that can be added to an offline Windows image.

**IMPORTANT**

Some optional components have prerequisites that must be installed in order. The [table below](#) includes information about prerequisites.

Before adding OCs to a WinPE image, you need a WinPE image (boot.wim). See [WinPE: Create bootable media](#) to learn how to make a set of WinPE working files that include a WinPE image. You'll find boot.wim at [sources\boot.wim](#) in your working files.

Once you have a WinPE image to work with, you can add packages to it with `DISM /add-package`. To learn more about using DISM to add packages to a Windows image, see [Add or remove packages offline using DISM](#).

Many WinPE optional components come split into two parts, a language-neutral package, and a set of language-specific packages. When installing an OC that has language-specific packages, you need to first add the language-neutral OC, and then add at least one of its associated language-specific packages. The language-specific and language-neutral resources must be of the same version. You can find OCs in the following folders:

- **Language-neutral -**

```
...\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\
```

- **Language-specific -**

```
...\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\xx-xx\
```

## WinPE Optional Components

Area/Optional Component Name	Description
Database/WinPE-MDAC	<p>WinPE-MDAC supports Microsoft Open Database Connectivity (ODBC), OLE DB, and Microsoft ActiveX Data Objects (ADO). This set of technologies provides access to various data sources, such as Microsoft SQL Server. For example, this access enables queries to Microsoft SQL Server installations that contain ADO objects. You can build a dynamic answer file from unique system information. Similarly, you can build data-driven client or server applications that integrate information from a variety of data sources, both relational (SQL Server) and non-relational.</p>
File management/WinPE-FMAPI	<p>WinPE-FMAPI provides access to the Windows PE File Management API (FMAPI) for discovering and restoring deleted files from unencrypted volumes. The FMAPI also provides the ability to use a password or recovery key file for the discovery and recovery of deleted files from Windows BitLocker Drive Encryption encrypted volumes.</p>
Fonts/WinPE-Fonts-Legacy	<p>WinPE-Fonts-Legacy contains 32 font files for various languages/writing scripts. Some of these fonts are no longer used as UI fonts. For example, scripts such as Bangla, Devanagari, Gujarati, Gurmukhi, Kannada, Malayalam, Odia, Tamil, Telugu, and Sinhalese were covered by Mangal, Latha, Vrinda, Gautami, Kalinga, artika, Raavi, Shruti, and Tunga, but in Windows 8, they were all unified under Nirmala UI, a single, pan-Indian font. The following list shows the fonts and languages included in this optional component:</p> <ul style="list-style-type: none"> <li>• estre.ttf Estrangelo Edessa (Syriac)</li> <li>• mvboli.ttf MV Boli (Thaana)</li> <li>• KhmerUI.ttf Khmer UI (Khmer UI)</li> <li>• KhmerUIB.ttf Khmer UI Bold (Khmer UI)</li> <li>• Laoui.ttf Lao UI (Lao)</li> <li>• Laouib.ttf Lao UI Bold (Lao)</li> <li>• daunpenh.ttf DaunPenh (Khmer)</li> <li>• moolbor.ttf MoolBoran (Khmer)</li> <li>• dokchamp.ttf DokChampa (Lao)</li> <li>• Himalaya.ttf Microsoft Himalaya (Tibetan)</li> <li>• monbaiti.ttf Mongolian Baiti (Mongolian)</li> <li>• MSYI.ttf Microsoft Yi Baiti (Yi Syllables)</li> <li>• nyala.ttf Nyala (Ethiopic)</li> <li>• sylfaen.ttf Sylfaen (Armenian &amp; Georgian)</li> <li>• euphemia.ttf Euphemia (Unified Canadian Aboriginal Syllabics)</li> <li>• plantc.ttf Plantagenet Cherokee (Cherokee)</li> </ul>

Area/Optional Component Name	Description
Fonts/WinPE-Font Support-JA-JP	<p>WinPE-Font Support-JA-JP contains two Japanese font families that are packaged as TrueType Collection (TTC) files. MS Gothic is the Windows Japanese user interface font in versions of Windows before Windows Vista. MS Gothic contains a large character set and embedded bitmaps to ensure legible rendering at small sizes. Meiryo, a font that was introduced in Windows Vista, is designed specifically for use in a Microsoft ClearType® rendering environment. Meiryo does not include embedded bitmaps. Instead, Meiryo relies on hinting instructions to produce legible characters at small sizes. In addition, the module contains two Japanese bitmap fonts, App932.fon and Vga932.fon. The module also contains a bitmap-only TrueType font, Jpn_font.ttf. This font is used on boot screens.</p>
Fonts/WinPE-Font Support-KO-KR	<p>WinPE-Font Support-KO-KR contains three core Korean font families: Gulim, Batang and Malgun Gothic. Gulim is the legacy UI font and, as a TTC file, contains Gulim, GulimChe, Dotum and DotumChe. Batang is the legacy text font and is also a TTC file, containing Batang, BatangChe, GungSuh and GungSuhChe. Malgun Gothic, a font that was introduced in Windows Vista, is designed specifically for use in a ClearType rendering environment. Malgun Gothic does not include embedded bitmaps and instead relies on hinting instructions to produce legible characters at small sizes.</p>
Fonts/WinPE-Font Support-ZH-CN	<p>WinPE-Font Support-ZH-CN contains two Chinese font families that are packaged as TTC files. Simsun is the Simplified Chinese user interface font in Windows versions before Windows Vista. Simsun contains embedded bitmaps to ensure legible rendering at small sizes. The other TTC font is MingLiu. MingLiu has embedded bitmaps and provides support for the Hong Kong Supplementary Character Set (HKSCS). YaHei, a font that was introduced in Windows Vista, is designed specifically for use in a ClearType rendering environment. YaHei does not include embedded bitmaps. YaHei relies on hinting instructions to produce legible characters at small sizes. In addition, the module contains one bitmap-only TrueType font, Chs_boot.ttf. This font is used on boot screens.</p>
Fonts/WinPE-Font Support-ZH-HK and WinPE-Font Support-ZH-TW	<p>The Hong Kong and Taiwan optional components contain two Chinese font families that are packaged as TTC files. Simsun is the Simplified Chinese user interface font in Windows versions before Windows Vista. Simsun contains embedded bitmaps to ensure legible rendering at small sizes. MingLiu has embedded bitmaps and provides support for the HKSCS. JhengHei, a font that was introduced in Windows Vista, is designed specifically for use in a ClearType rendering environment. JhengHei does not include embedded bitmaps. JhengHei relies on hinting instructions to produce legible characters at small sizes. In addition, the module contains one bitmap-only TrueType font, Cht_boot.ttf. This font is used on boot screens.</p>

Area/Optional Component Name	Description
HTML/WinPE-HTA	WinPE-HTA provides HTML Application (HTA) support to create GUI applications through the Windows Internet Explorer script engine and HTML services. These applications are trusted and display only the menus, icons, toolbars, and title information that you create.
Input/WinPE-GamingPeripherals	WinPE-GamingPeripherals adds support for Xbox wireless controllers in WinPE.
Microsoft .NET/WinPE-NetFX	<p>WinPE-NetFX contains a subset of the .NET Framework 4.5 that is designed for client applications.</p> <p>Not all Windows binaries are present in Windows PE, and therefore not all Windows APIs are present or usable.</p> <p>Due to the limited API set, the following .NET Framework features have no or reduced functionality in Windows PE:</p> <ul style="list-style-type: none"> <li>• Windows Presentation Foundation (WPF) (not supported)</li> <li>• Windows Runtime</li> <li>• .NET Framework Fusion APIs</li> <li>• Windows Control Library event logging</li> <li>• .NET Framework COM Interoperability</li> <li>• .NET Framework Cryptography Model</li> </ul> <p><b>Dependencies:</b></p> <ul style="list-style-type: none"> <li>• Install <b>WinPE-WMI</b> before you install <b>WinPE-NetFX</b>.</li> </ul>
Network/WinPE-Dot3Svc	Adds support for the IEEE 802.X authentication protocol on wired networks. For more info, see <a href="#">WinPE Network Drivers: Initializing and adding drivers</a> .
Network/WinPE-PPPoE	WinPE-PPPoE enables you to use Point-to-Point Protocol over Ethernet (PPPoE) to create, connect, disconnect, and delete PPPoE connections from Windows PE. PPPoE is a network protocol for encapsulating Point-to-Point Protocol (PPP) frames inside Ethernet frames. PPPoE enables Windows users to remotely connect their computers to the web. By using PPPoE, users can virtually dial from one computer to another over an Ethernet network, to establish a point-to-point connection between the computers. The computers can use this point-to-point connection to transport data packets.
Network/WinPE-RNDIS	WinPE-RNDIS contains Remote Network Driver Interface Specification (Remote NDIS) support. WinPE-RNDIS enables network support for devices that implement the Remote NDIS specification over USB. Remote NDIS defines a bus-independent message set and a description of how this message set operates over various I/O buses. Therefore, hardware vendors do not have to write an NDIS miniport device driver. Because this Remote NDIS interface is standardized, one set of host drivers can support any number of bus-attached networking devices.

Area/Optional Component Name	Description
Network/WinPE-WDS-Tools	<p>WinPE-WDS-Tools includes APIs to enable the Image Capture tool and a multicast scenario that involves a custom Windows Deployment Services client. It must be installed if you intend to run the Windows Deployment Services client on a custom Windows PE image.</p>
Network/WinPE-WiFi-Package	<p>WinPE-WiFi-Package is used by Windows Recovery Environment (Windows RE) for built-in recovery functions. This package is included in the base winre.wim file.</p> <p><b>Note:</b> Windows PE and Windows RE don't support general wireless networking functions.</p>
Windows PowerShell/WinPE-PlatformID	<p>WinPE-PlatformID contains the Windows PowerShell cmdlets to retrieve the Platform Identifier of the physical machine.</p> <p><b>Dependencies:</b> Install <b>WinPE-WMI</b> and <b>WinPE-SecureStartup</b> before you install <b>WinPE-PlatformID</b>. To use the Windows PowerShell cmdlet to retrieve the Platform Identifier, you will need install <b>WinPE-PowerShell</b> package.</p>
Windows PowerShell/WinPE-PowerShell	<p>WinPE-PowerShell contains Windows PowerShell-based diagnostics that simplify using Windows Management Instrumentation (WMI) to query the hardware during manufacturing. You can create Windows PowerShell-based deployment and administrative Windows PE-based tools. In addition to deployment, you can use Windows PowerShell for recovery scenarios. Customers can boot in Windows RE and then use Windows PowerShell scripts to resolve issues. Customers are not limited to the toolsets that run in Windows PE. Similarly, you can build scripted offline solutions to recover some computers from no-boot scenarios.</p> <p>WinPE-PowerShell has the following known limitations:</p> <ul style="list-style-type: none"> <li>• Windows PowerShell remoting is not supported. Any cmdlets that have remoting functionality will return an error.</li> <li>• The Windows PowerShell Integrated Scripting Environment (ISE) is not supported.</li> <li>• Windows PowerShell 2.0 is not supported.</li> </ul> <p><b>Dependencies:</b> Install <b>WinPE-WMI</b> &gt; <b>WinPE-NetFX</b> &gt; <b>WinPE-Scripting</b> before you install <b>WinPE-PowerShell</b>.</p>
Windows PowerShell/WinPE-DismCmdlets	<p>WinPE-DismCmdlets contains the DISM PowerShell module, which includes cmdlets used for managing and servicing Windows images.</p> <p>For more info, see <a href="#">Deployment Imaging Servicing Management (DISM) Cmdlets in Windows PowerShell</a>.</p> <p><b>Dependencies:</b> Install <b>WinPE-WMI</b> &gt; <b>WinPE-NetFX</b> &gt; <b>WinPE-Scripting</b> &gt; <b>WinPE-PowerShell</b> before you install <b>WinPE-DismCmdlets</b>.</p>

Area/Optional Component Name	Description
Windows PowerShell/WinPE-SecureBootCmdlets	<p>WinPE-SecureBootCmdlets contains the PowerShell cmdlets for managing the UEFI (Unified Extensible Firmware Interface) environment variables for Secure Boot.</p> <p><b>Dependencies:</b> Install <b>WinPE-WMI</b> &gt; <b>WinPE-NetFX</b> &gt; <b>WinPE-Scripting</b> &gt; <b>WinPE-PowerShell</b> before you install <b>WinPE-SecureBootCmdlets</b>.</p>
Windows PowerShell/WinPE-StorageWMI	<p>WinPE-StorageWMI contains PowerShell cmdlets for storage management. These cmdlets use the Windows Storage Management API (SMAPI) to manage local storage, such as disk, partition, and volume objects. Or, these cmdlets use the Windows SMAPI together with array storage management by using a storage management provider. WinPE-StorageWMI also contains Internet SCSI (iSCSI) Initiator cmdlets for connecting a host computer or server to virtual disks on external iSCSI-based storage arrays through an Ethernet network adapter or iSCSI Host Bus Adapter (HBA).</p> <p><b>Dependencies:</b> Install <b>WinPE-WMI</b> &gt; <b>WinPE-NetFX</b> &gt; <b>WinPE-Scripting</b> &gt; <b>WinPE-PowerShell</b> before you install <b>WinPE-StorageWMI</b>.</p>
Recovery/WinPE-Rejuv	<p>WinPE-Rejuv is used by Windows Recovery Environment (Windows RE). This package is included in the base winre.wim file.</p>
Recovery/WinPE-SRT	<p>WinPE-SRT is used by Windows RE. This package is included in the base winre.wim file.</p>
Recovery/WinPE-WinReCfg	<p>WinPE-WinReCfg contains the Winrecfg.exe tool, and it enables the following scenarios:</p> <ul style="list-style-type: none"> <li>• Boot from x86-based Windows PE to configure Windows RE settings on an offline x64-based operating system image.</li> <li>• Boot from x64-based Windows PE to configure Windows RE settings on an offline x86-based operating system image.</li> </ul>
Scripting/WinPE-Scripting	<p>WinPE-Scripting contains a multiple-language scripting environment that is ideal for automating system administration tasks, such as batch file processing. Scripts that run in the Windows Script Host (WSH) environment can call WSH objects and other COM-based technologies that support Automation, such as WMI, to manage the Windows subsystems that are central to many system administration tasks.</p> <p><b>Dependencies:</b> Install WinPE-Scripting to make sure that full scripting functionality is available when you are using WinPE-NetFX and WinPE-HTA. The installation order is irrelevant.</p>

Area/Optional Component Name	Description
Scripting/WinPE-WMI	<p>WinPE-WMI contains a subset of the Windows Management Instrumentation (WMI) providers that enable minimal system diagnostics. WMI is the infrastructure for management data and operations on Windows-based operating systems. You can write WMI scripts or applications to automate administrative tasks on remote computers. Additionally, WMI supplies management data to other parts of the operating system and products.</p>
Setup/Winpe-LegacySetup	<p>Winpe-LegacySetup contains all Setup files from the \Sources folder on the Windows media. Add this optional component when you service Setup or the \Sources folder on the Windows media. You must add this optional component together with the optional component for the Setup feature. To add a new Boot.wim file to the media, add the parent WinPE-Setup, either of the children (WinPE-Setup-Client or WinPE-Setup-Server), and Media optional components. Media Setup is required to support Windows Server 2008 R2 installation.</p>
Setup/WinPE-Setup	<p>WinPE-Setup is the parent of WinPE-Setup-Client and WinPE-Setup-Server. It contains all Setup files from the \Sources folder that are common to the client and the server.</p>
Setup/WinPE-Setup-Client	<p>WinPE-Setup-Client contains the client branding files for the parent WinPE-Setup optional component.</p> <p><b>Dependencies:</b> Install <b>WinPE-Setup</b> before you install <b>WinPE-Setup-Client</b>.</p>
Setup/WinPE-Setup-Server	<p>WinPE-Setup-Server includes the server branding files for the parent WinPE-Setup optional component.</p> <p><b>Dependencies:</b> Install <b>WinPE-Setup</b> before you install <b>WinPE-Setup-Server</b>.</p>
Startup/WinPE-SecureStartup	<p>WinPE-SecureStartup enables provisioning and management of BitLocker and the Trusted Platform Module (TPM). It includes BitLocker command-line tools, BitLocker WMI management libraries, a TPM driver, TPM Base Services (TBS), the Win32_TPM class, the BitLocker Unlock Wizard, and BitLocker UI libraries. The TPM driver provides better support for both BitLocker and the TPM in this preboot environment.</p> <p><b>Dependencies:</b> Install <b>WinPE-WMI</b> before you install <b>WinPE-SecureStartup</b>.</p>
Storage/WinPE-EnhancedStorage	<p>WinPE-EnhancedStorage enables Windows to discover additional functionality for storage devices, such as encrypted drives, and implementations that combine Trusted Computing Group (TCG) and IEEE 1667 ("Standard Protocol for Authentication in Host Attachments of Transient Storage Devices") specifications. This optional component enables Windows to manage these storage devices natively by using BitLocker.</p>

# Windows RE optional components

The default Windows RE image contains the following built-in optional components: | Included Optional Component | | ---- | | WinPE-EnhancedStorage | | WinPE-Rejuv | | WinPE-Scripting | | WinPE-SecureStartup | | WinPE-Setup | | WinPE-SRT | | WinPE-WDS-Tools | | WinPE-WMI |

## Add more languages to images that include optional components

Note: To add language packs using Windows PE, you may need to add pagefile support to Windows PE. For more information, see [Deployment Image Servicing and Management \(DISM\) Best Practices](#).

1. On a mounted image, list the optional components in the Windows PE image:

```
Dism /Get-Packages /Image:"C:\WinPE_amd64\mount"
```

2. Review the resulting list of packages, and add the corresponding language packs for each package in the image, including the base Windows PE language pack.

```
Dism /Add-Package /Image:"C:\WinPE_amd64\mount" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\lp.cab"

Dism /Add-Package /Image:"C:\WinPE_amd64\mount" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-HTA_fr-fr.cab"
```

where ... WinPE\_OCs\fr-fr\lp.cab represents the base Windows PE language pack.

3. If you're adding language packs for Japan, Korea, or China, add the font packages for these languages. Here's an example for Japan:

```
Dism /Add-Package /Image:"C:\WinPE_amd64\mount" /PackagePath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-Font Support-JA-JP.cab"
```

4. Verify that the language packs are part of the image:

```
Dism /Get-Packages /Image:"C:\WinPE_amd64\mount"
```

Review the resulting list of packages and verify that for each optional component, including the base Windows PE image, that there is an associated language pack.

5. Change the regional settings to the language you'd like to use:

```
Dism /Set-AllIntl:en-US /Image:"C:\WinPE_amd64\mount"
```

To switch languages while in Windows PE, use `wpeutil setmuilanguage`.

## Related topics

[WinPE: Optimize and shrink the image](#)

[WinPE for Windows 10](#)

[WinPE: Mount and Customize](#)



# WinPE: Mount and Customize

1/24/2019 • 6 minutes to read

WinPE ships as a .wim file. Mounting and customizing a WinPE image is the same process as any other Windows image. WinPE also has some customizations that are specific to it. This topic covers the common ways to customize a WinPE image.

## Common customizations:

- [Device drivers \(.inf files\)](#). You can customize device drivers, such as drivers that support network cards or storage devices.
- [Packages \(.cab files, also known as WinPE optional components\)](#) Add languages, hotfixes, or support for features like PowerShell and the HTML Application Language (HTA).
- [Languages](#). To run WinPE in multiple languages, add the packages (optional components) for those languages.
- Add files and folders. These can be added directly to the WinPE image.
- [DISM: Use a newer version](#). When new versions of Windows require features from the latest version of DISM, you can add DISM directly into WinPE.
- [Startup scripts](#). Examples include setting up a network connection, or adding a custom application, such as diagnostic software.
- [Apps](#). Note, WinPE only supports legacy apps.
- [Temporary storage \(scratch space\)](#). If your application requires temporary file storage, you can reserve extra memory space in RAM.
- [Background image](#)
- [Power scheme](#)
- [WinPE settings](#)
- [Windows updates](#)

## Get the Windows Assessment and Deployment Kit with Windows PE tools

- Install the [Windows Assessment and Deployment Kit \(Windows ADK\) Technical Reference](#), with **Windows Preinstallation Environment**. If you're using the ADK for Windows 10, version 1809, you'll have to download and install the WinPE [addon](#) after you install the ADK. Previous versions of the ADK include **Windows Preinstallation Environment** in the ADK installer.

## Create a set of either 32-bit or 64-bit Windows PE files

Before you can customize WinPE, you need to have a WinPE image to work with. If you need to get a WinPE image, see [WinPE: Create USB bootable drive](#) to learn how.

## Mount the Windows PE boot image

- Use DISM to mount the WinPE image into a temporary location on your technician PC:

```
Dism /Mount-Image /ImageFile:"C:\WinPE_amd64\media\sources\boot.wim" /index:1  
/MountDir:"C:\WinPE_amd64\mount"
```

# Add customizations

## Add device drivers (.inf files)

- Use `DISM /add-driver` to add a device driver to your WinPE image.

```
Dism /Add-Driver /Image:"C:\WinPE_amd64\mount" /Driver:"C:\SampleDriver\driver.inf"
```

You can add multiple drivers to an image by using one command, but it's often easier to troubleshoot problems if you add each driver package individually.

To learn more about drivers, see [Add device drivers \(.inf files\)](#). To see all available DISM driver servicing options, see [DISM driver servicing command-line options](#).

## Add packages/languages/optional components/.cab files

- WinPE has packages that you can add with DISM to enable additional features and languages. Use `DISM /add-package` to add optional components to your image. When you add a WinPE optional component, make sure that you add both the optional component and its associated language packs.

```
Dism /Add-Package /Image:"C:\WinPE_amd64\mount" /PackagePath:"C:\Program Files\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-HTA.cab"

Dism /Add-Package /Image:"C:\WinPE_amd64\mount" /PackagePath:"C:\Program Files\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-HTA_en-us.cab"
```

To learn more about available optional components and languages, see [WinPE: Add packages \(Optional Components Reference\)](#).

## Add files and folders

- Copy files and folders into the C:\WinPE\_amd64\mount folder. These files will show up in the X:\ folder in WinPE.

Don't add too many files, as these will slow down WinPE and can fill up the available memory in the default RAMDisk environment.

## Add a startup script

- Modify Startnet.cmd to include your customized commands. This file is located in your mounted image at `C:\WinPE_amd64\mount\Windows\System32\Startnet.cmd`.

You can also call other batch files or command line scripts from this file.

For Plug and Play or networking support, make sure that you include a call to `wpeinit` in your customized Startnet.cmd script. For more info, see [Wpeinit and Startnet.cmd: Using WinPE Startup Scripts](#).

## Add an app

1. Create an app directory inside the mounted WinPE image.

```
md "C:\WinPE_amd64\mount\windows\<MyApp>"
```

2. Copy the necessary app files to the local WinPE directory.

```
Xcopy C:\<MyApp> "C:\WinPE_amd64\mount\windows\<MyApp>"
```

3. Test the app later by booting WinPE and running the application from the X: directory.

```
X:\Windows\System32> X:\Windows\<MyApp>
```

If your app requires temporary storage, or if WinPE becomes unresponsive when it runs an app, you may need to increase the amount of temporary storage (scratch space) allocated to WinPE.

4. To automatically launch a shell or application that runs when WinPE starts, add the path location to the Winpeshl.ini file. For more info, see [Winpeshl.ini Reference: Launching an app when WinPE starts](#).

### Add temporary storage (scratch space)

- WinPE reserves memory on the X: drive to unpack the WinPE files, plus additional temporary file storage, known as scratch space, that can be used by your applications. By default, this is 512MB for PCs with more than 1GB of RAM, otherwise the default is 32MB. Valid values are 32, 64, 128, 256, or 512.

```
Dism /Set-ScratchSpace:256 /Image:"C:\WinPE_amd64\mount"
```

### Replace the background image

If you've got multiple versions of WinPE, you can set the background image so you can instantly tell which version of WinPE is running.

Change the security permissions of the WinPE background image file (`\windows\system32\winpe.jpg`). This allows you to modify or delete the file.

1. In Windows Explorer, navigate to `C:\WinPE_amd64\mount\windows\system32`.
2. Right-click the `c:\WinPE_amd64\mount\windows\system32\winpe.jpg` file, and select **Properties > Security tab > Advanced**.
3. Next to Owner, select **Change**. Change the owner to **Administrators**.
4. Apply the changes, and exit the Properties window to save changes.
5. Right-click the `c:\WinPE_amd64\mount\windows\system32\winpe.jpg` file, and select **Properties > Security tab > Advanced**.
6. Modify the permissions for **Administrators** to allow full access.
7. Apply the changes, and exit the Properties window to save changes.
8. Replace the `winpe.jpg` file with your own image file.

### Set the power scheme to high performance

Note: Using the high performance power scheme can make the device run hotter than usual.

1. In Notepad, edit the file: `C:\WinPE_amd64\mount\windows\system32\startnet.cmd`, adding a command to set the power scheme to High Performance.

```
wpeinit  
powercfg /s 8c5e7fda-e8bf-4a96-9a85-a6e23a8c635c
```

### Add answer file settings

- Some WinPE settings can be managed by using an answer file, such as firewall, network, and display settings. Create an answer file, name it unattend.xml, and add it to the root of the WinPE media to process these settings. For more information, see [Wpeinit and Startnet.cmd: Using WinPE Startup Scripts](#).

### Add updates to WinPE (if needed)

You can apply updates to your WinPE image, but you'll only need to for certain situations.

If you've been instructed to apply an update to your WinPE image, you'll have to first download the latest update for your WinPE version from the [Microsoft update catalog](#). Updates for WinPE are included in updates for the matching version of Windows 10. You can find information about the latest available updates for Windows 10 at [Windows 10 update history](#).

1. Download the latest update.
2. Apply the update to your mounted WinPE image.

```
dism /Add-Package /Image:"C:\WinPE_amd64\mount" /PackagePath:"E:\windows10.0-kbxxxxx.msu"
```

Where Windows10.0-kbxxxxx.msu is the name of the update file

3. Lock in the update:

```
dism /cleanup-image /image:C:\WinPE_amd64\mount\windows /startcomponentcleanup /resetbase  
/scratchdir:C:\temp
```

## Unmount the Windows PE image and create media

1. Unmount the WinPE image, committing changes.

```
Dism /Unmount-Image /MountDir:"C:\WinPE_amd64\mount" /commit
```

2. Create bootable media, such as a USB flash drive.

```
MakeWinPEMedia /UFD C:\WinPE_amd64 F:
```

3. Boot the media. WinPE starts automatically. After the WinPE window appears, the wpeinit command runs automatically. This may take a few minutes. Verify your customizations.

## Troubleshooting

- WinPE won't boot? See the troubleshooting tips at the end of the topic: [WinPE: Create USB Bootable drive](#)
- For tips on connecting to a network, see [WinPE Network Drivers: Initializing and adding drivers](#).
- If the WinPE image becomes unserviceable, you may need to clean up the images before you can mount the image again. For information, see [Repair a Windows Image](#).

### To delete a working directory:

In some cases, you may not be able to recover the mounted image. DISM protects you from accidentally deleting the working directory, so you may have to try the following steps to get access to delete the mounted directory. Try each of the following steps:

1. Try remounting the image:

```
dism /Remount-Image /MountDir:C:\mount
```

2. Try unmounting the image, discarding the changes:

```
dism /Unmount-Image /MountDir:C:\mount /discard
```

3. Try cleaning up the resources associated with the mounted image:

```
dism /Cleanup-Mountpoints
```

## Related topics

[WinPE: Optimize and shrink the image](#)

[WinPE for Windows 10](#)

[WinPE: Create USB Bootable drive](#)

[WinPE: Create a Boot CD, DVD, ISO, or VHD](#)

[WinPE: Install on a Hard Drive \(Flat Boot or Non-RAM\)](#)

[WinPE: Boot in UEFI or legacy BIOS mode](#)

[WinPE: Add packages \(Optional Components Reference\)](#)

# WinPE: Adding Windows PowerShell support to Windows PE

1/18/2019 • 2 minutes to read

The following sample script creates a version of Windows PE with Windows PowerShell and its DISM and Storage cmdlets, which can be used to help automate Windows deployment.

## Prepare a local copy of the Windows PE files

1. Install the [Windows Assessment and Deployment Kit \(ADK\)](#), adding the **Deployment Tools** and **Windows Preinstallation Environment** features. If you're using installing the ADK for Windows 10, version 1809, [Windows PE is a separate add-on](#) that you'll need to download and install after you install the ADK. Previous versions of the ADK include the **Windows Preinstallation Environment** in the ADK installer.
2. Start the **Deployment and Imaging Tools Environment** as an **administrator**.
3. Create a working copy of the Windows PE files. Specify either x86, amd64, or arm:

```
copype amd64 C:\WinPE_amd64_PS
```

## Sample script

Use the following script to mount the Windows image, add the Windows PE optional components for Windows PowerShell, and to unmount the image.

```
Dism /Mount-Image /ImageFile:"C:\WinPE_amd64_PS\media\sources\boot.wim" /Index:1
/MountDir:"C:\WinPE_amd64_PS\mount"
Dism /Add-Package /Image:"C:\WinPE_amd64_PS\mount" /PackagePath:"C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-WMI.cab"
Dism /Add-Package /Image:"C:\WinPE_amd64_PS\mount" /PackagePath:"C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-WMI_en-
us.cab"
Dism /Add-Package /Image:"C:\WinPE_amd64_PS\mount" /PackagePath:"C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-NetFX.cab"
Dism /Add-Package /Image:"C:\WinPE_amd64_PS\mount" /PackagePath:"C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-
NetFX_en-us.cab"
Dism /Add-Package /Image:"C:\WinPE_amd64_PS\mount" /PackagePath:"C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-Scripting.cab"
Dism /Add-Package /Image:"C:\WinPE_amd64_PS\mount" /PackagePath:"C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-
Scripting_en-us.cab"
Dism /Add-Package /Image:"C:\WinPE_amd64_PS\mount" /PackagePath:"C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-
PowerShell.cab"
Dism /Add-Package /Image:"C:\WinPE_amd64_PS\mount" /PackagePath:"C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-
PowerShell_en-us.cab"
Dism /Add-Package /Image:"C:\WinPE_amd64_PS\mount" /PackagePath:"C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-
StorageWMI.cab"
Dism /Add-Package /Image:"C:\WinPE_amd64_PS\mount" /PackagePath:"C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-
StorageWMI_en-us.cab"
Dism /Add-Package /Image:"C:\WinPE_amd64_PS\mount" /PackagePath:"C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-
DismCmdlets.cab"
Dism /Add-Package /Image:"C:\WinPE_amd64_PS\mount" /PackagePath:"C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\en-us\WinPE-
DismCmdlets_en-us.cab"
Dism /Unmount-Image /MountDir:C:\WinPE_amd64_PS\mount /Commit
```

## Install this version of Windows PE to a USB key

```
MakeWinPEMedia /UFD C:\WinPE_amd64_PS F:
```

## Start Windows PowerShell in Windows PE

After you boot a PC to Windows PE using this USB key, start Windows PowerShell:

```
X:\Windows\system32\WindowsPowerShell\v1.0\powershell
```

## Related topics

[WinPE for Windows 10](#)

[WinPE: Add packages \(Optional Components Reference\)](#)

[WinPE: Create USB Bootable drive](#)

[WinPE: Create a Boot CD, DVD, ISO, or VHD](#)

[WinPE: Mount and Customize](#)

# WinPE: Store or split images to deploy Windows using a single USB drive

1/18/2019 • 3 minutes to read

How can you deploy Windows to PCs with just one USB port?

The default Windows Preinstallation Environment (WinPE) drive format, FAT32, is used to boot UEFI-based PCs, but that's too small to store most Windows images:

- FAT32 has a maximum file size of 4GB in size. Most customized Windows images are over 4GB.
- FAT32 has a maximum partition size of 32GB. Some Windows images are larger than 32GB.

(You can still use a 64GB or 128GB USB key, but you have to format it to use only uses 32GB of its space.)

Here's a few ways around these limitations:

## Option 1: Create a multiple partition USB drive

Starting with Windows 10, Version 1703, you can create multiple partitions on USB drives. To work with a USB drive with multiple partitions, both your technician PC and WinPE have to be Windows 10, Version 1703.

### Create a USB drive with WinPE and data partitions

1. Start the **Deployment and Imaging Tools Environment** as an administrator.
2. Type **diskpart** and press Enter.
3. Use Diskpart to reformat the drive and create two new partitions for WinPE and for your images:

```
List disk
select disk X    (where X is your USB drive)
clean
create partition primary size=2048
active
format fs=FAT32 quick label="WinPE"
assign letter=P
create partition primary
format fs=NTFS quick label="Images"
assign letter=I
Exit
```

4. Copy the WinPE files to the WinPE partition:

```
copype amd64 C:\WinPE_amd64
xcopy C:\WinPE_amd64\media P:\ /s
```

5. Copy the Windows image file to the Images partition:

```
xcopy C:\Images\install.wim I:\install.wim
```

## Option 2: Store the image on a separate USB drive

If you are using Windows 10, Version 1607 or earlier and your PC only has one USB port, you can still deploy Windows using two separate USB keys.

1. Boot to WinPE.
2. Remove the WinPE drive. (After booting, WinPE runs in memory.)
3. Plug in a separate storage drive with your image and apply it to the device.

## Option 3: Store the image on a network location

1. Copy the image to a server on your network, for example, \\server\share\install.wim
2. Boot to WinPE.
3. Connect a network drive using a drive letter, for example, N.

```
net use N: \\server\share
```

4. Apply the image from the network.

```
Dism /apply-image /imagefile:N:\install.wim /index:1 /applydir:D:\
```

## Option 4: Split the image

### Limitations:

- Applying split image (.swm) files is only supported when all of the .swm files are in the same folder, as shown in these steps.
- You can't modify a split .wim file.
- To use a 64GB or 128GB key, format it to only use 32GB of space.
- For images larger than 32GB, you need a second USB key because of the FAT32 partition size limitation.

1. From your technician PC, create your WinPE key. See [WinPE: Create USB Bootable drive](#).

2. Open the **Deployment and Imaging Tools Environment** as an administrator.

3. Split the Windows image into files smaller than 4GB each:

```
Dism /Split-Image /ImageFile:C:\install.wim /SWMFile:C:\images\install.swm /FileSize:4000
```

where:

- `C:\images\install.wim` is the name and the location of the image file that you want to split.
- `C:\images\install.swm` is the destination name and the location for the split .wim files.
- `4000` is the maximum size in MB for each of the split .wim files to be created.

In this example, the `/split` option creates an `install.swm` file, an `install2.swm` file, an `install3.swm` file, and so on, in the `c:\Images` directory.

4. Copy the files to the WinPE key.
5. On the destination PC, boot to WinPE, and then apply the image using DISM /Apply-Image with `/ImageFile` and `/SWMFile` options, as shown:

```
Dism /Apply-Image /ImageFile:install.swm /SWMFile:install*.swm /Index:1 /ApplyDir:D:\
```

## Related topics

[WinPE: Identify drive letters with a script](#)

[Split a Windows image file \(.wim\) for FAT32 media or to span across multiple DVDs](#)

[DISM Image Management Command-Line Options](#)

# WinPE: Identify drive letters with a script

1/24/2019 • 2 minutes to read

WinPE drive letter assignments change each time you boot, and can change depending on which hardware is detected.

You can use a script to figure out which drive letter is which by searching for a file or folder.

This sample script looks for a drive that has a folder titled `Images`, and assigns it to a system variable: `%IMAGESDRIVE%`.

```
@echo Find a drive that has a folder titled Images.  
@for %%a in (C D E F G H I J K L M N O P Q R S T U V W X Y Z) do @if exist %%a:\Images\ set IMAGESDRIVE=%%a  
@echo The Images folder is on drive: %IMAGESDRIVE%  
@dir %IMAGESDRIVE%\Images /w
```

## Related topics

[WinPE for Windows 10](#)

[Wpeinit and Startnet.cmd: Using WinPE Startup Scripts](#)

[WinPE: Install on a Hard Drive \(Flat Boot or Non-RAM\)](#)

# WinPE: Storage Area Network (SAN) Policy

1/18/2019 • 3 minutes to read

Storage area network (SAN) functionality enables a computer to mount disks and other storage devices automatically from other computers. By configuring the SAN policy on a Windows Preinstallation Environment (Windows PE) image, you can control whether or not disks are automatically mounted and which disks can be mounted. You can also disable the policy to automatically mount disks.

## Configuring the SAN policy on a Windows PE image

For Windows PE images that are available in the Windows Assessment and Deployment Kit (Windows ADK), the default SAN policy is to mount available disks automatically. But if the SAN environment has many available disks, automatically mounting them might reduce the performance of Windows PE. The container ID determines the external and internal disk status. If the device container ID of a disk is the same as the root container ID, the disk is internal. Otherwise, it's an external disk. You can use the Setsanpolicy.cmd file in the Windows PE tools path to configure the SAN policy on a Windows PE image.

### To configure the SAN policy on a Windows PE image

1. Mount the Windows PE image to an available mount point. For example:

```
Dism /mount-image /imagefile:C:\winpe_x86\ISO\sources\boot.wim /index:<image_index>
/mountdir:C:\winpe_x86\mount
```

where *<image\_index>* is the number of the selected image in the .wim file.

2. Run the **setsanpolicy** command. For example:

```
Setsanpolicy.cmd <image_path> <policy_number>
```

where *<image\_path>* is the path of a mounted Windows PE image, and *<policy\_number>* is the SAN policy number.

These values are valid *<policy\_number>* values:

SAN POLICY NUMBER	DESCRIPTION
1	Mounts all available storage devices. This is the default value.
2	Mounts all storage devices except those on a shared bus.
3	Doesn't mount storage devices.

SAN POLICY NUMBER	DESCRIPTION
4	<p>New for Windows 8. Makes internal disks offline.</p> <div style="border: 1px solid black; padding: 5px;"> <p><b>Note</b></p> <p>All external disks and the boot disk are online.</p> </div>

This example shows how to configure the SAN policy on a Windows PE image to mount all disks except those disks on a shared bus:

```
Setsanpolicy C:\winpe_x86\mount <2>
```

where <2> is the SAN policy number that mounts all storage device except those on a shared bus.

- Unmount the image and commit the changes. For example:

```
Dism /unmount-image /mountdir:C:\winpe_x86\mount /commit
```

## Configuring the SAN Policy on a Windows Image

You can change the default SAN policy of a Windows image by using Windows System Image Manager (Windows SIM) to customize the Microsoft-Windows-PartitionManager component. You use the `SanPolicy` setting to configure the Windows image during an unattended installation.

### To configure the SAN policy by using an answer file

- On your technician computer, open Windows System Image Manager (Windows SIM). Click **Start**, type **Windows System Image Manager**, and then select **Windows System Image Manager**.
- Create a new answer file, or update an existing answer file. For more information, see [Create or Open an Answer File](#) and [Best Practices for Authoring Answer Files](#).
- On the **Insert** menu, click **RunSynchronous**.
- Select the configuration pass where you want to install the command. This can be the `auditUser` or `oobeSystem` configuration pass.

#### Note

Don't use the **RunSynchronousNetsh advfirewall** command during the **specialize** configuration pass.

The **Create Synchronous Command** dialog box appears.

- Enter the **Netsh advfirewall firewall** commands to add them to the answer file, and then click **OK**.

For more information, see the [Network Shell \(Netsh\) Technical Reference](#). You can convert **Netsh** commands to Windows PowerShell® commands. For more information, see the [Netshell to Powershell Conversion Guide](#).

- In the **SynchronousCommand Properties** pane, in the **Settings** section next to **Description**, enter a description like **Enable Windows Messenger**.

## Related topics

[WinPE for Windows 10](#)

[WinPE: Mount and Customize](#)

[WinPE Network Drivers: Initializing and adding drivers](#)

[DISM Image Management Command-Line Options](#)

[Configure Network Settings in an Unattended Installation](#)

[Windows Deployment Options](#)

# WinPE Network Drivers: Initializing and adding drivers

1/24/2019 • 3 minutes to read

The Wpeutil command initializes the Windows PE (WinPE) network drivers as soon as WinPE boots. The default WinPE image includes support for many popular network adapters, and supports many of the same networking commands as in Windows. Windows PE includes a basic set of network drivers for many popular network adapters, and supports many of the same networking commands as in Windows.

Networking in WinPE has the following limitations:

- The supported methods of connecting to file servers are TCP/IP and NetBIOS over TCP/IP. Other methods, like the Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX) network protocol are not supported.
- Distributed File System (DFS) name resolution is supported for stand-alone namespaces only. It doesn't support domain namespaces. Stand-alone DFS namespaces allow for a DFS namespace that exists only on the local PC and therefore doesn't use Active Directory Domain Services (AD DS).
- General wireless networking functionality is not supported in WinPE.
- Connecting to an IPv4 network from Windows PE on an IPv6 network is not supported.
- Starting with WinPE for Windows 10, version 1709, SMB1 protocol is disabled by default. You can enable SMB1 support by running `dism.exe /enable-feature /featurename=SMB1Protocol-client`.

## To connect to another PC or shared folder on a network

1. While in Windows PE, you can connect (or map) to a shared network folder by using the [net use](#) command. If you're connecting to a domain-joined PC, Windows PE prompts for a username and password.

```
net use n: \\server\share
```

2. You can also host Windows PE from a network by using Preboot Execution Environment (PXE), which is part of [Windows Deployment Services](#).

## Troubleshooting networking problems

1. Try adding a driver for your network device.

We recommend [WinPE: Mount and Customize](#), especially for any driver that requires a reboot during the installation process.

You may also be able to use the [Drvload Command-Line Options](#) to load some drivers while Windows PE is running. However, any updates made to the registry during the installation process will not persist after a reboot, even when Windows PE is running in a [WinPE: Install on a Hard Drive \(Flat Boot or Non-RAM\)](#).

2. Run [Wpeinit and Startnet.cmd: Using WinPE Startup Scripts](#) to initialize the network. By default, wpeinit runs when Windows PE starts.
3. In some cases, you may need to configure firewall settings on the PC that you are trying to connect to. Windows PE supports [IPSec configuration](#).
4. Note, you cannot join Windows PE to a domain, or run Windows PE as a server. For more information, see

WinPE for Windows 10.

**To connect to a wired network using 802.1x authentication protocols**

1. Create a custom Windows PE image that includes the **WinPE-Dot3Svc** optional component.
2. Boot a PC to Windows PE.
3. Start the dot3svc service.

```
net start dot3svc
```

4. Add a LAN profile. For example:

```
netsh lan add profile="G:\EthernetLANProfile.xml"
```

Sample LAN Profile:

```

<?xml version="1.0"?>
<!!-- Sample LAN profile: EthernetLANProfile.xml" -->
<LANProfile xmlns="https://www.microsoft.com/networking/LAN/profile/v1">
    <MSM>
        <security>
            <OneXEnforced>false</OneXEnforced>
            <OneXEnabled>true</OneXEnabled>
            <OneX xmlns="https://www.microsoft.com/networking/OneX/v1">
                <cacheUserData>true</cacheUserData>
                <authMode>user</authMode>
                <EAPConfig><EapHostConfig
                    xmlns="https://www.microsoft.com/provisioning/EapHostConfig"><EapMethod><Type
                    xmlns="https://www.microsoft.com/provisioning/EapCommon">25</Type><VendorId
                    xmlns="https://www.microsoft.com/provisioning/EapCommon">0</VendorId><VendorType
                    xmlns="https://www.microsoft.com/provisioning/EapCommon">0</VendorType><AuthorId
                    xmlns="https://www.microsoft.com/provisioning/EapCommon">0</AuthorId></EapMethod><Config
                    xmlns="https://www.microsoft.com/provisioning/EapHostConfig"><Eap
                    xmlns="https://www.microsoft.com/provisioning/BaseEapConnectionPropertiesV1">
                        <Type>25</Type><EapType
                            xmlns="https://www.microsoft.com/provisioning/MsPeapConnectionPropertiesV1">
                        <ServerValidation>
                            <DisableUserPromptForServerValidation>false</DisableUserPromptForServerValidation>
                            <ServerNames></ServerNames>
                            <TrustedRootCA>1a 2b 3c 4d 56 78 90 aa bb cc dd ee ff 1a 2b 3c 4d 5e 6f</TrustedRootCA>
                        </ServerValidation><FastReconnect>true</FastReconnect>
                            <InnerEapOptional>false</InnerEapOptional><Eap
                                xmlns="https://www.microsoft.com/provisioning/BaseEapConnectionPropertiesV1">
                            <Type>26</Type><EapType
                                xmlns="https://www.microsoft.com/provisioning/MsChapV2ConnectionPropertiesV1">
                            <UseWinLogonCredentials>false</UseWinLogonCredentials></EapType></Eap>
                            <EnableQuarantineChecks>false</EnableQuarantineChecks>
                            <RequireCryptoBinding>false</RequireCryptoBinding><PeapExtensions>
                            <PerformServerValidation
                                xmlns="https://www.microsoft.com/provisioning/MsPeapConnectionPropertiesV2">false
                            </PerformServerValidation><AcceptServerName
                                xmlns="https://www.microsoft.com/provisioning/MsPeapConnectionPropertiesV2">false
                            </AcceptServerName><PeapExtensionsV2
                                xmlns="https://www.microsoft.com/provisioning/MsPeapConnectionPropertiesV2">
                            <AllowPromptingWhenServerCNotFound
                                xmlns="https://www.microsoft.com/provisioning/MsPeapConnectionPropertiesV3">true
                            </AllowPromptingWhenServerCNotFound></PeapExtensionsV2></PeapExtensions></EapType>
                        </Eap></Config></EapHostConfig></EAPConfig>
                    </OneX>
                </security>
            </MSM>
        </LANProfile>

```

## 5. Link the EAP User Data with the profile. For example:

```
netsh lan set eapuserdata filename="g:\EAP_UserData.xml" alluser=yes Interface="ethernet"
```

Sample EAP User Data file:

```

<?xml version="1.0"?>
<!-- Sample EAP user data: EAP_UserData.xml -->
<EapHostUserCredentials
  xmlns="https://www.microsoft.com/provisioning/EapHostUserCredentials"
  xmlns:eapCommon="https://www.microsoft.com/provisioning/EapCommon"
  xmlns:baseEap="https://www.microsoft.com/provisioning/BaseEapMethodUserCredentials">
  <EapMethod>
    <eapCommon:Type>25</eapCommon:Type>
    <eapCommon:AuthorId>0</eapCommon:AuthorId>
  </EapMethod>
  <Credentials
    xmlns:eapUser="https://www.microsoft.com/provisioning/EapUserPropertiesV1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:baseEap="https://www.microsoft.com/provisioning/BaseEapUserPropertiesV1"
    xmlns:MsPeap="https://www.microsoft.com/provisioning/MsPeapUserPropertiesV1"
    xmlns:MsChapV2="https://www.microsoft.com/provisioning/MsChapV2UserPropertiesV1">
    <baseEap:Eap>
      <baseEap:Type>25</baseEap:Type>
      <MsPeap:EapType>
        <MsPeap:RoutingIdentity>onex\administrator</MsPeap:RoutingIdentity>
        <baseEap:Eap>
          <baseEap:Type>26</baseEap:Type>
          <MsChapV2:EapType>
            <MsChapV2:Username>actualuser</MsChapV2:Username>
            <MsChapV2:Password>actualpassword</MsChapV2:Password>
            <MsChapV2:LogonDomain>actualdomain</MsChapV2:LogonDomain>
          </MsChapV2:EapType>
        </baseEap:Eap>
      </MsPeap:EapType>
    </baseEap:Eap>
  </Credentials>
</EapHostUserCredentials>

```

6. For more info, see [How to enable computer-only authentication for an 802.1X-based network in Windows Vista, in Windows Server 2008, and in Windows XP Service Pack 3](#).

## Related topics

[WinPE for Windows 10](#)

[WinPE: Mount and Customize](#)

[Wpeinit and Startnet.cmd: Using WinPE Startup Scripts](#)

[Drvload Command-Line Options](#)

# WinPE: Create Apps

1/18/2019 • 6 minutes to read

Windows PE (WinPE) is licensed to original equipment manufacturers (OEMs) to create customized deployment and recovery utilities. This topic provides guidelines for OEMs to develop deployment and recovery apps that run in Windows PE.

**Note** Windows PE is not a general-purpose operating system. It may not be used for any purpose other than deployment and recovery. It should not be used as a thin client or an embedded operating system.

## Extensibility

The majority of Windows PE apps are fixed-function shell apps that provide their own GUI. Two examples are the Windows Setup app and the Windows Recovery Environment (Windows RE).

- If it is acceptable to show a command prompt, then modify Startnet.cmd – this is the most convenient way to automatically start an app. See [WinPE: Mount and Customize](#).
- To have your app bypass the command line and start in your GUI, use Winpeshl.exe, Wpeinit.exe, wpeutil.exe, and wpeutil.dll.

## Winpeshl.exe, Wpeinit.exe, wpeutil.exe, and wpeutil.dll

By default, Winpeshl.exe is the first process run when Windows PE is booted. This is specified by the following registry value of type REG\_SZ.

```
HKEY_LOCAL_MACHINE  
  System  
    Setup  
      CmdLine
```

Winpeshl.exe searches for a file called Winpeshl.ini. If the file does not exist, Winpeshl.exe starts a Cmd.exe process that executes the Startnet.cmd script. If Winpeshl.ini does exist and it contains apps to launch, these apps are executed instead of Cmd.exe.

Wpeinit.exe installs Plug and Play (PnP) devices, starting the networking stack, and processing Unattend.xml settings when Windows PE starts. For more information, see [Wpeinit and Startnet.cmd: Using WinPE Startup Scripts](#).

Networking can be started at any time by running either by allowing Wpeinit.exe to run when Windows PE starts, or by running the [Wpeutil Command-Line Options](#) command.

Customized shell apps can call directly into Wpeutil.dll with the [LoadLibrary](#) and [GetProcAddress](#) functions. For related information, see [INFO: Alternatives to Using GetProcAddress\(\) With LoadLibrary\(\)](#).

Each of the functions exported by Wpeutil.dll has the same function signature as [WinMain Function](#), as illustrated in the following code sample.

```
int InitializeNetworkingW(
HINSTANCE hInstance,
HINSTANCE hPrevInstance,
LPSTR lpCmdLine,
int nCmdShow
);
```

The following code sample illustrates how to initialize networking.

```

#include <windows.h>
#include <tchar.h>
#include <stdio.h>
typedef int (*WpeutilFunction)(
HINSTANCE hInst,
HINSTANCE hPrev,
LPTSTR lpszCmdLine,
int nCmdShow
);
int __cdecl _tmain( int argc, TCHAR *argv[] )
{
    HMODULE hWpeutil = NULL;
    WpeutilFunction InitializeNetwork = NULL;
    int result = 0;
    TCHAR szCmdLine[] = _T("");
    hWpeutil = LoadLibrary( _T("wpeutil") );
    if( NULL == hWpeutil )
    {
        _tprintf( _T("Unable to load wpeutil.dll \n") );
        return GetLastError();
    }
    InitializeNetwork = (WpeutilFunction)GetProcAddress(
hWpeutil,
"InitializeNetworkW"
);
    if( NULL == InitializeNetwork )
    {
        FreeLibrary( hWpeutil );
        return GetLastError();
    }
    result = InitializeNetwork( NULL, NULL, szCmdLine, SW_SHOW );
    if( ERROR_SUCCESS == result )
    {
        _tprintf( _T("Network initialized. \n") );
    }
    else
    {
        _tprintf( _T("Initialize failed: 0x%08x"), result );
    }
    FreeLibrary( hWpeutil );
}

```

For a complete list of Wpeutil.dll exports, see [Wpeutil Command-Line Options](#).

# Visual Studio project settings

Some basic Visual Studio project settings may be different from the defaults created by the Visual Studio Project Wizard. Ensure that you set up your project's build settings to produce apps and DLLs that are compatible with Windows PE, as follows:

1. You must develop Windows PE apps with native C or C++ code that does not use MFC or ATL. Therefore, if you use the Visual Studio Project Wizard, choose a Win32 project and make sure that neither MFC nor ATL are checked.
2. Set your project options to link to the static C/C++ runtime libraries, not the .dll version of Msrvct.dll.
3. Open your project properties and set **Configuration Properties \ C/C++ Runtime Library** to **Multi-threaded** or **Multi-threaded debug**, not one of the .dll versions. If you do not perform this step, your app might not run on Windows PE.
4. If you plan to host your app on the 64-bit version of Windows PE, set the project build options to compile all binaries with the x64 compiler in Visual Studio.
5. If you plan to host your app on the 32-bit version of Windows PE, set the project options to compile with the x86 compiler.
6. Ensure that your project does not have the /clr: compiler option set. This option produces managed C++ code, which will not run on Windows PE.

**Warning** Your app can use customized .dll files that you write or license from a third party. Add these .dll files to your app for Windows PE. However, do not use Msrvct.dll and do not include additional Windows .dll files that are not part of Windows PE.

## API Compatibility reference

Windows PE is a lightweight, bootstrap operating system based on a subset of components from the Windows operating system. It is designed to host deployment and recovery apps. As such, it contains many Windows binaries that are needed to host the APIs that are most important to these classes of app. Due to size and other design constraints, not all Windows binaries are present in Windows PE, and therefore not all Windows APIs are present or usable.

### Supported APIs in Windows PE

The following APIs are supported in Windows PE:

1. [Windows API sets \(Mincore.lib\)](#).
2. [Deployment Image Servicing and Management \(DISM\) API \(Dismapi.lib\)](#).
3. [Imaging APIs for Windows \(Wimgapi.lib\)](#).

If an API behaves the same as it does on the full Windows operating system and as documented in the Windows SDK for Windows operating system, it will be considered supported and can be used by apps unless otherwise noted. Because Windows PE is based on components from Windows, it contains a significant subset of Windows APIs that are published in the Windows SDK for Windows operating system. The parameters, calling conventions, and behaviors of these supported APIs will be the same or nearly the same as on the full Windows operating system, unless they are affected by the unique Windows PE environment. Apps using only these APIs should be portable between the full Windows operating system and Windows PE.

In some cases, a subset of the possible parameter values will be usable on Windows PE. This may be due to conditions unique to the runtime environment, such as running on a read-only medium, not having access to persistent state, or other design limitations. In this case, the API may not be supported, but may still be used to accomplish a specific task if there is no other alternative.

In general, if an API works incorrectly or not at all on Windows PE, it is not supported and must not be used, even if it resides in a binary that is included in Windows PE. The API may be failing because Windows PE is a subset of the Windows operating system, or because of the runtime design considerations unique to Windows PE. Such failures are not considered bugs in Windows PE.

Because many Windows components are not present in Windows PE, many APIs are not available. They may be completely missing because the Windows binary in which they reside is not present. Alternatively, they may be only partially present because although the Windows binary in which they reside is present, one or more binaries they depend on are not. In addition, some APIs that are present in Windows PE do not work correctly and behave differently than they do in Windows. These APIs are unsupported and must not be used, because their behavior on Windows PE is undefined.

Sometimes, there may be no suitable API to accomplish a specific task. To find an alternate solution, you would require different app logic, different algorithm design, or redefinition of the underlying problem.

## Related topics

[WinPE for Windows 10](#)

[WinPE: Debug Apps](#)

# WinPE: Debug Apps

1/18/2019 • 3 minutes to read

You can use Windows Debuggers, such as Ntsd.exe, Cdb.exe, and Windbg.exe, and supporting tools to debug applications on Windows PE and to debug the Windows PE kernel. Debugging tools are included in the [Windows 10 SDK](#). You must make the debugging tools available on the Windows PE computer by either copying them locally or using them from a share.

To debug Windows PE remotely, you may need to turn off the built-in firewall on the PC:

```
wpeutil disablefirewall
```

## User-mode debugging

The easiest user-mode debugging method is to run a process server on the Windows PE computer, and connect to it by using a debugger on another computer. The process server is included with the debugging tools in the [Windows 10 SDK](#).

### To run a process server in user-mode

1. Copy the Windows Debugging Process Server tool: **dbgsrv.exe**, from the [Windows 10 SDK](#) debugging tools folder (example: C:\Program Files (x86)\Windows Kits\10.0\Debuggers\x64), to the Windows PE computer.
2. At the Windows PE command prompt, disable the firewall.

```
wpeutil disablefirewall
```

3. Start the Windows Debugging Process Server, specifying a connection method to the PC, for example, a TCP port:

```
dbgsrv.exe -t tcp:port=1234
```

For more information, see [Activating a Process Server \(Windows Debuggers\)](#).

4. From the remote computer, use the process server to attach to or start processes on the Windows PE destination computer:

```
windbg -premove tcp:server=Server,port=1234
```

For more information, see [Activating a Smart Client \(Windows Debuggers\)](#).

It is also possible to run the debugger directly on the Windows PE computer. However, doing so requires setting up symbol and source paths after every reboot of the Windows PE computer. We recommend that you perform debugging from a computer running a full version of Windows, as described in this procedure.

The following debugging procedure is useful when you want to bypass startnet.cmd or setup.exe, and proceed directly to a command prompt for debugging purposes. This procedure bypasses all initialization, including setup, and runs no commands, such as Wpeinit.exe. This procedure must be performed online on an online operating

system.

### To enable user-mode debugging prior to any initialization

1. Delete the winpeshl.ini file, if it exists. If the winpeshl.ini file does not exist, then user-mode debugging can be accessed by default.
2. Hold down the Ctrl key during boot before the command prompt is shown. A command prompt appears.
3. Proceed with debugging.

## Kernel-mode debugging

To debug in kernel-mode, you must enable kernel-mode debugging before the system is booted. The boot configuration file has a setting for kernel mode debugging, which is enabled by using the bcdedit.exe command-line tool to modify the Boot Configuration Data (BCD) store. Kernel debugging can only be performed by using bcdedit.exe. Bcdedit.exe is located in the \Windows\System32 directory of the Windows partition.

The default debugger settings are as follows:

```
identifier          {dbgsettings}
debugtype          Serial
debugport          1
baudrate          115200
```

For creating ISOs for VM environments, enable the kernel with BCD entries before creating the ISO.

For information about how to modify the default BCD store (default.bcd), see [How to Modify the BCD Store Using Bcdedit](#).

### To enable kernel-mode debugging

1. Locate the BCD store, which is contained in a file named **bcd**. This file is located within the boot directory in the root of the media containing the Windows PE image.
2. At the command prompt, type the following bcdedit command to set the debug flag of the BCD store used to boot the image to `debug on`:

```
bcdedit /store <path to winpe>/boot/bcd /set {default} debug on
```

The `{default}` might need to be replaced by the unique identifier (UID) of the boot option for Windows PE.

Alternatively, you can also enable kernel debugging by pressing F8 during boot and selecting the debug option.

**Note** To use a symbol server from within Windows PE, use the `net use` command on the server's symbols and file shares.

For more information about command-line options that control debugging, see [BCDEdit Command-Line Options](#).

## Related topics

[WinPE for Windows 10](#)

[WinPE: Mount and Customize](#)

[Wpeutil Command-Line Options](#)

Winpeshl.ini Reference: Launching an app when WinPE starts

BCDEdit Command-Line Options

# Copype Command-Line Options

1/18/2019 • 2 minutes to read

The **Copype** tool creates a working directory that contains a standard set of Windows Preinstallation Environment (Windows PE) files. You use these files to customize images and (together with the **Makewinpemedia** script) to create bootable media. For more information, see [Makewinpemedia Command-Line Options](#).

## Copype Command-Line Options

**Copype** uses the following command-line options.

**Copype.cmd** architecture WorkingDirectory

COMMAND-LINE OPTION	DESCRIPTION
<i>architecture</i>	<p>Copies the boot files and the Windows PE base image (Winpe.wim) to &lt;WorkingDirectory&gt;\Media.</p> <p>Values include <b>amd64</b>, <b>x86</b>, or <b>arm</b>.</p> <p>The x86 version of Windows PE can boot 32-bit UEFI, 32-bit BIOS, or 64-bit BIOS-based PCs.</p> <p>The amd64 version of Windows PE can boot either 64-bit BIOS-based or 64-bit UEFI-based PCs.</p> <p>The arm version of Windows PE can boot ARM-based PCs.</p> <p>For more information about running Windows PE on PCs with different architectures, see <a href="#">Windows Setup Supported Platforms and Cross-Platform Deployments</a>.</p>
<i>WorkingDirectory</i>	<p>Specifies the name of the working directory where <b>Copype</b> creates the directory structure and copies the Windows PE files. For example:</p> <pre>copype amd64 C:\winpe_amd64</pre> <p><b>Copype</b> creates the following directory structure.</p> <pre>&lt;WorkingDirectory&gt; &lt;WorkingDirectory&gt;\media &lt;WorkingDirectory&gt;\mount</pre> <p>When <b>Copype</b> copies the Windows PE base image to the &lt;WorkingDirectory&gt;\Media\Sources folder, it renames the base image from Winpe.wim to Boot.wim.</p>

## Related topics

[WinPE for Windows 10](#)

[WinPE: Create USB Bootable drive](#)

## Makewinpemedia Command-Line Options

# Makewinpemedia Command-Line Options

1/18/2019 • 2 minutes to read

The **Makewinpemedia** tool is new for Windows 8. You can use **Makewinpemedia** to create bootable Windows Preinstallation Environment (Windows PE) media. Running the **Copype** tool is a prerequisite for creating bootable media. **Copype** creates a directory structure for Windows PE files and copies the necessary Windows PE media files. For more information, see [Copype Command-Line Options](#) and [WinPE: Create USB Bootable drive](#).

## Makewinpemedia Command-Line Options

The **Makewinpemedia** tool uses the following command-line options.

**Makewinpemedia** {*/ufd* | */iso*} [*/f*] <*WorkingDirectory*> <*DestinationLocation*>

COMMAND-LINE OPTION	DESCRIPTION
<b>/ufd</b>	Specifies a USB flash drive as the type of media to create. For example:  <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">Makewinpemedia /ufd C:\winpe_amd64 F:</div> where F is the drive letter of the USB flash drive.
<b>/iso</b>	Specifies a .iso file (CD or DVD) as the type of media to create. For example:  <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">Makewinpemedia /iso C:\winpe_amd64 C:\winpe_x64\winpe_amd64.iso</div>
<b>/f</b>	Optional. Suppresses the confirmation message that appears before you format the USB flash drive or overwrite an existing .iso file. For example:  <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">Makewinpemedia /ufd /f C:\winpe_amd64 F:</div> where F is the drive letter of the USB flash drive.
< <i>WorkingDirectory</i> >	Specifies the name of the working directory where the <b>Copype</b> tool creates the Windows PE directory structure and copies the necessary files for creating bootable media. For example:  <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">C:\winpe_amd64</div>

COMMAND-LINE OPTION	DESCRIPTION
<DestinationLocation>	Specifies the drive letter of the USB flash drive if you are using the <b>/ufd</b> option, or the name of the .iso file if you are using the <b>/iso</b> option.

## Related topics

[WinPE for Windows 10](#)

[WinPE: Create USB Bootable drive](#)

[WinPE: Mount and Customize](#)

[Oscdimg Command-Line Options](#)

# Drvload Command-Line Options

1/18/2019 • 2 minutes to read

The Drvload tool adds out-of-box drivers to a booted Windows Preinstallation Environment (Windows PE) image. It takes one or more driver .inf files as inputs. To add a driver to an offline Windows PE image, use the Deployment Image Servicing and Management (DISM) tool. For more information, see [Add and Remove Drivers to an Offline Windows Image](#).

If the driver .inf file requires a reboot, Windows PE will ignore the request. If the driver .sys file requires a reboot, then the driver cannot be added with Drvload. For more information, see [Device Drivers and Deployment Overview](#) and [DISM Driver Servicing Command-Line Options](#).

Drivers added using the Drvload tool are marked as the preferred driver for that device. If you add an updated driver during Windows Setup, the driver that you added with Drvload takes precedence.

## Drvload Command-Line Options

The following command-line options are available for Drvload.

**drvload** *inf\_path* [*inf\_path* [...]] [/?]

OPTION	DESCRIPTION
/?	Displays usage information.
<i>inf_path</i>	Specifies the path to the driver .inf file. The path can contain environment variables.

If any drivers were not installed, then Drvload will return a non-zero status (%errorlevel%).

## Related topics

[WinPE for Windows 10](#)

[WinPE: Mount and Customize](#)

# Winpeshl.ini Reference: Launching an app when WinPE starts

1/18/2019 • 2 minutes to read

Use the **Winpeshl.ini** file in Windows Preinstallation Environment (Windows PE) to replace the default command prompt with a shell application or other app. For example, your shell app might provide a GUI for deployment engineers to choose a method of installing Windows.

To add a customized app, create a file named Winpeshl.ini and place it in %SYSTEMROOT%\System32 a customized Windows PE image. For more information, see [WinPE: Mount and Customize](#).

## Example

```
[LaunchApp]
AppPath = %SYSTEMDRIVE%\Fabrikam\shell.exe
[LaunchApps]
%SYSTEMDRIVE%\Fabrikam\app1.exe
%SYSTEMDRIVE%\Fabrikam\app2.exe, /s "C:\Program Files\App3"
```

The Wpeshl.ini file may have either or both of the sections: [LaunchApp] and [LaunchApps]. The apps listed in [LaunchApp] and [LaunchApps] run in order of appearance, and don't start until the previous app has terminated.

## LaunchApp

Set the `AppPath` entry to the path to your app. You can use a fully qualified path, or you can include environment variables, such as `%SYSTEMDRIVE%` to describe the path.

### Note

- The [LaunchApp] entry may only include one app.
- You can't specify a command that is greater than 250 characters.
- You can't specify any command-line options with LaunchApp.

## LaunchApps

Use the `[LaunchApps]` section to run apps with command-line options.

### Note

- LaunchApps supports running apps, but does not support common scripting commands. To run commands, add a startup script instead (startnet.cmd). For more information, see [WinPE: Mount and Customize](#).
- You can't specify a command that is greater than 250 characters.
- To add command-line options to an app: add a comma (,) after the app name:  
`%SYSTEMDRIVE%\Fabrikam\app2.exe, <option>`

## Related topics

[WinPE for Windows 10](#)

[WinPE: Debug Apps](#)

# Wpeinit and Startnet.cmd: Using WinPE Startup Scripts

1/18/2019 • 2 minutes to read

Use Wpeinit and Startnet.cmd to run startup scripts when Windows PE (WinPE) first runs.

Wpeinit outputs log messages to **C:\Windows\system32\wpeinit.log**.

## Startnet.cmd

You can add customized command-line scripts in Windows PE by using Startnet.cmd. By default, Windows PE includes a Startnet.cmd script located at %SYSTEMROOT%\System32 of your customized Windows PE image.

Startnet.cmd starts Wpeinit.exe. Wpeinit.exe installs Plug and Play devices, processes Unattend.xml settings, and loads network resources.

For more info, see [WinPE: Mount and Customize](#).

## Wpeinit Command-Line Options

The following command-line option is available for Wpeinit:

**Wpeinit [-unattend:<path\_to\_answer\_file>]**

Example:

```
Wpeinit -unattend:"C:\Unattend-PE.xml"
```

## Supported Unattend settings

You can create an answer file and include any of the following settings for use with Windows PE:

- Microsoft-Windows-Setup/[Display](#)
- Microsoft-Windows-Setup/[EnableFirewall](#)
- Microsoft-Windows-Setup/[EnableNetwork](#)
- Microsoft-Windows-Setup/[LogPath](#)
- Microsoft-Windows-Setup/[PageFile](#)
- Microsoft-Windows-Setup/[Restart](#)
- Microsoft-Windows-Setup/[RunAsynchronous](#)
- Microsoft-Windows-Setup/[RunSynchronous](#)

## Related topics

[WinPE: Identify drive letters with a script](#)

[WinPE for Windows 10](#)

[Winpeshl.ini Reference: Launching an app when WinPE starts](#)

[WinPE: Mount and Customize](#)

[Unattended Windows Setup Reference](#)

# Wpeutil Command-Line Options

1/18/2019 • 4 minutes to read

The Windows PE utility (Wpeutil) is a command-line tool that enables you to run commands during a Windows PE session. For example, you can shut down or restart Windows PE, enable or disable a firewall, set language settings, and initialize a network.

Wpeutil uses the following conventions.

**Wpeutil** {command} [argument]

For example:

```
Wpeutil Shutdown  
Wpeutil Enablefirewall  
Wpeutil SetMuiLanguage de-DE
```

## IMPORTANT

Wpeutil can only accept one command per line.

OPTION	DESCRIPTION
<b>CreatePageFile</b> [/path=<path>] [/size=<size>]	<p>Creates a page file to a specified path and size. The default path is C:\pagefile.sys and default size is 64 megabytes. At least one option must be specified. For example:</p> <p><code>Wpeutil CreatePageFile /path=C:\pagefile.sys</code> -or- <code>Wpeutil CreatePageFile /path=C:\pagefile.sys /size=128</code></p> <p><b>Important</b> If a page file exists, the <b>CreatePageFile</b> option must be set equal to or greater than the current size of the page file or the command will fail.</p>
<b>DisableExtendedCharactersForVolume</b> <path_on_target_volume>	<p>Disables extended character support for DOS-compatible file names (8.3 format) for the volume that contains <i>path on target volume</i>. This command only applies to NTFS volumes. The <i>path on target volume</i> must specify the root of the volume. For example:</p> <p><code>Wpeutil DisableExtendedCharactersForVolume C:\</code></p> <p>If disabled, all files that have been created with extended characters will be converted to a short file name.</p>
<b>DisableFirewall</b>	<p>Disables a firewall. For example:</p> <p><code>Wpeutil DisableFirewall</code></p>

OPTION	DESCRIPTION
<b>EnableExtendedCharactersForVolume &lt;path_on_target_volume&gt;</b>	<p>Allows 8.3 format file names to contain extended characters on the volume that contains <i>path on target volume</i>. This command only applies to NTFS volumes. The <i>path on target volume</i> must specify the root of the volume. For example:</p> <pre>Wpeutil EnableExtendedCharactersForVolume C:\</pre> <p><b>Note:</b></p> <p>If you are installing an operating system in a language that has extended characters that are enabled by default, such as ja-JP or ko-KR, or using a copy of Windows PE in a language that doesn't have extended characters enabled, such as en-US, the installation will cause a Chkdsk error during first boot. Enabling this option before you install to that volume will prevent Chkdsk command from running.</p>
<b>EnableFirewall</b>	<p>Enables a firewall. For example:</p> <pre>Wpeutil EnableFirewall</pre>
<b>InitializeNetwork [/NoWait]</b>	<p>Initializes network components and drivers, and sets the computer name to a randomly-chosen value. The <b>/NoWait</b> option will skip the time where your PC would otherwise wait to acquire an IP address. If you don't use <b>/NoWait</b>, Windows PE will wait to acquire an address before it finishes loading your WinPE session. <b>/NoWait</b> is helpful for environments that don't use DHCP. For example:</p> <pre>Wpeutil InitializeNetwork</pre> <pre>wpeutil InitializeNetwork /NoWait</pre>
<b>ListKeyboardLayouts &lt;LCID&gt;</b>	<p>Lists the supported keyboard layouts (Name and ID) for a given Locale ID (LCID) value. The keyboard layouts will also be updated in the registry under the key:  <b>HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\WinPE\KeyboardLayouts</b>. For example:</p> <pre>Wpeutil ListKeyboardLayouts 0x0409</pre> <p>-or-</p> <pre>Wpeutil ListKeyboardLayouts 1033</pre> <p>For a list of valid Locale IDs, see <a href="#">Microsoft Locale ID Values</a>.</p>
<b>Reboot</b>	<p>Restarts the current Windows PE session. For example:</p> <pre>Wpeutil Reboot</pre>
<b>SaveProfile</b>	<p>Stops logging and saves the custom profile to the location the user specified earlier with the <b>Dism /enable-profiling</b> command. For more information about the <b>/enable-profiling</b> command-line option, see <a href="#">DISM Windows PE Servicing Command-Line Options</a>. For example:</p> <pre>Wpeutil SaveProfile profile_file_name "short description"</pre>
<b>SetKeyboardLayout&lt;keyboard_layout_ID&gt;</b>	<p>Sets the keyboard layout in the current Windows PE session. This will take effect for processes after the command succeeds. To obtain a list of supported keyboard layouts, enter:</p> <pre>ListKeyboardLayouts LCID</pre> <p>To set the keyboard for en-US, for example:</p> <pre>Wpeutil SetKeyboardLayout 0409:00000409</pre>

OPTION	DESCRIPTION
<b>SetMuiLanguage</b> <language-name>[;<language-name>]	<p>Sets the language. &lt;language-name&gt; uses the international language code format (for example, en-US for the U.S. English language). You can specify multiple languages in priority order, by separating them with a semicolon. For example:</p> <pre>Wpeutil SetMuiLanguage de-DE;en-US</pre>
<b>SetUserLocale</b> <language-name>[;<language-name>]	<p>Sets the user locale. &lt;language-name&gt; uses the international language code format (for example, en-US for the U.S. English language). You can specify multiple languages in priority order, by separating them with a semicolon. For example:</p> <pre>Wpeutil SetUserLocale de-DE;en-US</pre>
<b>Shutdown</b>	<p>Shuts down the current Windows PE session. For example:</p> <pre>Wpeutil Shutdown</pre> <p><b>Note:</b> You can also do the following in the Command Prompt window:</p> <ul style="list-style-type: none"> <li>• Click the <b>Close</b> button</li> <li>• Type EXIT</li> </ul>
<b>UpdateBootInfo</b>	<p>Populates the registry with information about how Windows PE boots.</p> <p>After you run this command, query the registry. For example:</p> <pre>wpeutil UpdateBootInfo reg query HKLM\System\CurrentControlSet\Control /v PEBootType</pre> <p>The results of this operation might change after loading additional driver support.</p> <p>To determine where Windows PE is booted from, examine the following:</p> <ul style="list-style-type: none"> <li>• <b>PEBootType</b>: Error, Flat, Remote, Ramdisk:Sourceldentified Ramdisk:SourceUnidentified, Ramdisk:OpticalDrive</li> <li>• <b>PEBootTypeErrorCode</b>: HRESULT code</li> <li>• <b>PEBootServerName</b>: Windows Deployment Services server name</li> <li>• <b>PEBootServerAddr</b>: Windows Deployment Services server IP address</li> <li>• <b>PEBootRamdiskSourceDrive</b>: Source drive letter, if available.</li> <li>• <b>PEFirmwareType</b>: Firmware boot mode: 0x1 for BIOS, 0x2 for UEFI.</li> </ul> <p>If you are not booting Windows Deployment Services, the best way to determine where Windows PE booted from is to first check for PEBootRamdiskSourceDrive registry key. If it is not present, scan the drives of the correct PEBootType and look for some kind of tag file that identifies the boot drive.</p>
<b>WaitForNetwork</b>	<p>Waits for the network card to be initialized. Use this command when creating scripts to make sure that the network card has been fully initialized before continuing.</p>

OPTION	DESCRIPTION
<b>WaitForRemovableStorage</b>	During the Windows PE startup sequence, this command will block startup until the removable storage devices, such as USB hard drives, are initialized. For example: <code>Wpeutil WaitForRemovableStorage</code>

## Related topics

[WinPE for Windows 10](#)

[WinPE: Mount and Customize](#)

[DISM Windows PE Servicing Command-Line Options](#)

# Windows Setup Technical Reference

1/18/2019 • 2 minutes to read

Windows Setup is a bootable program that installs the Windows operating system.

## Practical applications

- You can install or upgrade the Windows operating system on a PC from a USB key, a mounted .ISO file, DVD, or network device.
- You can automate the Windows installation process, including the configuration of drivers, packages, files, and Windows system settings by using answer files created from [Windows System Image Manager Technical Reference](#).
- You can use Windows Setup as an installer for your own customized Windows images.
- You can use the menus in Windows Setup to prepare the hard drives before installation.

## What's New

- Windows 8.1 upgrades are different from previous Windows upgrade scenarios. For more info, see [Windows 8.1 Upgrade Scenarios for OEMs](#).
- Windows Setup cannot be used to perform automated upgrades to most editions of Windows 8.1.  
For volume-licensed editions of Windows, we've added a new command-line option, `setup /auto`, to help enable upgrades. Note, we only plan to use this option for upgrades to Windows 8.1, and we may remove the option in future versions of Windows. For more info, see [Windows Setup Command-Line Options](#).
- **Settings for Automating OOBED:** The [NetworkLocation](#) setting is no longer needed to automate OOBED. The functionality of the [ProtectYourPC](#) setting has changed.

## See also

The following table contains links to resources related to this scenario.

CONTENT TYPE	REFERENCES
<b>Planning</b>	<a href="#">Windows Setup Scenarios and Best Practices</a>   <a href="#">Windows Setup Automation Overview</a>
<b>Deployment</b>	<a href="#">Windows Setup Installation Process</a>   <a href="#">Windows 8.1 Upgrade Scenarios for OEMs</a>   <a href="#">Boot from a DVD</a>   <a href="#">Install Windows from a USB Flash Drive</a>   <a href="#">Deploy a Custom Image</a>   <a href="#">WinPE: Create USB Bootable drive</a>
<b>Operations</b>	<a href="#">Automate Windows Setup</a>   <a href="#">Use a Configuration Set with Windows Setup</a>   <a href="#">Add Device Drivers to Windows During Windows Setup</a>   <a href="#">Add a Custom Script to Windows Setup</a>   <a href="#">Add languages to Windows Setup</a>   <a href="#">Boot Windows to Audit Mode or OOBED</a>

CONTENT TYPE	REFERENCES
<b>Tools and settings</b>	<a href="#">Windows Setup Command-Line Options</a>   <a href="#">Windows Setup Supported Platforms and Cross-Platform Deployments</a>   <a href="#">Windows Setup States</a>   <a href="#">Windows Setup Edition Configuration and Product ID Files (EI.cfg and PID.txt)</a>   <a href="#">Windows Setup Log Files and Event Logs</a>   <a href="#">Windows Setup Configuration Passes</a>
<b>Related technologies</b>	<a href="#">Windows System Image Manager Technical Reference</a>   <a href="#">Unattended Windows Setup Reference</a>   <a href="#">Sysprep (System Preparation) Overview</a>   <a href="#">WinPE for Windows 10</a>

# Windows Setup Supported Platforms and Cross-Platform Deployments

1/18/2019 • 6 minutes to read

This topic describes the supported platforms and deployment scenarios for running for Windows Setup.

When you're deploying different types of PCs, you can use Windows Setup as a way to choose between your images through the Windows Setup user interface to select a specific image. You can include images for a variety of hardware platforms (such as BIOS and UEFI, 32-bit and 64-bit PCs), and across different versions of Windows (such as Windows 8.1, Windows Server 2012 R2, and Windows 7).

You can also run Windows Setup through a script. Boot the PC to Windows PE, and then use the `\sources\setup.exe` file to specify your image.

## Firmware considerations: BIOS vs. UEFI

For UEFI-based PCs that support booting into either UEFI or legacy BIOS modes, make sure your PC is booted into the correct firmware mode before starting Windows Setup. Otherwise, Windows Setup may set up the hard drive partitions incorrectly, or may abort the installation if the hard drives are preconfigured. For more information, see [WinPE: Boot in UEFI or legacy BIOS mode](#).

## Firmware: BIOS 32-bit and 64-bit

To set up a single environment or set of scripts that can deploy Windows to both 32-bit and 64-bit BIOS PCs, use a 32-bit version of Windows PE and a 32-bit version of Windows Setup.

The 64-bit version of Windows Setup does not run on the 32-bit version of Windows PE.

### To install a 64-bit version of Windows from a 32-bit version of Windows PE:

1. Boot the PC using the 32-bit version of Windows PE.
2. Use any of the following techniques to install a 64-bit version of Windows:
  - Run a 32-bit version of Windows Setup, and use the `/InstallFrom` command-line option to select a 64-bit Windows image:

```
X:\windows\system32> D:\setup /InstallFrom:"N:\Windows_64-bit\sources\install.wim"
```

-or-

- Run a 32-bit version of Windows Setup, and use the `Microsoft-Windows-Setup\ImageInstall\OSImage\InstallFrom` unattend setting to select a 64-bit Windows image.

```
X:\windows\system32> D:\setup /unattend:"D:\unattend_install_64-bit.xml"
```

-or-

- Use image-capturing tools to apply a 64-bit version of Windows to the PC.

```
Dism /Apply-Image /ImageFile:"Fabrikam_64-bit_image.wim" /Index:1 /ApplyDir:D:\
```

For more information, see [Apply Images Using DISM](#).

**Warning** This procedure does not support deploying Windows 7.

## Using Windows Setup to Install Previous Versions of Windows

You can use the Windows 8.1 and Windows Server 2012 R2 versions of Windows Setup to install previous versions of Windows:

HOST OPERATING SYSTEM	WINDOWS 8.1 SETUP SUPPORT
Windows 8.1	Yes
Windows Server 2012 R2	Yes
Windows 8	Yes
Windows Server 2012	Yes
Windows 7	Yes
Windows Server 2008 R2	Yes
Windows Vista	No
Windows Server 2008	No
Windows XP with SP3	No
Windows Server 2003 R2 and previous versions	No
Windows XP with SP2 and previous versions	No

You can also run Windows Setup from the Windows Preinstallation Environment (Windows PE). The following table lists the supported Windows PE environments:

VERSION OF WINDOWS SETUP	WINDOWS PE 5.0 (WINDOWS 8.1)	WINDOWS PE 4.0 (WINDOWS 8)	WINDOWS PE 3.0 (WINDOWS 7)	WINDOWS PE 2.0 (WINDOWS VISTA)
Windows 8.1 Setup	Yes	Yes	Yes	No

VERSION OF WINDOWS SETUP	WINDOWS PE 5.0 (WINDOWS 8.1)	WINDOWS PE 4.0 (WINDOWS 8)	WINDOWS PE 3.0 (WINDOWS 7)	WINDOWS PE 2.0 (WINDOWS VISTA)
Windows 8 Setup	No	Yes	Yes	Yes
Windows 7 Setup	No	No	Yes	Yes
Windows Vista Setup	No	No	No	Yes

## Cross-Platform Deployment

Cross-platform deployment is the process of installing a specific architecture of Windows from an environment of a different architecture. For example, you can deploy a 64-bit edition of Windows 8.1 or Windows 8 from a 32-bit edition of Windows PE. The benefit of using a cross-platform deployment solution is that you don't have to maintain multiple versions of Windows PE for installing different architecture editions of Windows. You can build a single Windows PE image that you can use to install both 32-bit and 64-bit editions of Windows.

When you install a 64-bit edition of Windows from a 32-bit version of Windows PE, you must use Windows PE 2.0 or a later version. For more information about Windows PE releases, see [WinPE for Windows 10](#).

The following table lists the different architecture types of Windows images (32-bit or 64-bit) that a specific version of Windows 8.1 Setup is able to install.

	64-BIT WINDOWS 8.1 IMAGE	32-BIT WINDOWS 8.1 IMAGE	64-BIT WINDOWS 8 IMAGE	32-BIT WINDOWS 8 IMAGE
64-bit Windows 8.1 Setup	Yes	No	Yes	No
32-bit Windows 8.1 Setup	Yes	Yes	No	Yes

### Limitations of cross-platform deployment

These cross-platform deployment scenarios aren't supported:

- Installing a 64-bit Windows image on a 32-bit computer.
- Deploying a 32-bit Windows image from a 64-bit preinstallation environment.
- Using a 32-bit version of Windows Setup to upgrade a 64-bit operating system.
- Using a 32-bit version of Windows 8 Setup to deploy a 64-bit version of the Windows 7 operating system.

For example, you must use a 64-bit version of Windows 8 Setup to deploy a 64-bit version of Windows 7. In previous releases, the version of Windows Setup version had to match the operating system that you would deploy. For example, you had to use the Windows 7 Setup.exe to install Windows 7.

- Using Microsoft Internet SCSI (iSCSI) boot disk in a cross-platform deployment scenario.

For example, installing Windows (64-bit version) from cross-platform media, such as Windows PE (32-bit

version), to an iSCSI boot disk is unsupported. You must use the same architecture for Windows PE as the target deployment architecture when you deploy Windows to an iSCSI boot disk.

- On Unified Extensible Firmware Interface (UEFI), deploying a 64-bit edition of Windows from a 32-bit version of Windows PE. On some UEFI computers, you can't install Windows in BIOS-compatibility mode and must switch to UEFI-compatibility mode. For more information, see [Boot to UEFI Mode or Legacy BIOS mode](#).
- On BIOS:
  - Performing cross-platform deployments, except as part of a clean installation, or performing a Windows Deployment Services deployment.
  - Providing cross-platform installation media to users for recovery.

To prevent users from installing the wrong edition of Windows for the architecture of their computer, don't provide cross-platform installation media to users for recovery or reinstallation. Also, the Windows Recovery Environment (Windows RE) feature that's included on the media applies only to 32-bit Windows installations.

### **Creating a .wim file for multiple architecture types**

If a .wim file contains both 32-bit and 64-bit Windows editions, you must select the Windows image that you want to install. Typically, Windows Setup uses the product key that you specify in the `ProductKey` setting to determine which Windows image to install. But if the file contains 2 editions of the same Windows version, like Windows 8.1 Pro, you must use the `MetaData` setting in an answer file to specify the edition to install.

To choose an image, specify metadata that corresponds to the image index, name, description, or architecture type. For the metadata for architecture type, use 0 for 32-bit editions and 9 for 64-bit editions. For more info, see the `MetaData` `Key` setting.

The answer file must include processor-specific components. The answer-file settings in the [windowsPE](#) configuration pass must match the architecture type of the preinstallation environment. The settings that apply to the Windows image must match the architecture type of the image. For example, if you create an answer file that deploys 64-bit images from a 32-bit preinstallation environment, all components in the answer file for the [windowsPE](#) configuration pass must include the processor attribute type of **x86**. Settings to be applied in the [specialize](#), [oobeSystem](#), or other configuration passes must include the processor attribute type of **amd64**.

### **Installing 64-bit drivers**

All drivers that are included with Windows are signed. In cross-architecture deployments, you can use an out-of-box device driver. But if you use an unsigned out-of-box device driver that's boot critical in a 64-bit installation, the installation may become unusable.

You can install 64-bit drivers for a Windows image during Windows Setup in either of these ways:

- In attended installations, you can press F6 or click the **Load Driver** button on the **Disk Configuration** page of Windows Setup.
- In unattended installations, you can use the `Microsoft-Windows-PnpCustomizationsWinPE` or `Microsoft-Windows-PnpCustomizationsNonWinPE` component in an answer file to specify a driver path. For more information about how to automate your installation, see [Automate Windows Setup](#).

## **Hardware considerations: Encrypted Hard Drives (e-Drives)**

We added support for Encrypted Hard Drive Devices (also known as E-Drives) in Windows 8, Windows Server 2012, and Windows PE 4.0.

To install a previous version of Windows (examples: Windows 7 or Windows Vista) to an Encrypted Hard Drive

Device, use Windows PE 4.0 or later.

For more information, see [Encrypted Hard Drive Device Guide](#).

## Related topics

[WinPE: Boot in UEFI or legacy BIOS mode](#)

[Windows Setup Scenarios and Best Practices](#)

[Windows Setup Installation Process](#)

[Windows Setup Automation Overview](#)

[Audit Mode Overview](#)

[Windows Setup Configuration Passes](#)

# Add languages to Windows Setup

1/18/2019 • 4 minutes to read

To support multiple languages with Windows Setup:

- Add languages to the Windows Setup image (boot.wim index:2) to help technicians or users install Windows images through Windows Setup.
- When you add languages to a Windows image (install.wim), add a list of languages (lang.ini) so that during installation, the technician or user can select which language is installed as the default language.

You can include a different set of languages for your Windows images and Windows Setup.

## Prerequisites

- A technician computer that has the Windows Assessment and Deployment Kit (Windows ADK) installed with the **Deployment tools** option, including the **WinPE add-on**.

**CopyPE** and **MakeWinPEMedia** are installed when you choose the **Deployment tools** and **Windows Preinstallation Environment** options when [installing the ADK](#). If you're using the ADK for Windows 10, version 1809, Windows PE is a separate add-on that you download and install after you install the ADK. You can [download the WinPE add-on from the Download Center](#).

- Windows installation media for all languages that you are creating media.
- The Windows language pack ISO.

## Step 1. Copy installation files to a working folder

On your technician PC, copy the Windows installation files from the Windows installation media to a local directory.

If you are creating media for use with a customized image, you must use the Windows media that corresponds to the version of your customized image. For example, if you are building a custom Windows 10 Setup image, you must use the original Windows 10 product media.

```
md C:\my_distribution  
xcopy /E D: C:\my_distribution
```

Where *D*: is the location of the Windows installation media.

## Step 2. Add languages to the default Windows Setup boot image (index:2)

1. Click **Start**, and type **deployment**. Right-click **Deployment and Imaging Tools Environment** and then select **Run as administrator**.
2. Mount the second image (index 2) in Boot.wim.

```
md C:\mount\boot  
Dism /mount-image /imagefile:C:\my_distribution\sources\boot.wim /index:2 /mountdir:C:\Mount\boot
```

3. Add language packs from the Windows ADK into your mounted image using **Dism /Add-Package** for each language you want to support. Add *lp.cab*, *WinPE-setup\_<language>.cab*. For Windows 10, also add *WinPE-Setup-client\_<language>.cab*. For Windows Server, also add *WinPE-Setup-server\_<language>.cab*

Windows PE language packs are available in the Windows ADK.

#### For Windows 10:

```
Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\lp.cab"

Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-Setup_fr-fr.cab"

Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-Setup-Client_fr-fr.cab"
```

#### For Windows Server:

```
Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\lp.cab"

Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-Setup_fr-fr.cab"

Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\fr-fr\WinPE-Setup-Server_fr-fr.cab"
```

4. For Japanese (ja-JP), Korean (ko-KR), and Chinese (zh-HK, zh-CN, zh-TW), add font support to your image. For example, to add Japanese font support:

```
Dism /image:C:\mount\boot /add-package /packagepath:"C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Windows Preinstallation Environment\amd64\WinPE_OCs\WinPE-FontSupport-JA-JP.cab"
```

5. Change the Windows Setup default language:

```
Dism /image:C:\mount\boot /Set-SetupUILang:fr-FR /distribution:C:\my_distribution
```

For more information about specifying different international settings, see [DISM Languages and International Servicing Command-Line Options](#).

## Step 3: Add localized Windows Setup resources to the Windows distribution

In this step you copy the language-specific Setup resources from each language-specific Windows distribution to the Sources folder in your Windows distribution. For example, mount the fr-FR Windows installation media as drive E: and copy the Fr-FR sources folder to your Windows distribution.

Note, language interface packs (LIP) are not supported on Windows Setup.

- Copy the localized Windows Setup files to your Windows distribution.

```
xcopy E:\sources\fr-fr C:\my_distribution\sources\fr-fr /cherkyi
```

Where *E*: is the location of the Windows installation media that contains the localized Windows Setup resources.

## Step 4. Add language packs to the Windows image

To modify the languages in the Windows image, see [Add Language Packs to Windows](#).

After you do this, update the list of available Windows languages (Lang.ini):

1. Recreate the Lang.ini file to reflect the additional languages.

```
Dism /image:C:\mount\windows /gen-langINI /distribution:C:\my_distribution
```

2. Copy the lang.ini file in the Windows distribution to the boot folder.

```
Xcopy C:\my_distribution\sources\lang.ini C:\mount\boot\sources\lang.ini
```

## Step 6: Commit the changes to the Windows images

```
...
Dism /unmount-image /mountdir:C:\mount\boot /commit
Dism /unmount-image /mountdir:C:\mount\windows /commit
...
```

## Step 7: Create the media

### USB flash drive

Use **MakeWinPEMedia** with the `/UFD` option to format and install Windows PE to the USB flash drive, specifying the USB key's drive letter:

```
...
MakeWinPEMedia /UFD C:\my_distribution F:
...
```

To learn more, see [WinPE: Create USB Bootable drive](#).

### DVD or CD

1. Create a [boot order file](#). This is required for DVD or CD images larger than 4.5 GB to make sure that boot files are located at the beginning of the image.

For example, copy this into a text file called C:\Temp\BootOrder.txt:

```
boot\bcd
boot\boot.sdi
boot\bootfix.bin
boot\bootsect.exe
boot\etfsboot.com
boot\memtest.efi
boot\memtest.exe
boot\en-us\bootsect.exe.mui
boot\fonts\chs_boot.ttf
boot\fonts\cht_boot.ttf
boot\fonts\jpn_boot.ttf
boot\fonts\kor_boot.ttf
boot\fonts\wgl4_boot.ttf
sources\boot.wim
```

2. Create the ISO file, using the boot order file:

```
Oscdimg -m -n -yo C:\temp\BootOrder.txt -bC:\winpe_amd64\Efisys.bin C:\winpe_amd64\winpeamd64.iso
```

3. Right-click the ISO file and select **Burn disc image**.

## Related topics

[Windows Setup Technical Reference](#)

[DISM Image Management Command-Line Options](#)

[DISM Windows PE Servicing Command-Line Options](#)

[Oscdimg Command-Line Options](#)

[WinPE: Mount and Customize](#)

[WinPE: Install on a Hard Drive \(Flat Boot or Non-RAM\)](#)

# Windows Setup Scenarios and Best Practices

1/18/2019 • 5 minutes to read

Windows Setup installs the Windows operating system. Windows Setup uses a technology called Image-based Setup (IBS) that provides a single, unified process with which all customers can install Windows. IBS performs clean installations and upgrades of Windows and is used in both client and server installations. Windows Setup also enables you to customize Windows during installation by using Setup answer file settings.

In this topic:

- [Common Usage Scenarios](#)
- [Windows Setup Best Practices](#)
- [Windows Setup Limitations](#)

## Common Usage Scenarios

Common installation scenarios include performing clean installations, upgrades, and unattended installations.

### Custom Installations

The most common scenario for Windows Setup is performing a custom installation. In this scenario, you install Windows onto a computer that does not have an operating system, or has a previous version of Windows. This scenario consists of the following stages:

1. Run Setup.exe from your Windows product DVD or network share.
2. Select the **Custom** installation type.
3. If you are installing from a previous installation of Windows, Windows Setup creates a local boot directory and copies all of the required Windows Setup files to this directory.
4. Windows Setup reboots, installs and configures Windows components, and, after installation is complete, launches Windows Welcome.

Custom installations do not migrate any settings or preferences from previously installed versions of Windows. Files from previous Windows versions are copied to a \Windows.old directory. All data from the Windows installation including the Users, Program Files, and Windows directories are saved to this directory.

### Upgrades

Windows Setup can also perform upgrades from a supported operating system.

This scenario includes the following stages:

1. Run Setup.exe on the previous version of Windows.
2. Select the **Upgrade** installation type. Windows Setup upgrades the system and protects your files, settings, and preferences during the installation process.
3. Windows Setup reboots and restores your protected files, settings, and preferences. Windows Setup then launches Windows Welcome.

### Notes

- Upgrades are used to upgrade a single computer to Windows 10. Upgrades also support migrating user data to a new system.

- Windows Setup supports upgrading to an image that has the latest quality updates injected, but does not support upgrading to an image that's been customized to add additional apps and/or settings changes.

## Automated Installations

Automated installations enable you to customize a Windows installation and remove the need for a user to interact with Windows Setup. By using Windows System Image Manager (Windows SIM) or the Component Platform Interface (CPI) APIs, you can create one or more customized Windows installations that can then be deployed across many different hardware configurations.

The automated installation, also called an unattended installation, scenario includes the following stages:

1. Use Windows SIM or the CPI APIs to create an unattended installation answer file, typically called Unattend.xml. This answer file contains all of the settings that you configure in the Windows image. For more information, see [Windows System Image Manager How-to Topics](#).
2. From Windows PE, a previous version of Windows, or another preinstallation environment, run Setup.exe with the explicit path to the answer file. If you do not include the path to the answer file, Setup.exe searches for a valid answer file in several specific locations. For more information, see [Windows Setup Command-Line Options](#).
3. Windows Setup then installs the operating system and configures all settings listed in the answer file. Additional applications, device drivers, and updates can also be installed during Windows Setup. After the operating system is installed, Setup launches Windows Welcome.

## Windows Setup Best Practices

The following section describes some of the best practices to use with Windows Setup.

- **Verify that there is sufficient space for Windows Setup temporary files.** If you run setup from a previous version of Windows, verify that there is sufficient space on the disk for temporary Windows Setup files. The space that is required may vary, but it can be up to 500 megabytes (MB).
- **Previous Windows installations are moved to a Windows.old folder.** As a best practice, you should back up your data before you upgrade. If you install Windows over a previous Windows installation, all previous Windows files and directories are moved to a Windows.old folder, including the contents of the Users, Program Files, and Windows directories. You can access your data in the Windows.old folder after Windows Setup completes. If you have additional folders not in the Users, Program Files, or Windows directories, those folders are not moved. For example, if you have a folder that is named C:\Drivers, that folder will not be moved to the Windows.old folder.
- **Review the Windows Setup log files.** If you experience problems during Windows Setup, review the log files in %WINDIR%\panther. You will be able to identify and troubleshoot many issues by reviewing the installation log files. For more information, see [Deployment Troubleshooting and Log Files](#) and [Windows Setup Log Files and Event Logs](#).

## Windows Setup Limitations

The following sections describe some of the limitations of Windows Setup. Review this section before you run Windows Setup.

- **Enable UEFI-compatibility mode to install to an UEFI-based computer.** On some UEFI computers, you cannot install Windows in BIOS-compatibility mode. You may need to switch to UEFI-compatibility mode.
- **Applications might require a consistent drive letter.** If you install custom applications to your Windows image, install Windows to the same drive letter on the destination computer, because some

applications require a consistent drive letter. Uninstallation, servicing, and repair scenarios might not function appropriately if the drive letter of the system does not match the drive letter specified in the application. This limitation applies to both the Deployment Image Servicing and Management (DISM) tool and Windows Setup.

- **Deploying multiple images to multiple partitions.** If you capture and deploy multiple images on multiple partitions, the following requirements must be fulfilled:
  - The partition structure, bus location, and number of disks must be identical on the reference and destination computers.
  - The partition types (primary, extended, or logical) must match. The active partition on the reference computer must match that of the destination computer.
- **Installing Custom .wim files requires a description value in the .wim file.** When you create a custom .wim file, Windows Setup requires that you always include a description value. If a .wim file does not include a description value, the image may not install correctly. You can provide a description value when you use the **dism** command with the **/capture-image** option. If you install a .wim file that does not have a description value, recapture the image and provide a valid description value. For more information, see the [DISM - Deployment Image Servicing and Management Technical Reference for Windows](#).

**Note** For Windows Preinstallation Environment (Windows PE), the version of boot files must match the computer architecture. An x64 UEFI computer can only boot by using Windows PE x64 boot files. An x86 computer can only boot by using Windows PE x86 boot files. This is different from legacy BIOS. In legacy BIOS, an x64 computer can boot by using x86 boot files.

## Related topics

[Windows Setup Installation Process](#)

[Windows Setup Automation Overview](#)

[Audit Mode Overview](#)

[Windows Setup Configuration Passes](#)

[Windows Setup Supported Platforms and Cross-Platform Deployments](#)

# Windows Setup Command-Line Options

6/19/2019 • 14 minutes to read

The following command-line options are available for Windows Setup. Beginning with Windows 10, version 1607, you can use a setupconfig file as an alternative to passing parameters to Windows Setup on a command line. For more information, see [Windows Setup Automation Overview](#).

## setup.exe

The following table lists Setup command-line options:

OPTION	DESCRIPTION
<b>/1394Debug:&lt;channel&gt; [BaudRate:&lt;baudrate&gt;]</b>	Enables kernel debugging over an IEEE 1394 (FireWire) port while Windows is running and during the <a href="#">windowsPE</a> configuration pass of Windows Setup. <channel> specifies the debugging channel. The default value for <channel> is <b>1</b> . [baudrate:<baudrate>] specifies the baud to use when Windows transfers data during debugging. The default setting is <b>19200</b> . You can also set the <baudrate> setting to <b>57600</b> or <b>115200</b> . For example:  <code>Setup /1394debug:1 /baudrate:115200</code>
<b>/AddBootMgrLast</b>	Instructs Windows Setup to add the Windows Boot Manager as the last entry in the UEFI firmware boot order. This option is only supported on UEFI PCs running Windows PE 4.0 or later.

OPTION	DESCRIPTION
<b>/Auto {Clean   DataOnly   Upgrade}</b>	<p>Performs an automated upgrade to Windows 10 or Windows 8.1 volume license editions only. When /auto is used, an unattend file cannot be used. When /auto is used, Windows Setup consumes ei.cfg, and checks compatibility issues before starting the installation. If ei.cfg is malformed, setup exits silently and logs an exit code.</p> <p><b>Clean:</b> Performs a clean install of Windows.</p> <p><b>DataOnly:</b> Performs an upgrade of Windows, saving only data (and not apps.) If the data-only installation option is not available due to compatibility checks, Windows Setup will exit silently and log an exit code.</p> <p><b>Upgrade:</b> Performs an upgrade of Windows saving apps and data. If the upgrade installation option is not available, or the user needs to resolve an app compatibility issue, Windows Setup will exit silently and log an exit code.</p> <p><b>Setup.exe exit codes:</b> See <a href="#">table below</a>.</p> <p><b>/noautoexit:</b> Not used in Windows 10. In Windows 8.1, if an error is found, Windows Setup does not exit, but instead stops and stays on the setup screen until the user addresses the issue. The installation from that point on is attended.</p> <p><b>/performDU:</b> Not used in Windows 10. In Windows 8.1, Windows Setup checks for Dynamic Updates for Windows Setup.</p> <p>Examples:</p> <pre>Setup /auto clean Setup /auto dataonly Setup /auto upgrade</pre>
<b>/BitLocker {AlwaysSuspend   TryKeepActive   ForceKeepActive}</b>	<p>Specifies the BitLocker status during upgrades.</p> <p><b>AlwaysSuspend:</b> BitLocker is always suspended during an upgrade.</p> <p><b>TryKeepActive:</b> Attempts an upgrade without suspending BitLocker. If the upgrade fails, Windows Setup will suspend BitLocker and complete the upgrade. This is the default behavior if the <b>/bitlocker</b> option is not specified.</p> <p><b>ForceKeepActive:</b> Enables upgrading without suspending BitLocker. If the upgrade can't be completed because BitLocker is active, the upgrade will fail.</p>
<b>/BusParams:&lt;bus.device.function&gt;</b>	<p>Specifies the PCI address of a 1394, USB, or NET debug port. The bus, device, and function numbers must be in decimal format. Example:</p> <pre>Setup /busparams:0.29.7</pre> <p>For more info, see <a href="#">Setting Up Kernel Debugging with USB 2.0</a>.</p>

OPTION	DESCRIPTION
<b>/CompactOS {Enable / Disable}</b>	<p>Specifies whether to use the Compact OS feature to save hard drive space. By default, Windows Setup determines whether to use this feature automatically.</p> <p><b>Enable:</b> Setup installs Windows using compressed system files.</p> <p><b>Disable:</b> Setup installs Windows using uncompressed system files</p> <p>To learn more about Compact OS, see <a href="#">Compact OS, single-instancing, and image optimization</a>.</p> <p><code>Setup /compactos enable</code></p>
<b>/Compat {IgnoreWarning / ScanOnly}</b>	<p><b>IgnoreWarning:</b> Setup completes installation, ignoring any dismissible compatibility messages.</p> <p><b>ScanOnly:</b> Windows Setup runs through compatibility scans, and then exits (without completing the installation) with an exit code to indicate if any compatibility concerns are present. Setup will return 0xC1900210 if no concerns are found. Setup will return 0xC1900208 if compatibility concerns are found.</p> <p>Example:  <code>Setup /compat IgnoreWarning</code></p> <p>If you launch Setup with <b>/Compat ScanOnly</b>:</p> <ul style="list-style-type: none"> <li>• If it does not find any compat issue, it will return MOSETUP_E_COMPAT_SCANONLY (0xC1900210)</li> <li>• If it finds Actionable compat issues, like Apps, it will return MOSETUP_E_COMPAT_INSTALLREQ_BLOCK (0xC1900208)</li> <li>• If it finds that the Mig-Choice selected is not available, it will return MOSETUP_E_COMPAT_MIGCHOICE_BLOCK (0xC1900204)</li> <li>• If it finds that machine is not eligible for Windows 10, it will return MOSETUP_E_COMPAT_SYSREQ_BLOCK (0xC1900200)</li> <li>• If it finds that machine does not have enough free space to install, it will return MOSETUP_E_INSTALLDISKSPACE_BLOCK (0xC190020E)</li> </ul> <p>This command works with other switches. For example, to run Setup in the background without any UI:</p> <p><code>Setup /Auto Upgrade /Quiet /Compat ScanOnly</code></p> <p>To ignore common disclaimers in the UI, for example, language changes:</p> <p><code>Setup /Auto Upgrade /Quiet /Compat ScanOnly /Compat IgnoreWarning</code></p> <p>Most of the time, an Admin would like to look at the compat XML if Setup found compat issues. For that the admin can even use copy logs flag to collect Setup logs:</p> <p><code>Setup /Auto Upgrade /Quiet /Compat ScanOnly /Compat IgnoreWarning /CopyLogs C:\Temp\Logfiles.log</code></p> <p>This setting is new for Windows 10.</p>

OPTION	DESCRIPTION
<b>/CopyLogs&lt;location&gt;</b>	<p>Setup will copy or upload logs(compressed) upon failure to the specified location (assuming machine/user has permission and network access to location). Accepted parameters are local file paths and UNC network paths.</p> <p>Note: This runs in the system context, so it may not have permissions to copy to locations that require user permissions.</p> <p>Example:  <code>Setup /copylogs \\server\share\</code></p>
<b>/Debug:&lt;port&gt; [BaudRate:&lt;baudrate&gt;]</b>	<p>Enables kernel debugging over a communications (COM) port when Windows is running, and during the <a href="#">windowsPE</a> configuration pass of Windows Setup.</p> <p>&lt;port&gt; specifies the debugging port. The default value for &lt;port&gt; is 1.</p> <p>[baudrate:&lt;baudrate&gt;] specifies the baud to use when Windows transfers data during debugging. The default setting is <b>19200</b>. You can also set the &lt;baudrate&gt; setting to <b>57600</b> or <b>115200</b>. For example:</p> <p><code>Setup /debug:1 /baudrate:115200</code></p>
<b>/DiagnosticPrompt {enable   disable}</b>	<p>Specifies that the Command Prompt is available during Windows Setup.</p> <p><b>Enable:</b> The Command Prompt can be accessed by pressing Shift+F10 during Windows setup.</p> <p><b>Disable:</b> The Command Prompt is not available during Windows setup. The Command Prompt will not be available while offline and OOBE phases are running. This is the default setting.</p> <p>Example:  <code>setup /DiagnosticPrompt enable</code>  This setting is new for Windows 10, Version 1703.</p>
<b>/DynamicUpdate {enable   disable}</b>	<p>Specifies whether setup will perform Dynamic Update operations (search, download, and install updates).</p> <p>Example:  <code>setup /auto upgrade /DynamicUpdate disable</code></p>

OPTION	DESCRIPTION
<p><b>/EMSPort:</b> {COM1   COM2   off} [/emsbaudrate:&lt;baudrate&gt;]</p>	<p>Enables or disables Emergency Management Services (EMS) during Windows Setup and after the server operating system has been installed. The following arguments are used to specify the behavior of EMS during Windows Setup.</p> <p><b>COM1</b> enables EMS over COM1. Supported for x86 systems only.</p> <p><b>COM2</b> enables EMS over COM2. Supported for x86 systems only.</p> <p><b>usebiossettings</b> uses the setting that the BIOS specifies. For x86 systems, Windows uses the value from the Serial Port Console Redirection (SPCR) table. If no SPCR table or EFI console device path is specified in the BIOS, Windows disables <b>usebiossettings</b>.</p> <p><b>off</b> disables EMS. If EMS is disabled in Windows Setup, you can later enable EMS by modifying the boot settings.</p> <p>[/emsbaudrate:&lt;baudrate&gt;] specifies the baud to use when Windows transfers data during debugging. The default is <b>19200</b>. You can also set the &lt;baudrate&gt; setting to <b>57600</b> or <b>115200</b>. For example:</p> <p style="color: red;">Setup /emspor:COM1 /emsbaudrate:115200</p>
<p><b>/InstallDrivers</b>&lt;location&gt;</p>	<p>Adds .inf-style drivers to the new Windows 10 installation. The driver .inf can be in a folder within the specified location. The command will recurse through the specified location.</p> <p>Accepted parameters are a local file path or UNC network path to a folder that contains .inf files. Example:</p> <p style="color: red;">setup.exe /auto upgrade /installdrivers C:\Fabrikam\drivers /noreboot</p> <p>This setting is new for Windows 10.</p>
<p><b>/InstallFrom</b>&lt;path&gt;</p>	<p>Specifies a different Install.wim file to use during Windows Setup. This enables you to use a single preinstallation environment to install multiple versions of Windows images. For example, you can use a 32-bit version of Windows Setup to deploy a 64-bit Windows image. You can also use an answer file for cross-platform deployments. For more information, see "Creating a WIM for Multiple Architecture Types" in <a href="#">Windows Setup Supported Platforms and Cross-Platform Deployments</a>.</p> <p>&lt;path&gt; specifies the path of the .wim file to install. For example:</p> <p style="color: red;">Setup /installfrom D:\custom.wim</p> <p>Can also be used with split image files (.swm). Select the first split image file in the series, for example:</p> <p style="color: red;">Setup /installfrom D:\install.swm</p>

OPTION	DESCRIPTION
<b>/InstallLangPacks</b> <location>	<p>Adds language packs (lp.cab) to the new Windows 10 installation.</p> <p>The language packs can be in a folder within the specified location. The command installs all lp.cab files and language capabilities such as text-to-speech recognition, in the folder and subfolders at the specified location.</p> <p>Accepted parameters are a local file path or UNC network path to a folder that contains .inf files.</p> <pre>setup /auto upgrade /installlangpacks C:\Fabrikam\Languages\French /noreboot</pre> <p>This setting is new for Windows 10.</p>
<b>/m:&lt;folder_name&gt;</b>	<p>Instructs Setup to copy alternate files from an alternate location. This option instructs Setup to look in the alternate location first, and, if files are present, to use them instead of the files from the default location.</p> <p>&lt;folder_name&gt; specifies the name and the location of the folder that contains the replacement files and can be any local drive location. UNC paths are not supported. You must know where the files will be installed on the Windows installation. All the additional files must be copied to an \$OEM\$ folder in your installation sources or in the &lt;folder_name&gt;. The \$OEM\$ structure provides a representation of the destination installation disk. For example:</p> <p>\$OEM\$\\$1 maps to %SYSTEMDRIVE%, which could be drive C.</p> <p>\$OEM\$\\$\$ maps to %WINDIR%, which could be C:\windows.</p> <p>\$OEM\$\\$progs maps to the program files directory.</p> <p>\$OEM\$\\$docs maps to the user's My Documents folder.</p> <p>For example, to copy an updated C:\Program Files\Messenger\Msmsgs.exe file into the Windows installation, create the following folder structure on the Pro\Sources\\$\OEM\$\\$\\$Progs\Messenger\Msmsgs.exe installation source by using the <b>Setup</b> command:</p> <pre>Pro\sources\setup.exe /m</pre> <p>If you replace a file that Windows file protection protects, you must also copy the updated file to the local sources to be installed with Windows. For example, you may copy the file to the C:\Windows\i386 folder. The file name must be the same as the name that is used in Windows Setup. For example, add the following file and folder structure to your \$OEM\$ directory:</p> <pre>Pro\sources\\$\OEM\$\\$\\$i386\msmsgs.exe</pre> <p>If you use files that are not on an installation share, you must specify the folder name. In this example the &lt;folder_name&gt; is C:\additional_files:</p> <pre>Setup /m:C:\additional_files</pre> <p>where C:\additional_files is your customized \$OEM\$ directory. For example:</p> <pre>C:\additional_files\\$\\$i386\msmsgs.exe</pre> <p>If you change resources in your replacement files, you must add the updated Multilanguage User Interface (MUI) files to the installation.</p>

OPTION	DESCRIPTION
<b>/MigNEO Disable</b>	Tells Windows Setup to perform an upgrade of Windows without additional offline phase optimizations. This option is available in Windows 10, version 1803 and later.
<b>/MigrateDrivers {all   none}</b>	<p>Instructs Setup whether to migrate the drivers from the existing installation during the upgrade. You can specify <b>All</b> or <b>None</b>. By default, Setup decides which is best for each individual driver based on the install choice.</p> <p>You can use this switch with <b>/installdrivers</b>, though it's not required.</p> <pre>Setup /auto upgrade /migratedrivers all Setup /auto upgrade /migratedrivers none /installdrivers N:\NewDrivers</pre>
<b>/NetDebug</b> :hostip=<w.x.y.z>,port=<n>,key= <q.r.s.t> [,{nodhcp}][,{busparams=n.o.p}]	<p>Enables kernel debugging over the network.</p> <p>Use hostip to identify the IP address of the host computer.</p> <p>Use port to identify the port. The default start port is 49152, and the default end port is 65535.</p> <p>Use key to provide a password to set up a secure connection.</p> <p>Use nodhcp to avoid using a DHCP connection. (optional)</p> <p>Use busparams to select the bus number, device number, and function number of an adapter for a specific PCI bus device. (optional)</p> <p>Examples:</p> <pre>setup /netdebug:hostip=10.125.4.86,port=50000,key= 0.0.0.0 setup /netdebug:hostip=10.125.4.86,port=50000, key=abcdefg.123.hijklmnop.456,nodhcp setup /netdebug:hostip=10.1.4.8,port=50000, key=dont.use.previous.keys,busparams=1.5.0</pre> <p>For details, see <a href="#">Setting Up Kernel-Mode Debugging over a Network Cable Manually</a>.</p>
<b>/NoReboot</b>	<p>Instructs Windows Setup not to restart the computer after the down-level phase of Windows Setup completes. The <b>/noreboot</b> option enables you to execute additional commands before Windows restarts. This option suppresses only the first reboot. The option does not suppress subsequent reboots. For example:</p> <pre>Setup /noreboot</pre>
<b>/PKey&lt;product key&gt;</b>	<p>Supplies Setup with the specific product key. Example:</p> <pre>setup.exe /auto upgrade /pkey xxxxx-xxxxx- xxxxx-xxxxx-xxxxx</pre> <p>This setting is new for Windows 10.</p>

OPTION	DESCRIPTION
<b>/Priority Normal</b>	<p>Tells Windows Setup to increase the thread priority from low to high for feature updates through Windows Update. This option is available in Windows 10, version 1709 and later <b>Note:</b> Media based installations already run at normal priority.</p>
<b>/PostOOBE&lt;location&gt; [\setupcomplete.cmd]</b>	<p>After Setup is complete, run a script. Accepted parameters are a local file path or UNC network path to a file named setupcomplete.cmd or to a folder that contains setupcomplete.cmd.</p> <pre>setup.exe /auto upgrade /postoobe c:\Fabrikam\setupcomplete.cmd</pre> <p>Path to folder that contains a script with the name: <b>setupcomplete.cmd:</b> Copies setupcomplete.cmd to \$Windows.~BT to be run after OOBE.</p> <pre>setup.exe /auto upgrade /postoobe c:\Fabrikam</pre> <p>This setting is new for Windows 10.</p>
<b>/PostRollback&lt;location&gt; [\setuprollback.cmd]</b> <b>/postrollbackcontext {system / user}</b>	<p>If the feature update fails to install and rolls back the changes, or if the user chooses to uninstall the feature update and go back to a previous version of Windows, run a script.</p> <p>Accepted parameters are a local file path or UNC network path to a file named setuprollback.cmd, or to a folder that contains setuprollback.cmd.</p> <p>By default, updates from media run <b>setuprollback.cmd</b> in <b>user</b> context, which requires the first user who logs in post-upgrade to have administrator rights. For updates from Windows Update, <b>setuprollback.cmd</b> runs in <b>system</b> context, regardless of the rights of the first logged-in user. The <b>postrollbackcontext</b> option allows you to specify whether the script runs in the context of the System account or the account of the signed in user.</p> <pre>setup.exe /auto upgrade /postrollback c:\Fabrikam\setuprollback.cmd</pre> <p>Path to folder that contains a script with the name: <b>setuprollback.cmd:</b> Copies setuprollback.cmd to \$Windows.~BT to be run after OOBE.</p> <pre>setup.exe /auto upgrade /postrollback \server\share setup.exe /postrollback C:\Fabrikam\setuprollback.cmd</pre> <p><b>/postrollbackcontext user</b></p> <p><b>/postrollbackcontext</b> is new for Windows 10, version 1803.</p>
<b>/Quiet</b>	<p>This will suppress any Setup user experience including the rollback user experience. Example:</p> <pre>setup /auto upgrade /quiet</pre> <p>This setting is new for Windows 10.</p>

OPTION	DESCRIPTION
<b>/ReflectDrivers</b> <location>	<p>Specifies the path to a folder that contains encryption drivers for a computer that has third-party encryption enabled.</p> <p><b>Setup /ReflectDrivers</b></p> <p>This setting is new for Windows 10, version 1607. Make sure that &lt;folder_path&gt; contains only a minimal set of encryption drivers. Having more drivers than necessary in &lt;folder_path&gt; can negatively impact upgrade scenarios.</p>
<b>/ResizeRecoveryPartition</b> {Enable / Disable}	<p>Specifies whether it's OK to resize the existing Windows Recovery Environment (Windows RE) partition or create a new one during installation.</p> <p><b>Enable:</b> During installation, Windows can resize the existing Windows RE tools partition or create a new one if needed.</p> <p><b>Disable:</b> Windows does not resize the existing Windows RE tools partition or create a new one during installation. To learn more about Windows RE partitions, see <a href="#">UEFI/GPT-based hard drive partitions</a> and <a href="#">BIOS/MBR-based hard drive partitions</a>.</p> <p><b>Setup /resizerecoverypartition disable</b></p>
<b>/ShowOOBE</b> {full / none}	<p><b>full:</b> Requires the user to interactively complete the out of box experience (OOBE).</p> <p><b>none:</b> Skips OOBE and selects the default settings.</p> <p>Example: <b>setup.exe /auto upgrade /showoobe full</b></p> <p>This setting is new for Windows 10.</p>
<b>/Telemetry</b> {Enable / Disable}	<p>Specifies whether Windows Setup should capture and report installation data.</p> <p><b>Enable:</b> Setup captures and reports installation data.</p> <p><b>Disable:</b> Setup does not capture and report installation data.</p> <p><b>Setup /telemetry disable</b></p>
<b>/TempDrive</b> <drive_letter>	<p>Instructs Windows Setup to put temporary installation files on the specified partition. For an upgrade, the <b>/tempdrive</b> option affects only the placement of temporary files. The operating system is upgraded in the partition from which you run the Setup.exe file.</p> <p>The <b>/tempdrive</b> parameter is available in Windows 10, version 1607, but it is not available in earlier versions of Windows 10.</p> <p>&lt;drive_letter&gt; specifies the partition to copy installation files to during Windows Setup. For example: <b>Setup /tempdrive H</b></p>

OPTION	DESCRIPTION
<b>/Unattend:&lt;answer_file&gt;</b>	<p>Enables you to use an answer file with Windows Setup. This is known as an unattended installation. You must specify a value for &lt;answer_file&gt;. Windows Setup applies the values in the answer file during installation.</p> <p>&lt;answer_file&gt; specifies the file path and file name of the unattended Windows Setup answer file.</p> <p>When /Unattend is used, /Auto cannot be used.</p> <p>Setup /unattend:\server\share\unattend.xml</p>
<b>/Uninstall {enable / disable}</b>	<p>Determines whether Windows will include controls that allow the user to go back to the previous operating system.</p> <p>This setting is new for Windows 10.</p> <p>Setup /uninstall disable</p>
<b>/USBDebug:&lt;hostname&gt;</b>	<p>Sets up debugging on a USB port. Debug data is effective on the next reboot.</p> <p>&lt;hostname&gt; specifies the name of the computer to debug. For example:</p> <p>Setup /usbdebug:testmachine01</p>
<b>/WDSDiscover</b>	<p>Specifies that the Windows Deployment Services (WDS) client should be in discover mode.</p> <p>If you do not specify /wdsserver with this option, WDS searches for a server. For example, to start the WDS client in this dynamic discover mode, run the following command:</p> <p>Setup /wds /wdsdiscover</p>
<b>/WDSServer:&lt;servername&gt;</b>	<p>Specifies the name of the Windows Deployment Services server that the client should connect to.</p> <p>To use this setting, you must also use the /wdsdiscover option.</p> <p>&lt;servername&gt; can be an IP address, a NetBIOS name, or a fully qualified domain name (FQDN). For example, to start the Windows Deployment Services client in this static discover mode, run the following command:</p> <p>Setup /wds /wdsdiscover /wdsserver:MyWDSServer</p>

## Setup.exe exit codes

EXIT CODE NAME	EXIT CODE	CAUSE
CONX_SETUP_EXITCODE_CONTINUE_R_EBOOT	0x3	This upgrade was successful.
CONX_SETUP_EXITCODE_RESUME_AT_COMPAT_REPORT	0x5	The compatibility check detected issues that require resolution before the upgrade can continue.

EXIT CODE NAME	EXIT CODE	CAUSE
CONX_SETUP_EXITCODE_AUTO_INSTALL_FAIL	0x7	The installation option (upgrade or data only) was not available.

## Related topics

[Windows Setup States](#)

[Windows Setup Edition Configuration and Product ID Files \(EI.cfg and PID.txt\)](#)

[Windows Setup Log Files and Event Logs](#)

# Windows Setup States

1/18/2019 • 2 minutes to read

There are several states assigned to a Windows® image during installation. This state information can be used to detect automatically the different states and stages of Windows Setup.

## Windows Setup State Information

The Windows image state is stored in two locations, in the registry and in a file.

- In the registry:

KEY: **HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Setup\State**

TYPE: REG\_SZ

VALUE: *StateName*

- In a file:

FILE: %WINDIR%\Setup\State\State.ini

SECTION: [State]

VALUE: *StateName*

The following table describes the values that exist for *StateName*.

STATE NAME	DESCRIPTION
IMAGE_STATE_COMPLETE	The image has successfully been installed. The <b>specialize</b> and <b>oobeSystem</b> configuration passes are complete. This image is not deployable to a computer that has a different hardware configuration because it is now hardware-dependent. To deploy this image to a computer that has a different hardware configuration, you must run <b>sysprep /generalize</b> .
IMAGE_STATE_UNDEPLOYABLE	This is the default state for an image in a given phase of Windows Setup that is not yet complete. If a process queries the IMAGE_STATE value and IMG_UNDEPLOYABLE is returned, the image is in one of the following states: <ul style="list-style-type: none"><li>• Setup is currently running and has not fully completed the phase. Once a given phase is complete, the IMAGE_STATE will be set to an appropriate completion value.</li><li>• If queried online when Setup is not running, there was a failure when completing a Setup phase. This image must be reinstalled.</li><li>• If queried offline, the image did not finish a phase and will never be deployable.</li></ul>

STATE NAME	DESCRIPTION
IMAGE_STATE_GENERALIZE_RESEAL_TO_OOBE	The image has successfully completed the <b>generalize</b> configuration pass and will continue into <b>OOBESystem</b> configuration pass when Setup is initiated.
IMAGE_STATE_GENERALIZE_RESEAL_TO_AUDIT	The image has successfully completed the <b>generalize</b> configuration pass and will continue into audit mode when Setup is initiated.
IMAGE_STATE_SPECIALIZE_RESEAL_TO_OOBE	The image has successfully completed the <b>specialize</b> pass and will continue into <b>OOBESystem</b> configuration pass when Setup is initiated.
IMAGE_STATE_SPECIALIZE_RESEAL_TO_AUDIT	The image has successfully completed the <b>specialize</b> configuration pass and will continue into audit mode when Setup is initiated.

The following examples show how to access state information.

- To access state information from the registry:

```
C:\>reg query HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Setup\State /v ImageState
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Setup\State
ImageState      REG_SZ      IMAGE_STATE_SPECIALIZE_RESEAL_TO_OOBE
```

- To access state information from a file:

```
C:\>type %windir%\Setup\State\State.ini
[State]
ImageState="IMAGE_STATE_SPECIALIZE_RESEAL_TO_OOBE"
```

## Related topics

[Windows Setup Command-Line Options](#)

[Windows Setup Edition Configuration and Product ID Files \(EI.cfg and PID.txt\)](#)

[Windows Setup Log Files and Event Logs](#)

# Windows Setup Edition Configuration and Product ID Files (EI.cfg and PID.txt)

1/18/2019 • 2 minutes to read

The edition configuration (EI.cfg) file and the product ID (PID.txt) file are optional configuration files that you can use to specify the Windows® product key and the Windows edition during Windows installation. You can use these files to automate the product-key entry page in Windows Setup instead of using an answer file. If you use an EI.cfg file to differentiate volume license media, but you do not include a PID.txt file, the user receives a prompt for a product key to continue Windows Setup.

You can reuse the product key in the product ID file for multiple installations. The product key in the product ID file is only used to install Windows. This key is not used to activate Windows. For more information, see [Work with Product Keys and Activation](#).

## Using EI.cfg and PID.txt

1. Create these configuration files in a text editor such as Notepad.
2. Save the files into the `\Sources` folder on the installation media. Windows Setup will use these files automatically during installation.
3. Run Windows Setup. Setup uses these files during the Windows PE configuration pass as soon as it is launched.

**Note** An answer file takes precedence over these files. If you use an answer file during installation, Windows Setup ignores the EI.cfg and PID.txt files.

## EI.cfg Format

The EI.cfg file specifies the values for the edition ID, the channel, and the volume license.

The EI.cfg file has the following format:

```
[EditionID]
{Edition ID}
[Channel]
{Channel Type}
[VL]
{Volume License}
```

`{Edition ID}` must be a valid Windows edition ID, for example, "Enterprise". To obtain the current EditionID, use the **Dism /Get-ImageInfo** command or the **Dism /Get-CurrentEdition** command. For more information, see [Take Inventory of an Image or Component Using DISM](#) and [DISM Windows Edition-Servicing Command-Line Options](#).

`{Channel Type}` must be either "OEM" or "Retail"

`{Volume License}` must be either 1, if this is a volume license, or 0, if this is not a volume license. For example:

```
[EditionID]
Enterprise
[Channel]
OEM
[VL]
0
```

## PID.txt Format

The PID.txt file contains the product key for the edition of Windows that you are installing.

The PID.txt file has the following format:

```
[PID]
Value=XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

where XXXXX-XXXXX-XXXXX-XXXXX-XXXXX is the product key.

## Troubleshooting

**"The product key entered does not match any of the Windows images available for installation. Enter a different product key."**: You may need to download a separate version of Windows. OEM versions are only available to OEMs, and volume licenses are only available to MSDN subscribers.

## Related topics

[Work with Product Keys and Activation](#)

[Windows Setup Command-Line Options](#)

[Windows Setup States](#)

# Windows Setup Log Files and Event Logs

1/18/2019 • 2 minutes to read

Windows® Setup creates log files for all actions that occur during installation. If you are experiencing problems installing Windows, consult the log files to troubleshoot the installation.

Windows Setup log files are available in the following directories:

LOG FILE LOCATION	DESCRIPTION
\$windows.~bt\Sources\Panther	Log location before Setup can access the drive.
\$windows.~bt\Sources\Rollback	Log location when Setup rolls back in the event of a fatal error.
%WINDIR%\Panther	Log location of Setup actions after disk configuration.
%WINDIR%\Inf\Setupapi.log	Used to log Plug and Play device installations.
%WINDIR%\Memory.dmp	Location of memory dump from bug checks.
%WINDIR%\Minidump.dmp	Location of log minidumps from bug checks.
%WINDIR%\System32\Sysprep\Panther	Location of Sysprep logs.

## Windows Setup Event Logs

Windows Setup includes the ability to review the Windows Setup performance events in the Windows Event Log viewer. This enables you to more easily review the actions that occurred during Windows Setup and to review the performance statistics for different parts of Windows Setup. You can filter the log so as to view only relevant items that you are interested in. The Windows Setup performance events are saved into a log file that is named Setup.etl, which is available in the %WINDIR%\Panther directory of all installations. To view the logs, you must use the Event Viewer included with the Windows media that corresponds to the version of the customized image that you are building.

To view the logs on a computer that does not include the corresponding kit, you must run a script from the root of the media that installs the Event Trace for Windows (ETW) provider. From the command line, type:

```
Cscript D:\sources\etwproviders\etwproviderinstall.vbs install D:\sources\etwproviders
```

where *D* is the drive letter of the Windows DVD media.

### To view the Windows Setup event logs

1. Start the Event Viewer, expand the Windows Logs node, and then click **System**.

2. In the **Actions** pane, click **Open Saved Log** and then locate the Setup.etl file. By default, this file is available in the %WINDIR%\Panther directory.

3. The log file contents appear in the Event Viewer.

### To Export the log to a file

From the command line, use the **Wevtutil** or **Tracerpt** commands to save the log to an .xml or text file. For information about how to use these tools, see the command-line Help. The following commands show examples of how to use the tools:

```
Wevtutil qe /lf C:\windows\panther\setup.etl
```

-or-

```
Tracerpt /l C:\windows\panther\setup.etl
```

## Related topics

[Windows Setup Command-Line Options](#)

[Windows Setup States](#)

[Windows Setup Edition Configuration and Product ID Files \(EI.cfg and PID.txt\)](#)

# Windows Setup Configuration Passes

1/18/2019 • 2 minutes to read

Configuration passes are used to specify different phases of Windows® Setup. Unattended installation settings can be applied in one or more configuration passes.

## In This Section

The following topics describe the configuration passes used with Windows Setup.

<a href="#">How Configuration Passes Work</a>	A description of the different phases of Windows Setup, and the different configuration passes used to install and configure a Windows installation.
<a href="#">auditSystem</a>	The <b>auditSystem</b> configuration pass is one of the configuration passes used in audit mode.
<a href="#">auditUser</a>	The <b>auditUser</b> configuration pass is one of the configuration passes used in audit mode.
<a href="#">generalize</a>	The <b>generalize</b> configuration pass prepares a Windows image to be deployed across many computers.
<a href="#">offlineServicing</a>	The <b>offlineServicing</b> configuration pass is used to install packages, drivers, and other updates to an offline Windows image.
<a href="#">oobeSystem</a>	The <b>oobeSystem</b> configuration pass, also known as Windows Welcome, can be used to preconfigure user interface pages for an end user.
<a href="#">specialize</a>	The <b>specialize</b> configuration pass customizes a specific Windows installation to a specific computer.
<a href="#">windowsPE</a>	The <b>windowsPE</b> configuration pass is used to configure Windows PE in addition to some aspects of Windows Setup.

## Related topics

[Windows Setup Scenarios and Best Practices](#)

[Windows Setup Installation Process](#)

[Windows Setup Automation Overview](#)

[Audit Mode Overview](#)

## Windows Setup Supported Platforms and Cross-Platform Deployments

# How Configuration Passes Work

1/18/2019 • 11 minutes to read

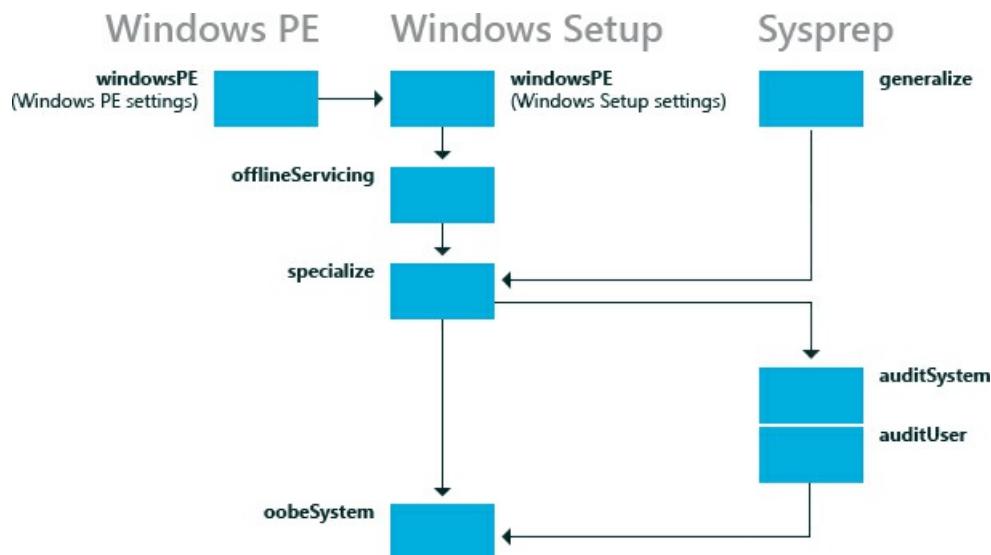
Configuration passes are the phases of a Windows® installation during which you can customize an image. Windows unattended installation settings can be applied in one or more configuration passes, depending on the setting you use. Understanding how and when configuration passes run is very important in developing a Windows deployment strategy.

In this topic:

- [Understanding Configuration Passes](#)
- [Configuring Device Drivers](#)
- [Configuring International Settings](#)
- [Examples](#)

## Understanding Configuration Passes

The following diagram shows the relationship between the configuration passes relative to the different deployment tools.



Not all configuration passes run in a particular installation of Windows. Some configuration passes, such as **auditSystem** and **auditUser**, run only if you boot the computer to audit mode. Most Windows Setup unattend settings can be added to either the **specialize** or the **oobeSystem** configuration pass. The other configuration passes can also be useful in certain situations. The following table describes each of the configuration passes.

CONFIGURATION PASS	DESCRIPTION	CONFIGURATION PASS RUNS WHEN
--------------------	-------------	------------------------------

CONFIGURATION PASS	DESCRIPTION	CONFIGURATION PASS RUNS WHEN
<b>windowsPE</b>	<p>Many aspects of the installation process can be automated during the <b>windowsPE</b> configuration pass. In this pass you can configure:</p> <ul style="list-style-type: none"> <li>• Windows PE options           <p>These options can include specifying the location of the Windows PE log file, which enables networking or a Windows PE page file.</p> </li> <li>• Windows Setup options           <p>These options can include specifying the Windows image to install and configuring a disk on the destination computer.</p> </li> </ul> <p>During this configuration pass, the Windows image is copied to the destination computer after the settings in the <b>windowsPE</b> configuration pass are processed.</p> <p>If your installation of Windows PE requires boot-critical drivers to access the local hard disk drive or a network, use this configuration pass to add drivers to the Windows PE driver store and to reflect the required boot-critical drivers</p>	<p>One of the following occurs:</p> <ul style="list-style-type: none"> <li>• Booting the Windows Setup media</li> <li>• Starting Windows Setup from a previous Windows installation</li> </ul> <p>The Windows PE options are applied only when you are running Windows Setup from a Windows PE environment. The Windows Setup options are applied when it runs from either Windows PE or a previous Windows installation.</p>
<b>offlineServicing</b>	<p>This configuration pass is used to apply updates, drivers, or language packs to a Windows image.</p> <p>During Windows Setup, the Windows image is applied to a hard disk and any settings in the <b>offlineServicing</b> section of an answer file are then applied to that image before the computer reboots.</p> <p>During this configuration pass, you can add drivers to a Windows image before the image starts. This enables you to install and process out-of-box device drivers during Windows Setup.</p> <p>This configuration pass is also used to apply updates to a Windows image during servicing scenarios.</p>	<ul style="list-style-type: none"> <li>• Automatically after the <b>windowsPE</b> configuration pass and before the computer reboots.</li> <li>• During servicing scenarios when you specify an answer file by using the Deployment Image Servicing and Management tool (Dism.exe).</li> </ul>

CONFIGURATION PASS	DESCRIPTION	CONFIGURATION PASS RUNS WHEN
<b>specialize</b>	<p>This configuration pass is used to create and configure information in the Windows image, and is specific to the hardware that the Windows image is installing to.</p> <p>After the Windows image boots for the first time, the <b>specialize</b> configuration pass runs. During this pass, unique security IDs (SIDs) are created. Additionally, you can configure many Windows features, including network settings, international settings, and domain information.</p> <p>The answer file settings for the <b>specialize</b> pass appear in audit mode. When a computer boots to audit mode, the <b>auditSystem</b> pass runs, and the computer processes the auditUser settings.</p>	<ul style="list-style-type: none"> <li>Automatically when the Windows image boots for the first time.</li> <li>On the next boot after you run the <b>sysprep</b> command with the <b>/generalize</b> option.</li> </ul>
<b>generalize</b>	<p>During this configuration pass, computer-specific information is removed from the Windows installation enabling you to capture and reapply the Windows image to different computers. For example, during this pass, the unique security ID (SID), unique device drivers, and other hardware-specific settings are removed from the image.</p> <p>This configuration pass enables you to minimally configure the <b>sysprep /generalize</b> command, in addition to configuring other Windows settings that must persist on your master image.</p> <p>After the <b>generalize</b> pass finishes, the next time that Windows image boots, the <b>specialize</b> configuration pass runs. If you want to retain the unique device drivers that are installed to your Windows installation, you can use the Microsoft-Windows-PnpSysprep   <b>PersistAllDeviceInstalls</b> setting. If this setting is configured, unique device drivers are not removed from the installation.</p>	<ul style="list-style-type: none"> <li>The following setting is configured: Microsoft-Windows-Deployment   <b>Generalize</b>.</li> </ul> <p>- or -</p> <ul style="list-style-type: none"> <li>Run the <b>sysprep /generalize</b> command.</li> </ul>

CONFIGURATION PASS	DESCRIPTION	CONFIGURATION PASS RUNS WHEN
<b>auditSystem</b>	<p>During this configuration pass, settings are processed when Windows is running in system context, before a user logs onto the computer in Audit mode.</p> <p>This pass is typically used to make additional configurations to an installation, such as installing out-of-box device drivers.</p> <p>This pass runs only when a computer is configured to boot to audit mode.</p>	<ul style="list-style-type: none"> <li>The following unattended Setup setting is configured: Microsoft-Windows-Deployment   Reseal   Mode =Audit.</li> </ul> <p>- or -</p> <ul style="list-style-type: none"> <li>Run the <b>sysprep</b> command with the <b>/audit</b> option.</li> </ul>
<b>auditUser</b>	<p>This pass processes unattended Setup settings, after a user logs onto the computer in audit mode.</p> <p>This pass is typically used to run custom commands or configure Windows Shell options.</p> <p>This pass runs only when a computer is configured to boot to audit mode.</p>	<ul style="list-style-type: none"> <li>The following unattended Setup setting is configured: Microsoft-Windows-Deployment   Reseal   Mode =Audit.</li> </ul> <p>- or -</p> <ul style="list-style-type: none"> <li>Run the <b>sysprep</b> command with the <b>/audit</b> option.</li> </ul>
<b>oobeSystem</b>	<p>During this configuration pass, settings are applied to Windows before Windows Welcome starts.</p> <p>This pass is typically used to configure Windows Shell options, create user accounts, and specify language and locale settings.</p> <p>The answer file settings for the <b>oobeSystem</b> pass appear in Windows Welcome, also known as OOBE. These settings do not appear in audit mode.</p>	<ul style="list-style-type: none"> <li>The following setting is configured: Microsoft-Windows-Deployment   Reseal   Mode =OOBE</li> </ul> <p>- or -</p> <ul style="list-style-type: none"> <li>Run the <b>sysprep</b> command with the <b>/OOBE</b> option.</li> </ul>

For more information about Windows components and settings that can be added to an answer file, see the Unattended Windows Setup Reference Guide. For more information about logging, see [Deployment Troubleshooting and Log Files](#) and [Windows Setup Log Files and Event Logs](#).

## Configuring Device Drivers

To add out-of-box, boot-critical drivers during an unattended installation, you must make sure that the boot-critical driver is available on preinstallation media. Boot-critical drivers should be added during the **windowsPE** configuration pass. All drivers are staged in the driver store, but only boot-critical drivers are reflected or installed in the offline Windows image in addition to the Windows PE image. Non-boot-critical drivers can be added to the **offlineServicing** configuration pass. This makes sure that boot-critical drivers are available and when the computer boots, the driver will load.

For more information, see [Device Drivers and Deployment Overview](#).

# Configuring International Settings

International settings are available in multiple configuration passes, to enable you to customize the Windows image based on customer requirements and different deployment scenarios.

For example, if you build a computer in the United States (which would be an en-US international setting), you might perform all your tests in English. However, if you intend to deliver the computer to France and need Windows to boot in French, you can add the fr-FR language pack, if the language pack is not already installed, and then configure the Microsoft-Windows-International-Core component to apply fr-FR settings during the **specialize** configuration pass. When the computer boots, the installation will display English text. However, after the specialize configuration pass finishes, French text will be displayed.

You can use DISM to configure the language settings of a Windows image (either online or offline). For more information, see [DISM Languages and International Servicing Command-Line Options](#).

By default, Windows Welcome displays a Regional Settings user interface (UI) page for the end user to select default language, locale, and input settings. You can preconfigure the settings on this UI page by specifying language and locale settings in the **oobeSystem** configuration pass in the Microsoft-Windows-International-Core component. If settings are set in **oobeSystem** configuration pass, the Regional Settings page is skipped. If language settings are configured during specialize, the Regional Settings page will be displayed.

For more information, see [Add Language Packs to Windows](#).

## Examples

The following sections describe sample deployment scenarios and describe when configuration passes run.

### To run Windows Setup

In this scenario, you install Windows to a new computer. You start with the Windows product media and an answer file.

1. Run Windows Setup and specify an answer file. Windows Setup starts.
2. The **windowsPE** configuration pass runs. Settings in the `<settings pass="windowsPE">` section of an answer file are processed. There are two different types of settings that you can configure during the **windowsPE** configuration pass: Settings that apply to the Windows PE environment, such as the display resolution and log file locations for Windows PE. You can also specify settings that apply to the Windows installation, such as configuring disk partitions or enabling dynamic updates.
  - The Windows PE-specific settings in an answer file are applied only when you are running Windows Setup from a Windows PE environment.
  - The Windows Setup options in the **windowsPE** configuration pass are applied when it runs from either Windows PE or a previous Windows installation.
3. After the Windows image is copied to the hard disk, the **offlineServicing** configuration pass runs. Any settings in the `<servicing>` and `<settings pass="offlineServicing">` section of an answer file are applied to the Windows image. Typically, the actions in this configuration pass install or remove packages, language packs, or device drivers.
4. The system restarts and Windows Setup runs the **specialize** configuration pass. At this point, settings in the `<settings pass="specialize">` section of the answer file are processed.
5. After Windows Setup completes, the computer restarts. Then, the **oobeSystem** configuration pass runs and settings in the `<settings pass="oobeSystem">` section of an answer file are processed.

**Note** You can create a separate content file called Oobe.xml that you can use to customize Windows Welcome, Getting Started, and ISP sign up. Using Oobe.xml is useful for organizing these

customizations, because it enables you to maintain a single file that lists all of the branding, license terms, and signup opportunities for multiple countries, regions and/or languages. For more information, see [Configure Oobe.xml](#). Generally, Oobe.xml is used by OEMs and System Builders. However some aspects of Oobe.xml might also benefit corporate deployment scenarios.

6. Windows Welcome starts and you can begin using the computer.

### To run the **Sysprep /generalize /shutdown** command

In this scenario, you will create a reference Windows image to use throughout your environment. You start with a customized Windows installation.

1. Run the **sysprep** command with the **/generalize /shutdown /oobe** options, to create a master image, configure the computer to boot to Windows Welcome, and then shut down the computer.
2. The settings in the `<settings pass="generalize">` section of an answer file are applied.
  - If you did not specify an answer file with the **Sysprep** command, the answer file cached to the computer will be used. For more information about how to use answer files, see [Windows Setup Automation Overview](#).
  - If you specified an answer file with the **sysprep** command, that answer file is cached to the %WINDIR%\Panther directory of the Windows installation and will be used on subsequent configuration passes.
3. The computer shuts down, enabling you to boot to Windows PE or another operating system and capture the image. The next time the Windows image boots, the **specialize** configuration pass will run and Windows will boot the computer to Windows Welcome.

### Using a Script to Deploy a Windows Image

In this scenario, you boot the computer with a master image on which the **sysprep /generalize /shutdown /oobe** command was run and the image was captured. You start with a master image, Windows PE and the DISM tool.

1. Apply the master image to a computer by using the **dism** command with the **/apply-image** option.
2. Boot the computer with the master image. Windows starts.
3. The **specialize** configuration pass runs. Settings in the `<settings pass="specialize">` section of the answer file are processed.
4. The computer restarts.
5. The **oobeSystem** configuration pass runs. Settings in the `<settings pass="oobeSystem">` section of the answer file are processed.
6. Windows Welcome starts and you can begin using your computer.

### To boot Windows to audit mode

In this scenario, you boot a Windows image that is configured to start in audit mode. Audit mode is useful for adding custom applications, drivers, and other updates to a master image. You can configure a Windows image to boot the computer to audit mode by configuring the following setting in an answer file: Microsoft-Windows-Deployment | Reseal | `Mode =Audit` or, run the **Sysprep** command with the **/audit** option.

1. Configure the Windows image to boot the computer to audit mode. In this scenario, run the **sysprep** command with the **/audit /reboot** options.
2. Windows reboots the computer.
3. The **auditSystem** configuration pass runs. Settings in the `<settings pass="auditSystem">` section of the

answer file are processed.

4. The Built-in administrator account is enabled.
5. The **auditUser** configuration pass runs. Settings in the `<settings pass="auditUser">` section of the answer file are processed.
6. The desktop appears.

The next time that you reboot the computer, it will boot to audit mode again.

To configure the computer to boot to Windows Welcome, you must use the **sysprep** command with the **/oobe** option, or configure the Microsoft-Windows-Deployment | Reseal | `Mode` setting to **oobe** in an answer file.

### To run DISM against an offline Windows image

In this scenario, you run DISM against an offline Windows image.

1. Run DISM tool against an offline Windows image and specify an answer file. For example, to list the package in an offline Windows image, use the following command:

```
Dism /image:C:\test\offline /Get-Packages
```

2. Settings in the `<servicing>` and `<settings pass="offlineServicing">` sections of an answer file are applied to the Windows image. The next time that you boot your computer, the packages and settings are processed.

For more information, see [DISM Image Management Command-Line Options](#).

### To use DISM on a running Windows image

In this scenario, you run the DISM tool against a running Windows installation.

- Run DISM against an online Windows image and specify an answer file. For example, to list driver information in a Windows image, use the following command:

```
Dism /online /Get-Drivers
```

**Important** When you use DISM with an answer file against an online Windows installation, the answer file should contain only the elements in the **offlineServicing** configuration pass. This is because some settings in the **specialize** configuration pass might be applied to the online Windows installation.

In some instances, you might be required to restart your computer. For example, if you add a language pack to your Windows installation, you must reboot the computer.

## Related topics

[auditSystem](#)

[auditUser](#)

[generalize](#)

[offlineServicing](#)

[oobeSystem](#)

[specialize](#)

[windowsPE](#)

# auditSystem

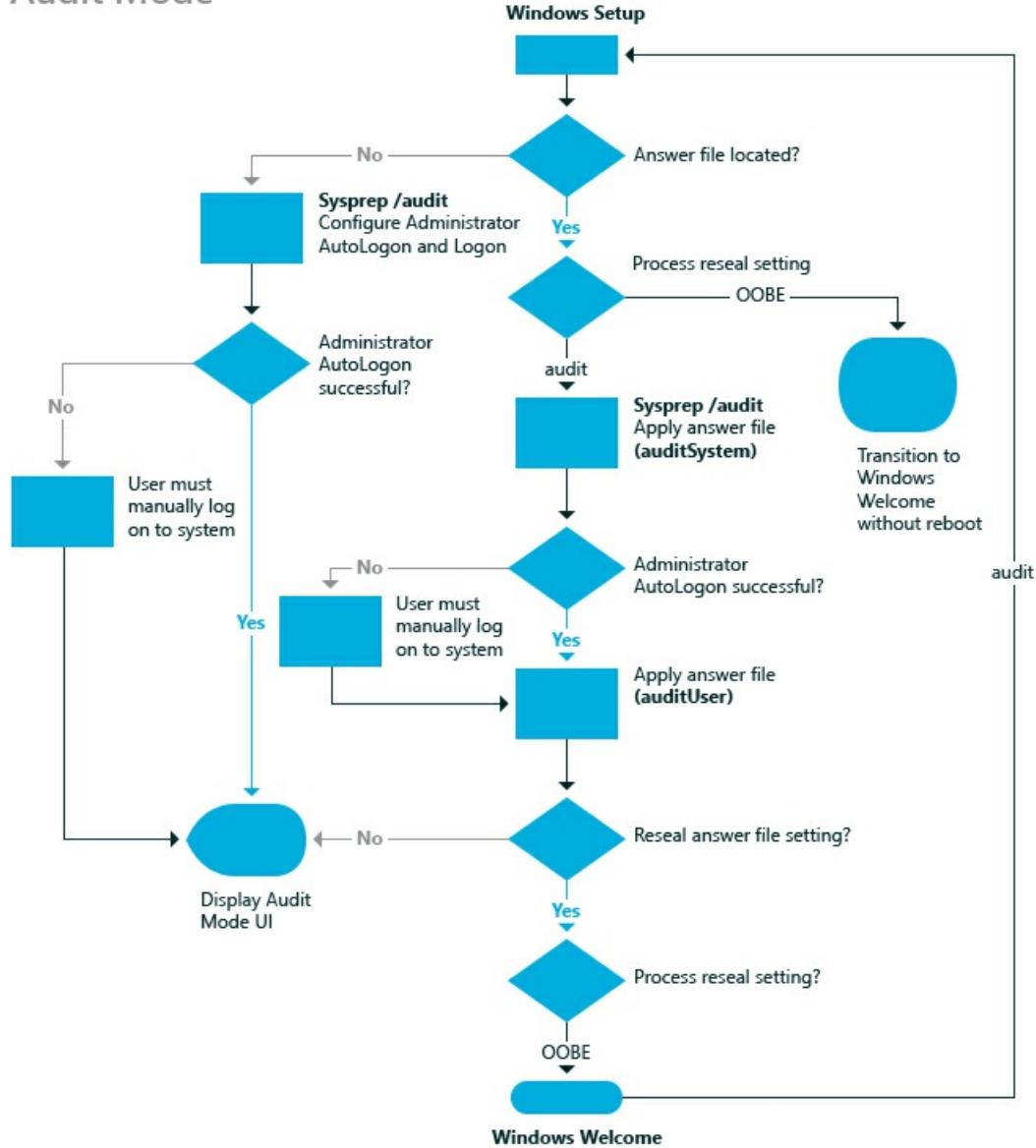
1/18/2019 • 2 minutes to read

The **auditSystem** configuration pass processes unattended Windows® Setup settings in system context in audit mode. The **auditSystem** configuration pass runs immediately before the **auditUser** configuration pass, which is used to apply settings in user context. When Windows boots to audit mode, the **auditSystem** configuration pass and the **auditUser** unattended Windows Setup settings are processed.

Audit mode enables OEMs and corporations to install additional device drivers, applications, and other updates to a master Windows image. By using audit mode, you can maintain fewer images because you can create a reference image with a minimal set of drivers and applications. The reference image can then be updated with additional drivers during audit mode. Additionally, you can then test and resolve any issues related to malfunctioning or incorrectly installed devices on the Windows image before shipping the computer to a customer. Audit mode is optional.

The following diagram shows when the **auditSystem** configuration pass is processed in audit mode.

## Audit Mode



The **auditSystem** configuration pass runs only when you configure Windows Setup to boot into audit mode. You can boot to audit mode by using the **sysprep** command with the **audit** option, or the **sysprep** command

with the **generalize** and **audit** options, or you can specify the **Reseal** setting in the Microsoft-Windows-Deployment component. For more information, see [Audit Mode Overview](#) and [Boot Windows to Audit Mode or OOBED](#).

## Related topics

[How Configuration Passes Work](#)

[auditUser](#)

[generalize](#)

[offlineServicing](#)

[oobeSystem](#)

[specialize](#)

[windowsPE](#)

# auditUser

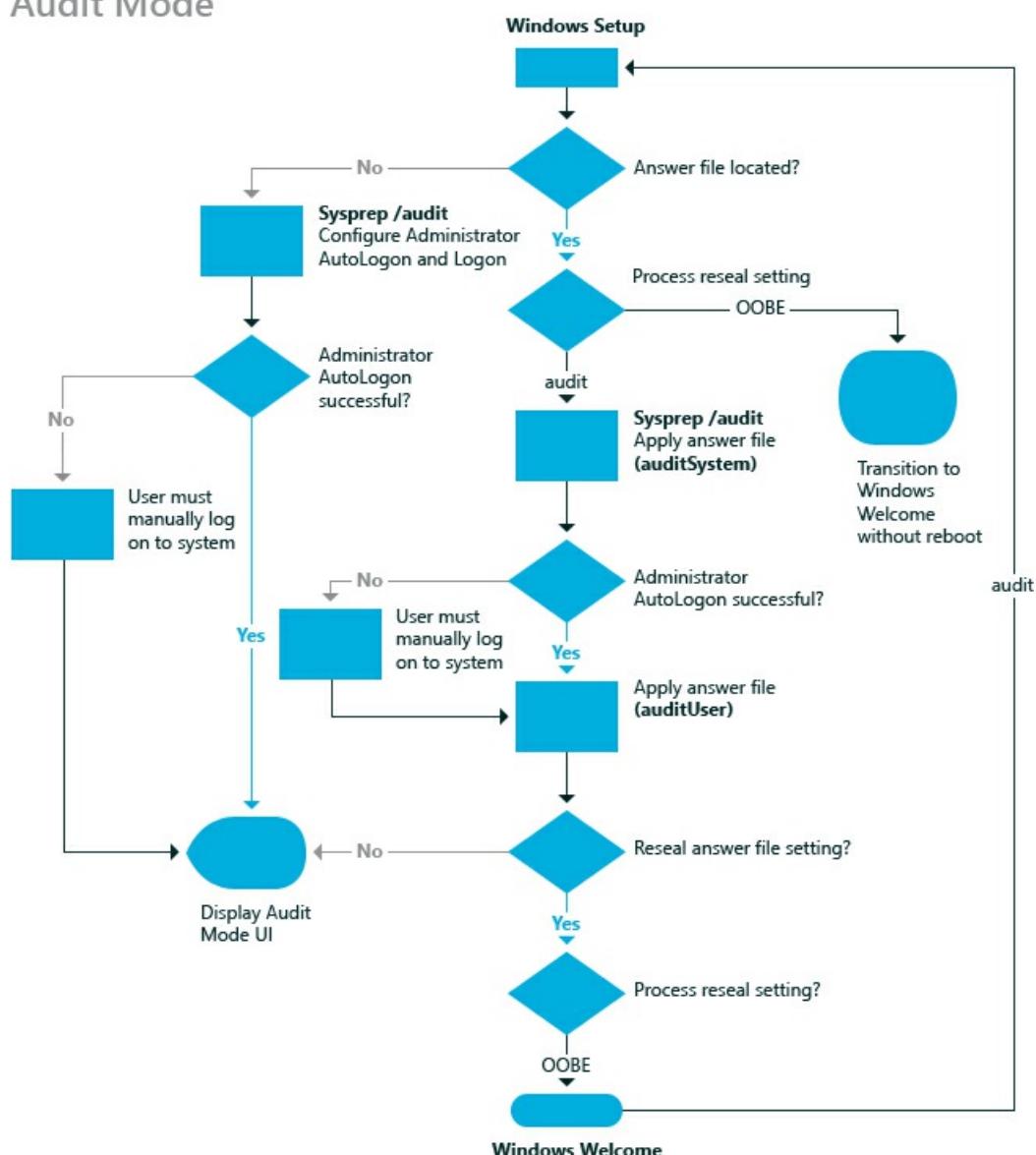
1/18/2019 • 2 minutes to read

The **auditUser** configuration pass processes unattended Windows® Setup settings in user context in audit mode. The **auditUser** configuration pass always runs after the **auditSystem** pass, which is used to apply settings in system context. Typically, the **auditUser** configuration pass is used to execute **RunSynchronous** or **RunAsynchronous** commands. These commands are used to run scripts, applications, or other executables during audit mode. When Windows boots to audit mode, the **auditSystem** and **auditUser** settings for unattended Windows Setup are processed.

Audit mode enables OEMs and corporations to install additional device drivers, applications, and other updates to a master Windows® image. By using audit mode, you can maintain fewer images because you can create a reference image with a minimal set of drivers and applications. The reference image can then be updated with additional drivers during audit mode. Additionally, you can test and resolve any issues related to malfunctioning or incorrectly installed devices on the Windows image before shipping the computer to a customer. Audit mode is optional.

The following diagram illustrates when the **auditUser** configuration pass is processed in audit mode.

## Audit Mode



The **auditUser** configuration pass runs only when you configure Windows Setup to boot into audit mode. You can boot to audit mode by using the **sysprep /audit** or **sysprep /generalize /audit** commands, or you can specify the **Reseal** setting in the Microsoft-Windows-Deployment component. For more information about audit mode, see [Audit Mode Overview](#) and [Boot Windows to Audit Mode or OOBE](#).

## Related topics

[How Configuration Passes Work](#)

[auditSystem](#)

[generalize](#)

[offlineServicing](#)

[oobeSystem](#)

[specialize](#)

[windowsPE](#)

# generalize

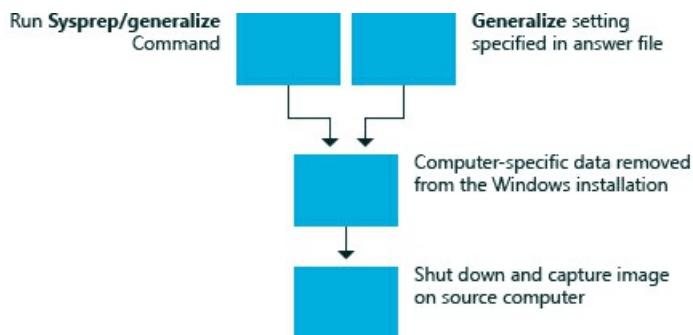
1/18/2019 • 2 minutes to read

The **generalize** configuration pass of Windows® Setup is used to create a Windows reference image that can be used throughout an organization. Settings in the **generalize** configuration pass enable you to automate the behavior for all deployments of this reference image. In comparison, settings applied in the **specialize** configuration pass enable you to override behavior for a single, specific deployment.

When a system is generalized, specific configuration data for a given installation of Windows is removed. For example, during the **generalize** configuration pass, the unique security ID (SID) and other hardware-specific settings are removed from the image.

The **generalize** configuration pass runs only when you use the **Sysprep** command with the **/generalize** option. Answer file settings in the `<generalize>` section of an answer file are applied to the system before **Sysprep** generalization occurs. The system then shuts down.

The following diagram shows the process of the **generalize** configuration pass.



The **specialize** configuration pass runs immediately after the next time that the system boots. When you run **Sysprep**, you can decide whether Windows will boot to audit mode or Windows Welcome by specifying **/audit** or **/oobe**. The **specialize** configuration pass always runs after a computer has been generalized, regardless of whether the computer is configured to boot to audit mode or Windows Welcome.

Any method of moving or copying a Windows image to a new computer must be prepared with the **sysprep /generalize** command. For more information, see [Sysprep \(Generalize\) a Windows installation](#).

## Related topics

[How Configuration Passes Work](#)

[auditSystem](#)

[auditUser](#)

[offlineServicing](#)

[oobeSystem](#)

[specialize](#)

[windowsPE](#)

# offlineServicing

1/18/2019 • 2 minutes to read

Use the **offlineServicing** configuration pass to apply unattended Setup settings to an offline Microsoft® Windows® image. During this configuration pass, you can add language packs, updates, device drivers, or other packages to the offline image.

The **offlineServicing** configuration pass runs during Windows Setup. Setup extracts and installs the Windows image, and then executes the Deployment Image Servicing and Management (Dism.exe) tool. Packages listed in the `<servicing>` section and settings in the `<offlineServicing>` section of the answer file are applied to the offline Windows image.

Additionally, you can use the Deployment Image Servicing and Management tool with an answer file to apply settings in the **offlineServicing** pass. For more information, see [Service a Windows Image Using DISM](#).

## Related topics

[How Configuration Passes Work](#)

[auditSystem](#)

[auditUser](#)

[generalize](#)

[oobeSystem](#)

[specialize](#)

[windowsPE](#)

# oobeSystem

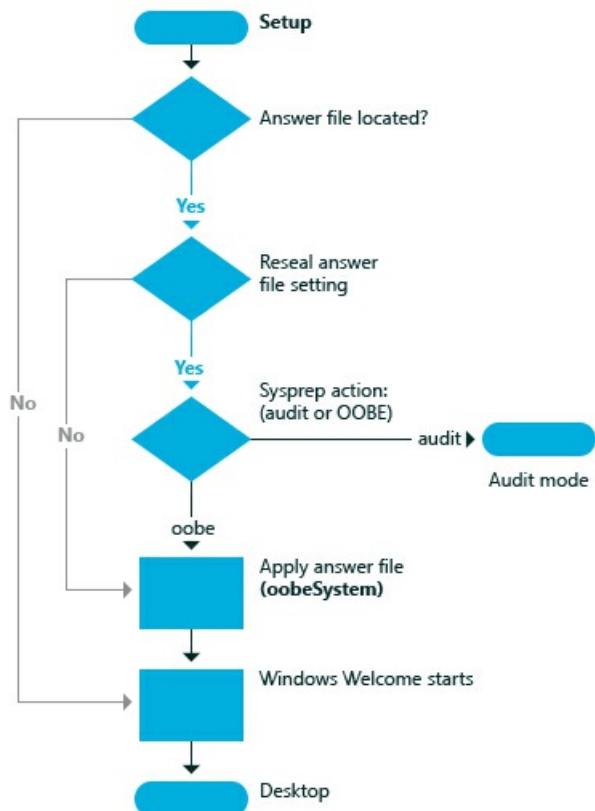
1/18/2019 • 2 minutes to read

The **oobeSystem** configuration pass configures settings that are applied during the end-user first-boot experience, also called Out-Of-Box Experience (OOBE). The **oobeSystem** configuration pass settings are processed before a user first logs on to Windows®.

Out-of-Box-Experience (OOBE) runs the first time the user starts a newly configured computer. OOBE runs before the Windows shell or any additional software runs, and it performs a small set of tasks that are required to configure and run Windows.

The following diagram illustrates the process that occurs when an end user first boots a newly configured computer. The result is OOBE, or a user's first-boot experience.

## Windows Welcome



You can configure Windows to boot to OOBE by running the **sysprep** command by using the **/oobe** option. By default, after running Windows Setup, OOBE starts.

## Related topics

[How Configuration Passes Work](#)

[auditSystem](#)

[auditUser](#)

[generalize](#)

[offlineServicing](#)

[specialize](#)

windowsPE

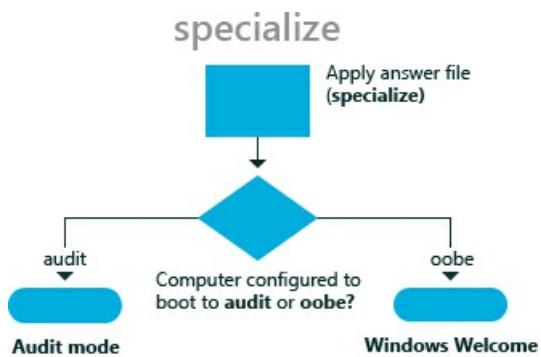
# specialize

1/18/2019 • 2 minutes to read

During the **specialize** configuration pass of Windows® Setup, computer-specific information for the image is applied. For example, you can configure network settings, international settings, and domain information.

The **specialize** configuration pass is used together with the [generalize](#) configuration pass. The [generalize](#) pass is used to create a Windows reference image that can be used throughout an organization. From this basic Windows reference image, you can add additional customizations that apply to different divisions in an organization or to different installations of Windows. Any method of moving or copying a Windows image to a new computer must be prepared with the **sysprep /generalize** command. For more information, see [Sysprep \(System Preparation\) Overview](#) and [Sysprep Command-Line Options](#).

The following diagram illustrates how the **specialize** configuration pass is used to apply these specific customizations.



For example, during the **specialize** configuration pass, you can specify different home pages in Internet Explorer® for different departments or branches in your business. This setting will then override the default home page.

## Related topics

[How Configuration Passes Work](#)

[auditSystem](#)

[auditUser](#)

[generalize](#)

[offlineServicing](#)

[oobeSystem](#)

[windowsPE](#)

# windowsPE

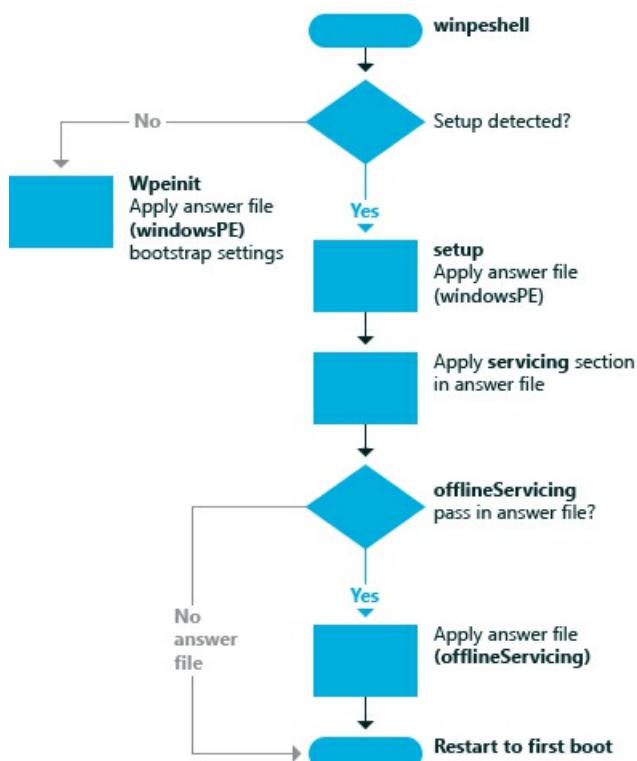
1/18/2019 • 2 minutes to read

The **windowsPE** configuration pass is used to configure settings specific to Windows® Preinstallation Environment (Windows PE) in addition to settings that apply to installation.

For example, you can specify the display resolution of Windows PE, where to save a log file, and other Windows PE-related settings.

The following diagram illustrates the **windowsPE** configuration pass.

## windowsPE



The **windowsPE** configuration pass also enables you to specify Windows Setup-related settings, including:

- Partition and format a hard disk.
- Select a specific Windows image to install, the path of that image, and any credentials required to access that image.
- Select a partition on the destination computer where you install Windows.
- Apply a product key and administrator password.
- Run specific commands during Windows Setup.

## Related topics

[How Configuration Passes Work](#)

[auditSystem](#)

[auditUser](#)

generalize

offlineServicing

oobeSystem

# Deployment Troubleshooting and Log Files

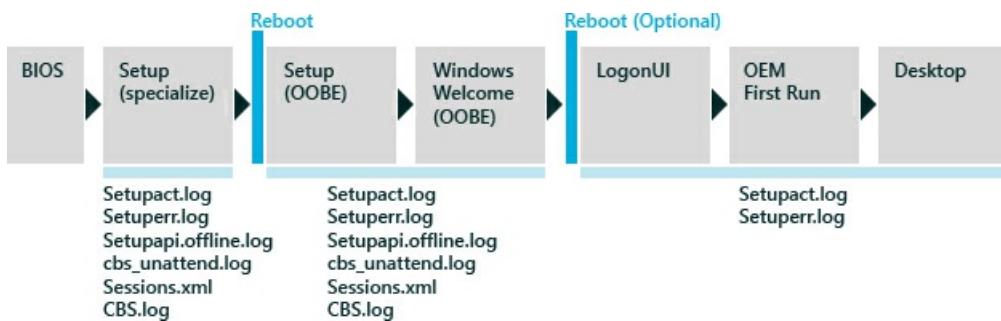
1/18/2019 • 5 minutes to read

The following section describes the relationship between common deployment scenarios and their associated log files. Windows® deployment is a highly customizable process, which has the potential for many points of failure. Identifying the specific point of failure you have encountered begins with understanding how the underlying technologies work.

## Windows Setup Scenario

This scenario begins with completing Windows Setup on a new computer, so that you arrive at the desktop. This scenario is most common when you are creating a reference image. This process is also known as the *first user experience*.

As shown in the following illustration, the key to solving failures is identifying where you are in the installation process and when a failure occurs. Because you are creating a new installation, the hard drive is not initially available, so Windows Setup writes logs into memory, specifically in a Windows PE session (X:\Windows). After the hard drive is formatted, Setup continues logging directly onto the new hard drive (C:\Windows). Log files created during the Windows PE session are temporary.



When a failure occurs in Windows Setup, review the entries in the Setupperr.log file first, then the Setupact.log file second, and then other log files as needed.

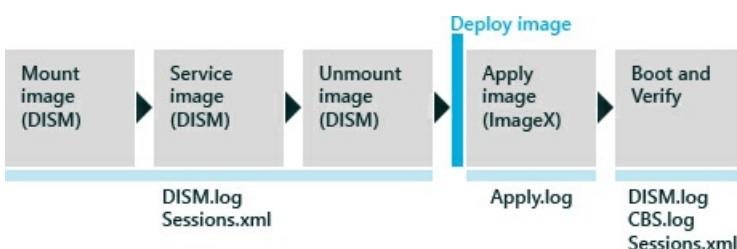
## Windows Setup-Related Log Files

LOG FILE	DESCRIPTION	LOCATION
Setupact.log	Primary log file for most errors that occur during the Windows installation process. There are several instances of the Setupact.log file, depending on what point in the installation process the failure occurs. It is important to know which version of the Setupact.log file to look at, based on the phase you are in.	<b>Setup (specialize):</b> X:\Windows\panther  <b>Setup (OOBE), LogonUI, OEM First Run:</b> %windir%\panther  <b>Out-Of-Box Experience (OOBE):</b> %windir%\panther\unattendGC

LOG FILE	DESCRIPTION	LOCATION
Setuperr.log	High-level list of errors that occurred during the <b>specialize</b> phase of Setup. The Setuperr.log file does not provide any specific details.	<b>Setup (specialize):</b> %windir%\panther  <b>Setup (specialize):</b> %windir%\panther  <b>Setup (OOBE), LogonUI, OEM First Run:</b> %windir%\panther
Setupapi.offline.log	Driver failures during the Component Specialization sub-phase of the Setup <b>specialize</b> phase.	%windir%\inf
Cbs_unattend.log	Unattended-setup servicing failures.	%windir%\panther
Setupapi.dev.log	Driver failures during the <b>oobe</b> phase of Setup.	%windir%\inf
Sessions.xml	An XML-based transaction log file that tracks all servicing activity, based on session id, client, status, tasks, and actions. If necessary, the Sessions.log file will point to the DISM.log and CBS.log files for more details.	%windir%\servicing\sessions
CBS.log	Servicing log file that provides more details about offline-servicing failures.	%windir%\panther

## Offline Servicing Scenario

This scenario involves adding and removing updates, drivers, and language packs, and configuring other settings, without booting Windows. Offline servicing is an efficient way to manage existing images that are stored on a server, because it eliminates the need for recreating updated images. You can perform offline servicing on an image that is mounted or applied to a drive or directory.



The Deployment Image Servicing and Management (DISM) tool is the primary tool for all offline-servicing tasks. DISM runs from a command prompt from Windows PE or a running Windows operating system. If a failure occurs when executing a DISM command, the tool will provide an immediate response, and log the issue in the DISM.log file. The Session.xml file is a transaction log file that captures all servicing activities on the target operating system. The Session.xml file can be used in conjunction with the DISM.log file to determine points of

failures and the required servicing activity.

When a failure occurs in offline servicing, look at the DISM.log file first for specific errors. If the DISM.log file doesn't contain any errors, review the Sessions.xml log file second, and then the CBS.log file.

### Offline Servicing Related Log Files

LOG FILE	DESCRIPTION	LOCATION
DISM.log	Primary log file for all offline actions using DISM.	%windir%\logs\dism  You can also create the DISM log file in a different location by using the <b>/LogPath</b> option. The level of data written to the log file can also be controlled by using the <b>/LogLevel</b> option.
Sessions.xml	An XML-based transaction log that tracks all servicing activity, based on session id, client, status, tasks, and actions. If necessary, the Sessions.log file will point to the DISM.log and CBS.log files for more details.	%windir%\servicing\sessions

To learn more about offline servicing, see [Understanding Servicing Strategies](#).

## Online Servicing Scenario

This scenario is servicing a running operating system. This scenario involves booting the computer to audit mode to add drivers, applications, and other packages. Online servicing is ideal for drivers if the driver packages have co-installers or application dependencies. It is also efficient when the majority of your servicing packages have installers, the updates are in either .msi or KB.exe file formats, or the applications rely on Windows-installed services and technologies (such as the .NET Framework or full plug and play support).



Like offline servicing, all logging is captured in the DISM.log, CBS.log, and Sessions.xml files. If a failure occurs when executing a DISM command, the tool will provide immediate response as well as log the issue in the DISM.log file. The Session.xml file is a transaction log file that captures all servicing activities on the target operating system. The Session.xml file can be used in conjunction with the DISM.log file to determine points of failures and the required servicing activities.

When a failure occurs in offline servicing, look at the DISM.log file for specific errors. If the DISM.log file doesn't contain any errors, review the Sessions.xml log file and then the CBS.log file.

### Online Servicing-Related Log Files

LOG FILE	DESCRIPTION	LOCATION
----------	-------------	----------

LOG FILE	DESCRIPTION	LOCATION
DISM.log	Primary log file for all online actions using DISM. If necessary, DISM.log will point to CBS.log for more details.	%windir%\logs\dism  You can also point DISM log file to a different location by using the /LogPath command option. The log data can also be controlled by using the /LogLevel command option.
CBS.log	Secondary log file that provides more details about an online servicing failure. DISM.log will reference CBS.log for more details.	%windir%\logs\cbs
Sessions.xml	An xml based transaction log that tracks all servicing activity based on session id, client, status, tasks, and actions. If necessary, Sessions.log will point to DISM.log and CBS.log for more details.	%windir%\servicing\sessions

To learn more about offline servicing, see [Understanding Servicing Strategies](#).

# Windows Deployment Command-Line Tools Reference

1/18/2019 • 2 minutes to read

These command-line tools are often used when manufacturing Windows devices.

## In This Section

<a href="#">BCDBoot Command-Line Options</a>	Initializes the boot configuration data (BCD) store and copies boot environment files to the system partition during image deployment.
<a href="#">BCDEdit Command-Line Options</a>	Manages Boot Configuration Data (BCD).
<a href="#">Bootsect Command-Line Options</a>	Updates the master boot code for hard disk partitions to switch between Windows Boot Manager ( <b>Bootmgr.exe</b> ) and Windows NT Loader (NTLDR).
<a href="#">DiskPart</a>	Manages disk partitions.
<a href="#">Oscdimg Command-Line Options</a>	Creates an image (.iso) file of a customized 32-bit or 64-bit version of Windows PE.

## Related topics

[Server Manager command-line tools](#)

[Windows Deployment Tools Technical Reference](#)

# BCDBoot Command-Line Options

1/18/2019 • 7 minutes to read

BCDBoot is a command-line tool used to configure the boot files on a PC or device to run the Windows operating system. You can use the tool in the following scenarios:

- **Add boot files to a PC after applying a new Windows image.** In a typical image-based Windows deployment, use BCDBoot to set up the firmware and system partition to boot to your image. To learn more, see [Capture and Apply Windows, System, and Recovery Partitions](#).
- **Set up the PC to boot to a virtual hard disk (VHD) file that includes a Windows image.** To learn more, see [Boot to VHD \(Native Boot\): Add a Virtual Hard Disk to the Boot Menu](#).
- **Repair the system partition.** If the system partition has been corrupted, you can use BCDBoot to recreate the system partition files by using new copies of these files from the Windows partition.
- **Set up or repair the boot menu on a dual-boot PC.** If you've installed more than one copy of Windows on a PC, you can use BCDBoot to add or repair the boot menu.

## File Locations

In Windows and Windows Preinstallation Environment (WinPE)	%WINDIR%\System32\BCDBoot.exe
In the Windows Assessment and Deployment Kit (Windows ADK):	C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Deployment Tools\amd64\BCDBoot\BCDBoot.exe

## Supported operating systems

BCDBoot can copy boot environment files from images of Windows 10, Windows 8.1, Windows 8, Windows 7, Windows Vista, Windows Server 2016 Technical Preview, Windows Server 2012 R2, Windows Server 2012, Windows Server 2008 R2, or Windows Server 2008.

## How It Works

To configure the system partition, BCDBoot copies a small set of boot-environment files from the installed Windows image to the system partition.

BCDBoot can create a Boot Configuration Data (BCD) store on the system partition using the latest version of the Windows files:

- BCDBoot creates a new BCD store and initialize the BCD boot-environment files on the system partition, including the Windows Boot Manager, using the %WINDIR%\System32\Config\BCD-Template file.
- New in Windows 10: During an upgrade, BCDBoot preserves any other existing boot entries, such as **debugsettings**, when creating the new store. Use the **/c** option to ignore the old settings and start fresh with a new BCD store.
- If there is already a boot entry for this Windows partition, by default, BCDBoot erases the old boot entry and its values. Use the **/m** option to retain the values from an existing boot entry when you update the system files.
- By default, BCDBoot moves the boot entry for the selected Windows partition to the top of the Windows

Boot Manager boot order. Use the **/d** option to preserve the existing boot order.

On UEFI PCs, BCDBoot can update the firmware entries in the device's NVRAM:

- BCDBoot adds a firmware entry in the NVRAM to point to the Windows Boot Manager. By default, this entry is placed as the first item in the boot list. Use the **/p** option to preserve the existing UEFI boot order. Use **/addlast** to add it to the bottom of the boot order list.

## Command-Line Options

The following command-line options are available for BCDBoot.exe.

**BCDBOOT** <source> [/l <locale>] [/s <volume-letter>] [/f <firmware type>] [/v] [/m [{OS Loader GUID}]]  
[/addlast or /p] [/d] [/c]

OPTION	DESCRIPTION
<source>	<p>Required. Specifies the location of the Windows directory to use as the source for copying boot-environment files.</p> <p>The following example initializes the system partition by using BCD files from the C:\Windows folder:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">bcdboot C:\Windows</div>
/l <locale>	<p>Optional. Specifies the locale. The default is US English (en-us).</p> <p>The following example sets the default BCD locale to Japanese:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">bcdboot C:\Windows /l ja-jp</div>

OPTION	DESCRIPTION
<p>/s &lt;volume letter&gt;</p>	<p>Optional. Specifies the volume letter of the system partition. This option should not be used in typical deployment scenarios.</p> <p>Use this setting to specify a system partition when you are configuring a drive that will be booted on another computer, such as a USB flash drive or a secondary hard drive.</p> <p><b>UEFI:</b></p> <ul style="list-style-type: none"> <li>BCDBoot copies the boot files to either the EFI system partition, or the partition specified by the /s option.</li> </ul> <p>BCDBoot creates the BCD store in the same partition.</p> <p>By default, BCDBoot creates a Windows Boot Manager entry in the NVRAM on the firmware to identify the boot files on the system partition. If the /s option is used, then this entry is not created. Instead, BCDBoot relies on the default firmware settings to identify the boot files on the system partition. By the UEFI 2.3.1 spec, the default firmware settings should open the file: \efi\boot\bootx64.efi in the EFI System Partition (ESP).</p> <p><b>BIOS:</b></p> <ol style="list-style-type: none"> <li>BCDBoot copies the boot files to either the active partition on the primary hard drive, or the partition specified by the /s option.</li> <li>BCDBoot creates the BCD store in the same partition.</li> </ol> <p>The following example copies BCD files from the C:\Windows folder to a system partition on a secondary hard drive that will be booted on another computer. The system partition on the secondary drive was assigned the volume letter S:</p> <pre>bcdboot C:\Windows /s S:</pre> <p>The following example creates boot entries on a USB flash drive with the volume letter S, including boot files to support either a UEFI-based or a BIOS-based computer:</p> <pre>bcdboot C:\Windows /s S: /f ALL</pre>

OPTION	DESCRIPTION
<p>/f &lt;firmware type&gt;</p>	<p>Optional. Specifies the firmware type. Valid values include <b>UEFI</b>, <b>BIOS</b>, and <b>ALL</b>.</p> <ul style="list-style-type: none"> <li>On BIOS/MBR-based systems, the default value is <b>BIOS</b>. This option creates the <b>\Boot</b> directory on the system partition and copies all required boot-environment files to this directory.</li> <li>On UEFI/GPT-based systems, the default value is <b>UEFI</b>. This option creates the <b>\Efi\Microsoft\Boot</b> directory and copies all required boot-environment files to this directory.</li> <li>When you specify the <b>ALL</b> value, BCDBoot creates both the <b>\Boot</b> and the <b>\Efi\Microsoft\Boot</b> directories, and copies all required boot-environment files for BIOS and UEFI to these directories.</li> </ul> <p>If you specify the <b>/f</b> option, you must also specify the <b>/s</b> option to identify the volume letter of the system partition.</p> <p>The following example copies BCD files that support booting on either a UEFI-based or a BIOS-based computer from the C:\Windows folder to a USB flash drive that was assigned the volume letter S:</p> <pre>bcdboot C:\Windows /s S: /f ALL</pre>
<p>/v</p>	<p>Optional. Enables verbose mode. Example:</p> <pre>bcdboot C:\Windows /v</pre>
<p>/m [{OS Loader GUID}]</p>	<p>Optional. Merges the values from an existing boot entry into a new boot entry.</p> <p>By default, this option merges only global objects. If you specify an <i>OS Loader GUID</i>, this option merges the loader object in the system template to produce a bootable entry.</p> <p>The following example merges the operating-system loader in the current BCD store that the specified GUID identifies in the new BCD store:</p> <pre>bcdboot c:\Windows /m {xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx}</pre>

OPTION	DESCRIPTION
/addlast	<p>Optional. Specifies that the Windows Boot Manager firmware entry should be added last. The default behavior is to add it first. Cannot be used with /p.</p> <pre data-bbox="831 300 1434 368">bcdboot C:\Windows /addlast</pre>
/p	<p>Optional. Specifies that the existing Windows Boot Manager firmware entry position should be preserved in the UEFI boot order. If entry does not exist, a new entry is added in the first position. Cannot be used with /addlast.</p> <p>By default, during an upgrade BCDBoot moves the Windows Boot Manager to be the first entry in the UEFI boot order.</p> <pre data-bbox="831 817 1434 929">bcdboot C:\Windows /p bcdboot C:\Windows /p /d</pre>
/d	<p>Optional. Preserves the existing default operating system entry in the {bootmgr} object in Windows Boot Manager.</p> <pre data-bbox="831 1087 1434 1154">bcdboot C:\Windows /d</pre>
/c	<p>Optional. Specifies that any existing BCD elements should not be migrated.</p> <p>New for Windows 10: By default, during an upgrade, BCD elements such as <b>debugsettings</b> or <b>flightsigning</b> are preserved.</p> <pre data-bbox="831 1423 1434 1491">bcdboot C:\Windows /c</pre>

## Repair the system partition

If the system partition has been corrupted, you can use BCDBoot to recreate the system partition files by using new copies of these files from the Windows partition.

1. Boot your PC to a command line. For example, boot to the Windows installation disk and press Shift+F10, or boot to Windows PE ([WinPE: Create USB Bootable drive](#)).
2. Use Diskpart to determine which drive letter contains your Windows partition and system partition (

```
diskpart, list vol, exit
```

).
3. Optional: Format your system partition: 

```
format (drive letter of your system partition) /q
```
4. Add a boot entry for your Windows partition: 

```
bcdboot D:\Windows
```
5. Reboot the PC. Windows should appear.

# Set up or repair the boot menu on a dual-boot PC

When setting up a PC to boot more than one operating system, you may sometimes lose the ability to boot into one of the operating systems. The BCDBoot option allows you to quickly add boot options for a Windows-based operating system. To set up a dual-boot PC:

1. Install a separate hard drive or prepare a separate partition for each operating system.
2. Install the operating systems. For example, if your PC has Windows 7, install Windows 10 onto the other hard drive or partition.
3. Reboot the PC. The boot menus should appear with both operating systems listed.

If both operating systems aren't listed:

- a. Open a command line, either as an administrator from inside Windows, or by booting to a command line using the Windows installation media and pressing Shift+F10, or by booting to Windows PE ([WinPE: Create USB Bootable drive](#)).
- b. Add boot options for a Windows operating system.

```
bcdboot D:\Windows
```

- c. Reboot the PC. Now, the boot menu will show both menu options.

## Troubleshooting

For information about repairing the boot files on a PC with Windows XP and a more recent version of Windows such as Windows 7, see the Microsoft Knowledge Base Article [2277998](#).

## Related topics

[Capture and Apply Windows, System, and Recovery Partitions](#)

[Configure BIOS/MBR-Based Hard Drive Partitions](#)

[Configure UEFI/GPT-Based Hard Drive Partitions](#)

[BCDedit](#)

[Bootsect Command-Line Options](#)

[Diskpart Command line syntax](#)

# Repair the boot menu on a dual-boot PC

1/18/2019 • 2 minutes to read

When setting up a PC to boot more than one operating system, you may sometimes lose the ability to boot into one of the operating systems. The BCDBoot option allows you to quickly add boot options for a Windows-based operating system.

## Repairing a Windows partition on a dual-boot PC

1. Install a separate hard drive or prepare a separate partition for each operating system.
2. Install the operating systems. For example, if your PC has Windows 8.1, install Windows 10 onto the other hard drive or partition.
3. Reboot the PC. The boot menus should appear with both operating systems listed.

If both operating systems aren't listed:

- a. Open a command line, either as an administrator from inside Windows, or by booting to a command line using the Windows installation disk and presssing Shift+F10, or by booting to Windows PE ([WinPE: Create USB Bootable drive](#)).
- b. Add boot options for a Windows operating system.

```
Bcdboot D:\Windows
```

- c. Reboot the PC. Now, the boot menu will show both menu options.

## Repair another operating system partition

You can manually add create partitions using BCDEdit, or you can use a third-party tool such as [EasyBCD](#) to set up the boot partitions.

## Related topics

[BCDboot Command-Line Options](#)

# BCDEdit Command-Line Options

1/18/2019 • 5 minutes to read

BCDEdit is a command-line tool for managing Boot Configuration Data (BCD).

BCD files provide a store that is used to describe boot applications and boot application settings.

BCDEdit can be used for a variety of purposes, including creating new stores, modifying existing stores, adding boot menu options, and so on.

You'll need administrative privileges to use BCDEdit to modify BCD. Start the Command Prompt (Admin) or use Windows PE.

A normal shutdown and reboot is necessary to ensure that any modified BCDEdit settings are flushed to disk.

BCDEdit is included in the %WINDIR%\System32 folder.

BCDEdit is limited to the standard data types and is designed primarily to perform single common changes to BCD. Related resources:

- Some common BCD operations, such as recovering a partition or setting up a new PC's system partition, may be more easily accomplished by using [BCDboot](#).
- For complex operations or nonstandard data types, consider using the BCD Windows Management Instrumentation (WMI) application programming interface (API) to create more powerful and flexible custom tools.

## BCDEdit Command-Line Options

The following command-line options are available for BCDEdit.exe.

**BCDEdit /Command**[Argument1] [Argument2] ...

### Help

OPTION	DESCRIPTION
/? [command]	<p>Displays a list of BCDEdit commands.</p> <p>To display detailed help for a particular command, run <b>bcdedit /?command</b>, where <i>command</i> is the name of the command you are searching for more information about.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"><code>bcdedit /? createstore</code></div>

### Operating on a store

OPTION	DESCRIPTION
/createstore	Creates a new empty boot configuration data store. The created store is not a system store.

OPTION	DESCRIPTION
/export	Exports the contents of the system store into a file. This file can be used later to restore the state of the system store. This command is valid only for the system store.
/import	Restores the state of the system store by using a backup data file previously generated by using the /export option. This command deletes any existing entries in the system store before the import takes place. This command is valid only for the system store.
/store	This option can be used with most BCDEDIT commands to specify the store to be used. If this option is not specified, then BCDEDIT operates on the system store. Running the bcddedit /store command by itself is equivalent to running the bcddedit /enum active command.
/sysstore	Sets the system store device. This only affects EFI-based systems. It does not persist across reboots, and is only used in cases where the system store device is ambiguous.

### Operating on entries in a store

OPTION	DESCRIPTION
/copy	Makes a copy of a specified boot entry in the same system store.
/create	Creates a new entry in the boot configuration data store. If a well-known identifier is specified, then the /application, /inherit, and /device options cannot be specified. If an identifier is not specified or not well known, an /application, /inherit, or /device option must be specified.
/delete	Deletes an element from a specified entry.
/mirror	Creates mirror of entries in the store.

### Changing entry options

OPTION	DESCRIPTION
/deletevalue	Deletes a specified element from a boot entry.
/set	Sets an entry option value.

For example, this command will enable the system to trust Windows Insider Preview builds that are signed with certificates that are not trusted by default:

```
Bcdedit /set {bootmgr} flightsigning on
Bcdedit /set flightsigning on
```

Reboot after running the command. To turn off flightsigning:

```
Bcdedit /set {bootmgr} flightsigning off
Bcdedit /set flightsigning off
```

## Controlling output

OPTION	DESCRIPTION
/enum	Lists entries in a store. The /enum option is the default value for BCEdit, so running the bcdedit command without options is equivalent to running the bcdedit /enum active command.
/v	Verbose mode. Usually, any well-known entry identifiers are represented by their friendly shorthand form. Specifying /v as a command-line option displays all identifiers in full. Running the bcdedit /v command by itself is equivalent to running the bcdedit /enum active /v command.

## Controlling the boot manager

OPTION	DESCRIPTION
/bootsequence	Specifies a one-time display order to be used for the next boot. This command is similar to the /displayorder option, except that it is used only the next time the computer starts. Afterwards, the computer reverts to the original display order.
/default	Specifies the default entry that the boot manager selects when the timeout expires.
/displayorder	Specifies the display order that the boot manager uses when displaying boot options to a user.
/timeout	Specifies the time to wait, in seconds, before the boot manager selects the default entry.
/toolsdisplayorder	Specifies the display order for the boot manager to use when displaying the Tools menu.

## Emergency Management Services options

OPTION	DESCRIPTION
/bootems	Enables or disables Emergency Management Services (EMS) for the specified entry.
/ems	Enables or disables EMS for the specified operating system boot entry.
/emssettings	Sets the global EMS settings for the computer. /emssettings does not enable or disable EMS for any particular boot entry.

## Debugging

OPTION	DESCRIPTION
/bootdebug	Enables or disables the boot debugger for a specified boot entry. Although this command works for any boot entry, it is effective only for boot applications.
/dbgsettings	Specifies or displays the global debugger settings for the system. This command does not enable or disable the kernel debugger; use the /debug option for that purpose. To set an individual global debugger setting, use the bcdedit /setdbgsettings type value command.
/debug	Enables or disables the kernel debugger for a specified boot entry.
/hypervisorsettings	Sets the hypervisor parameters.

To troubleshoot a new installation, enable debug mode by modifying the boot configuration file (BCD). For example, use the following syntax to enable kernel or boot debug.

```
bcdeedit /set <id> debug on
```

-or-

```
bcdeedit /set <id> bootdebug on
```

where <id> is the GUID of the Loader object that is used to load the operating system. "Default" can be used if the operating system is the default option of the Boot Manager menu.

For examples of BCDEdit, see [Boot Configuration Data in Windows Vista](#).

## Remote event logging

OPTION	DESCRIPTION
/eventsettings	Sets the global remote event logging parameters.
/event	Enables or disables remote event logging for an operating system entry.

## Related topics

[BCDboot](#)

[BCD System Store Settings for UEFI](#)

[BCDEdit Commands for Boot Environment](#)

[4-Gigabyte Tuning: BCDEdit and Boot.ini](#)

[Boot Configuration Data in Windows Vista](#)

# Bootsect Command-Line Options

1/18/2019 • 2 minutes to read

Bootsect.exe updates the master boot code for hard disk partitions to switch between Bootmgr and NT Loader (**NTLDR**). You can use this tool to restore the boot sector on your computer. This tool replaces **FixFAT** and **FixNTFS**.

## Bootsect Commands

Bootsect uses the following command-line options:

**bootsect {/help | /nt52 | /nt60} {SYS | ALL | <DriveLetter:>} [/force] /mbr**

For example, to apply the master boot code that is compatible with NTLDR to the volume labeled E, use the following command:

**bootsect /nt52 E:**

COMMAND-LINE OPTIONS	DESCRIPTION
<b>/help</b>	Displays these usage instructions.
<b>/nt52</b>	Applies the master boot code that is compatible with NTLDR to <b>SYS</b> , <b>ALL</b> , or <DriveLetter>. The operating system installed on <b>SYS</b> , <b>ALL</b> , or <DriveLetter> must be older than Windows Vista.
<b>/nt60</b>	Applies the master boot code that is compatible with Bootmgr to <b>SYS</b> , <b>ALL</b> , or <DriveLetter>. The operating system installed on <b>SYS</b> , <b>ALL</b> , or <DriveLetter> must be Windows 8, Windows® 7, Windows Vista, Windows Server® 2012, Windows Server 2008 R2, or Windows Server 2008.
<b>SYS</b>	Updates the master boot code on the system partition that is used to boot Windows.
<b>ALL</b>	Updates the master boot code on all partitions. The <b>ALL</b> option does not necessarily update the boot code for each volume. Instead, this option updates the boot code on volumes that can be used as Windows boot volumes, which excludes any dynamic volumes that are not connected with an underlying disk partition. This restriction is present because boot code must be located at the beginning of a disk partition.

COMMAND-LINE OPTIONS	DESCRIPTION
<DriveLetter>	<p>Updates the master boot code on the volume associated with this drive letter. Boot code will not be updated if either:</p> <ul style="list-style-type: none"> <li>• &lt;DriveLetter&gt; is not associated with a volume</li> <li>• &lt;DriveLetter&gt; is associated with a volume not connected to an underlying disk partition.</li> </ul>
<b>/force</b>	<p>Forcibly dismounts the volumes during the boot code update. You must use this option with caution.</p> <p>If Bootsect.exe cannot gain exclusive volume access, then the file system may overwrite the boot code before the next reboot. Bootsect.exe always attempts to lock and dismount the volume before each update. When <b>/force</b> is specified, a forced dismount is tried if the initial lock attempt fails. A lock can fail, for example, if files on the destination volume are currently opened by other programs.</p> <p>When successful, a forced dismount enables exclusive volume access and a reliable boot code update even though the initial lock failed. At the same time, a forced dismount invalidates all open handles to files on the destination volume. This can cause unexpected behavior from the programs that opened these files. Therefore, use this option with caution.</p>
<b>/mbr</b>	<p>Updates the master boot record without changing the partition table on sector 0 of the disk that contains the partition specified by <b>SYS</b>, <b>ALL</b>, or &lt;drive letter&gt;. When used with the <b>/nt52</b> option, the master boot record is compatible with operating systems older than Windows Vista. When used with the <b>/nt60</b> option, the master boot record is compatible with Windows 8, Windows 7, Windows Vista, Windows Server 2012, Windows Server 2008 R2, or Windows Server 2008.</p>

## Related topics

[BCDboot Command-Line Options](#)

# Oscdimg Command-Line Options

1/18/2019 • 9 minutes to read

Oscdimg is a command-line tool that you can use to create an image (.iso) file of a customized 32-bit or 64-bit version of Windows Preinstallation Environment (Windows PE). You can then burn the .iso file to a CD or DVD. Oscdimg supports ISO 9660, Joliet, and Universal Disk Format (UDF) file systems.

In this topic:

- [File System Options](#)
- [CD or DVD Boot Options](#)
- [Optimization Options](#)
- [Order Options](#)
- [DVD Video and Audio Options](#)
- [Messaging Options](#)
- [General Image Creation Options](#)
- [Examples](#)

## Oscdimg Command-Line Options

The following command-line options are available for Oscdimg.

**Oscdimg** [<options>] <sourceLocation> <destinationFile>

### File System Options

The Oscdimg tool and Microsoft Windows image mastering API (IMAPI) support three file system formats: ISO 9660, Joliet, and UDF.

#### ISO 9660 Options

ISO 9660 options cannot be combined with Joliet or UDF options. The length of the file name combined with the length of the file name extension cannot exceed 30 characters in the ISO 9660 file system.

The **-d** and **-nt** options cannot be used together.

OPTION	DESCRIPTION
<b>-d</b>	Permits lower case file names. Does not force lowercase file names to upper case.
<b>-n</b>	Permits file names longer than DOS 8.3 file names.
<b>-nt</b>	Permits long file names that are compatible with Windows NT 3.51.

### Joliet Options

Joliet is an extension of the ISO 9660 file system. Joliet allows longer file names, Unicode characters, and directory depths larger than eight. Joliet options cannot be combined with ISO 9660 options.

The **-j2** Joliet option cannot be used with any UDF options.

OPTION	DESCRIPTION
<b>-j1</b>	Permits both file systems to view all the data on the disk. Using this option does not duplicate all files on the image. This option encodes Joliet Unicode file names and generates DOS-compatible 8.3 file names in the ISO 9660 namespace. These file names can be read by either Joliet systems or conventional ISO 9660 systems. However, Oscdimg may change some of the file names in the ISO 9660 namespace to comply with DOS 8.3 and ISO 9660 naming restrictions.
<b>-j2</b>	Encodes Joliet Unicode file names without standard ISO 9660 names. This option is used to produce an image that contains only the Joliet file system. Any system that cannot read Joliet sees only a default text file that alerts the user that this image is only available on computers that support Joliet.
<b>-js</b>	Overrides the default text file that is used when the user specifies the <b>-j2</b> option. For example:

```
-jsc:\readme.txt
```

## UDF Options

UDF options cannot be combined with ISO 9660 options. The **-ue**, **-uf**, and **-us** options only apply when they are used together with the **-u2** option.

OPTION	DESCRIPTION
<b>-u1</b>	Produces an image that has both the UDF file system and the ISO 9660 file system. The ISO 9660 file system is written by using DOS-compatible 8.3 file names. The UDF file system is written by using Unicode file names.
<b>-u2</b>	Produces an image that contains only the UDF file system. Any system that cannot read UDF sees only a default text file that alerts the user that this image is only available on computers that support UDF.
<b>-udfver102</b>	Specifies UDF file system version 1.02.
<b>-ue</b>	Creates embedded files.
<b>-uf</b>	Embeds UDF file identifier entries.

OPTION	DESCRIPTION
<b>-ur</b>	Overrides the default text file that is used together with the <b>-u2</b> option. For example:  -urC:\Readme.txt
<b>-us</b>	Creates sparse files, when available, to make disk space usage more efficient.
<b>-yl</b>	Specifies long allocation descriptors instead of short allocation descriptors.

## CD or DVD Boot Options

Boot options can be used to create bootable CD or DVD images. The following boot options can be used to generate single-boot entries. For more information, see [Use a single boot entry to create a bootable image](#).

OPTION	DESCRIPTION
<b>-b</b> <bootSectorFile>	Specifies the El Torito boot sector file that will be written in the boot sector or sectors of the disk. Do not use spaces. For example:  On UEFI: -bC:\winpe_x86\Efi.sys.bin  On BIOS: -bC:\winpe_x86\Etfsboot.com
<b>-e</b>	Disables floppy disk emulation in the El Torito catalog.
<b>-p</b>	Specifies the value to use for the platform ID in the El Torito catalog. The default ID is 0xEF to represent a Unified Extensible Firmware Interface (UEFI) system. 0x00 represents a BIOS system.
<sourceLocation>	Required. Specifies the location of the files that you intend to build into an .iso image.
<targetFile>	Specifies the name of the .iso image file.

**Important** Single-boot entries and multi-boot entries cannot be combined in the same command.

The following boot options can be used to generate multi-boot entries. For more information, see [Use multi-boot entries to create an image file](#).

OPTION	DESCRIPTION
--------	-------------

OPTION	DESCRIPTION
<b>b</b> <bootSectorFile>	Specifies the El Torito boot sector file that will be written in the boot sector or sectors of the disk. Do not use spaces. For example:  On UEFI: <code>bEfisys.bin</code>  On BIOS: <code>bEtfsboot.com</code>
<b>-bootdata:&lt;number&gt;</b>	Specifies a multi-boot image, followed by the number of boot entries. Do not use spaces. For example:  <code>-bootdata:&lt;3&gt;#&lt;defaultBootEntry&gt;#&lt;bootEntry1&gt;#&lt;bootEntryN&gt;</code>  where <3> is the number of boot entries that follow.
<b>e</b>	Disables floppy disk emulation in the El Torito catalog.
<b>p</b>	Specifies the value to use for the platform ID in the El Torito catalog. The default ID is 0xEF to represent a UEFI system. 0x00 represents a BIOS system.
<b>t</b>	Specifies the El Torito load segment. If not specified, this option defaults to 0x7C0.
<sourceLocation>	Required. Specifies the location of the files that you intend to build into an .iso image.
<targetFile>	Specifies the name of the .iso image file.

## Optimization Options

Optimization options can be used to optimize storage by encoding duplicate files only once.

OPTION	DESCRIPTION
<b>-o</b>	Uses a MD5 hashing algorithm to compare files.
<b>-oc</b>	Uses a binary comparison of each file, and is slower than the <b>-o</b> option.
<b>-oi</b>	Ignores Diamond compression timestamps when comparing files.

## Order Options

Order options specify the file order on disk. The file order does not have to list all files. Any files that do not appear in this file are ordered as they would be ordinarily (that is, if the ordering file did not exist). For more information, see [Specify the boot order](#).

The **-yo** option takes precedence over the **-y5** option.

OPTION	DESCRIPTION
<b>-y5</b>	Specifies file layout on disk. This option writes all files in an i386 directory first and in reverse sort order.
<b>-yo</b> <bootOrder.txt>	Specifies a text file that has a layout for the files to be put in the image. Do not use spaces. For example:  -yoC:\temp\bootOrder.txt

## DVD Video and Audio Options

The DVD video and audio disk creation options cannot be combined with ISO 9660, Joliet, or UDF options.

OPTION	DESCRIPTION
<b>-ut</b>	Truncates the ISO 9660 section of the image during DVD video and audio disk creation. When this option is used, only the VIDEO_TS, AUDIO_TS, and JACKET_P directories are visible from the ISO 9660 file system.
<b>-uv</b>	Specifies UDF Video Zone compatibility during DVD video and audio disk creation. During creation, UDF 1.02 and ISO 9660 are written to the disk. All files in the VIDEO_TS, AUDIO_TS, and JACKET_P directories are written first. These directories take precedence over all other ordering rules that are used for this image.

## Messaging Options

Messaging options customize how file and directory information appears.

OPTION	DESCRIPTION
<b>-a</b>	Displays the allocation summary for files and directories.
<b>-os</b>	Shows duplicate files when the system creates the image.
<b>-w1</b>	Reports all file names or directories that are not ISO-compliant or Joliet-compliant.
<b>-w2</b>	Reports all file names that are not DOS-compliant.
<b>-w3</b>	Reports all zero-length files.
<b>-w4</b>	Reports each file name that is copied to the image.

OPTION	DESCRIPTION
<b>-yd</b>	Suppresses warnings for non-identical files that have the same initial 64,000 bytes.

## General Image Creation Options

General image creation options can be used together with a single-boot entry option or multi-boot entry options to create bootable CD or DVD images. For more information, see [Boot Options](#) and [Examples](#).

The **-m** and **-maxsize** options cannot be used together.

OPTION	DESCRIPTION
<b>-c</b>	Specifies that the system must use ANSI file names instead of OEM file names.
<b>-g</b>	Encodes time values as Universal Coordinated Time (UCT) for all files, instead of the local time.
<b>-h</b>	Includes hidden files and directories in the source path of the image.
<b>-k</b>	Creates an image even if some of the source files cannot be opened.
<b>-l&lt;volumeLabel&gt;</b>	Specifies the volume label. Do not use spaces. For example:  -l<volumeLabel>
<b>-m</b>	Ignores the maximum size limit of an image.
<b>-maxsize:&lt;limit&gt;</b>	Overrides the default maximum size of an image. The default value is a 74-minute CD. However, if UDF is used, the default has no maximum size. Do not use spaces. For example:  -maxsize:<4096>  where <4096> limits the image to 4096 MB.
<b>-q</b>	Scans the source files only. This option does not create an image.
<b>-r</b>	New for Windows 8. Resolves symbolic links to their target location.

OPTION	DESCRIPTION
<b>-t&lt;mm/dd/yyyy,hh:mm:ss&gt;</b>	Specifies the timestamp for all files and directories. Do not use spaces. You can use any delimiter between the items. For example:  -t12/31/2000,15:01:00
<b>-y6</b>	Specifies that directory records must be exactly aligned at the end of sectors.
<b>-yw</b>	Opens source files that have write sharing.

## Examples

These examples illustrate how to do the following:

- Create a bootable CD or DVD for a UEFI-based computer by using a single-boot entry.
- Create a bootable CD or DVD for a UEFI-based or BIOS-based computer by using a multi-boot entry.
- Specify the boot file order on a disk.

### Use a single-boot entry to create a bootable image

You can use the Oscdimg tool to create a bootable CD or DVD by using a single-boot entry.

#### To use a single-boot entry

- Create an image file for a UEFI-based computer. For example:

```
Oscdimg -bC:\winpe_amd64\Efisys.bin -pEF -u1 -udfver102 C:\winpe_amd64\media
C:\winpe_amd64\winpeamd64.iso
```

where C:\winpe\_amd64\media is the location of the source files, and C:\winpe\_amd64\winpeamd64.iso is the path of the .iso file.

### Use multi-boot entries to create a bootable image

You can use the Oscdimg tool to create a bootable CD or DVD by using multi-boot entries. When you do this, note the following:

- The **bootdata** option must be followed by the number of boot entries in the command (-**bootdata:<number>**).
- Each multi-boot entry must be delimited by using a hash symbol (#).
- Each option for a boot entry must be delimited by using a comma (,).
- Each boot entry must specify the platform ID.

#### To use multi-boot entries

- Create an image file for either a UEFI-based or BIOS-based computer by using a multi-boot command. For example:

```
Oscdimg -bootdata:2#p0,e,bEtfssboot.com#pEF,e,bEfisys.bin -u1  
-udfver102 C:\winpe_amd64\media C:\winpe_amd64\winpeamd64.iso
```

where this command starts the Etfssboot.com boot file for a BIOS image, and then starts the Efisys.bin boot file for a UEFI image.

### Specify the boot order

For images larger than 4.5 GB, you must create a boot order file to make sure that boot files are located at the beginning of the image.

The rules for file ordering are as follows:

- The order file must be in ANSI.
- The order file must end in a new line.
- The order file must have one file per line.
- Each file must be specified relative to the root of the image.
- Each file must be specified as a long file name. No short names are allowed.
- Each file path cannot be longer than MAX\_PATH. This includes the volume name.

For example, D:\cdimage would resemble the following (where D is the drive letter of the DVD drive):

- D:\cdimage\1\1.txt
- D:\cdimage\2\2.txt
- D:\cdimage\3\3.txt
- D:\cdimage\3\3\_5.txt
- D:\cdimage\<longFileName>.txt

### To create a boot order file

- Create a boot order file. For example:

```
Oscdimg -m -n -yoC:\temp\bootOrder.txt  
-bC:\winpe_amd64\Efisys.bin C:\winpe_amd64\winpeamd64.iso
```

where BootOrder.txt contains the following list of files:

```
boot\bcd  
boot\boot.sdi  
boot\bootfix.bin  
boot\bootsect.exe  
boot\etfsboot.com  
boot\memtest.efi  
boot\memtest.exe  
boot\en-us\bootsect.exe.mui  
boot\fonts\chs_boot.ttf  
boot\fonts\cht_boot.ttf  
boot\fonts\jpn_boot.ttf  
boot\fonts\kor_boot.ttf  
boot\fonts\wg14_boot.ttf  
sources\boot.wim
```

## Related topics

[WinPE: Create USB Bootable drive](#)

[Windows Deployment Command-Line Tools Reference](#)

# Mobile manufacturing

1/25/2019 • 5 minutes to read

After you have completed the steps covered in the other guides to prepare the device, the focus shifts to preparing the device for the final retail configuration.

During this process, you set the final configuration values, remove debug logging, and optimize the OS for shipment. Next, you determine how the OS will be transferred to the device hardware in the manufacturing line.

## Manufacturing acronyms

Here are some common acronyms that might come in handy.

<b>ATE</b>	automated test equipment
<b>BER</b>	bit error rate
<b>BIST</b>	built-in self-test
<b>COT</b>	cost of test
<b>CIT Testing</b>	computer interactive testing—semi-automated testing of the device. During this stage, the device is connected to a PC or workstation
<b>DIB</b>	device interface board
<b>DFT</b>	designed for test
<b>DUT</b>	device under test
<b>ESD</b>	electrostatic discharge
<b>EVM</b>	error vector magnitude
<b>FA</b>	final assembly
<b>FQC</b>	final quality check
<b>NIST</b>	National Institute of Standards and Technology

<b>OOBT</b>	out-of-the-box test
<b>PIB</b>	probe interface board
<b>RTC</b>	real-time clock—on-board hardware clock used to track the current time
<b>SCM</b>	subcontract manufacturer
<b>SOC</b>	system on a chip
<b>UPH</b>	units per hour
<b>UUT</b>	unit under test

## General manufacturing guidance

The goal for Windows 10 Mobile is that partners are successful in establishing efficient and effective processes that span manufacturing, testing, and servicing. To that end, Microsoft will provide guidance on the tools and process that are used for manufacturing and support of a Windows 10 Mobile device. This guidance describes the tools and techniques that are available to OEMs during the manufacturing process.

- [Manufacturing workflow](#)
- [Example test area by manufacturing phase](#)
- [Using a host computer to reboot a phone to flashing mode and get version information](#)

## Manufacturing security requirements

Final retail images must be configured to meet a set of security requirements. To help OEMs ensure that their retail images meet these requirements, Windows 10 Mobile automatically checks for some of these requirements during first boot. Other requirements must be verified by OEMs.

## Mobile deployment and imaging

Getting ready to build and test Windows 10 for mobile editions? Here's a lab that walks through building new mobile devices and customizing them to meet your customers' needs.

- [Mobile deployment and imaging](#)

## Manufacturing mode of the full operating system

Manufacturing mode is a mode of the full operating system that can be used for manufacturing-related tasks, such as component and support testing.

- [Manufacturing Mode](#)
- [Boot mode management UEFI protocol](#)

# Microsoft Manufacturing OS (MMOS)

Microsoft Manufacturing OS (MMOS) is an optimized configuration of the operating system that facilitates efficient manufacturing.

- Microsoft Manufacturing OS
- MMOS image definition
- Flash MMOS to the device
- Develop MMOS test applications
- Manufacturing test environment supported APIs
- Deploy and test a user-mode test application in MMOS
- Working with WIM MMOS images

## Flashing tools

You can develop a custom flashing tool to address the life cycle needs of the device

- Flashing tools
- Developing custom OEM flashing tools

## Manufacturing workflow

OEMs need to determine the manufacturing process to use to implement MMOS in their manufacturing facilities.

To discuss the manufacturing process, a simplified model of the manufacturing line workflow will be used. Note that each OEM will have a unique process; this simplified model is used as a common reference point.

### Simple factory flow



**Board tests/SMT** – Image is flashed via gang programmer.

**Final assembly, Boot** – Marry board with plastic; the first time the device is booted on the manufacturing floor.

**Manual tests** – Line worker runs device tests such as sound, vibration, camera, keyboard, and so on.

**RF/Call testing** – Automated testing in which the device is tethered to enable power and the recording of test data.

**Final provisioning** – Automated process where IMEI data is written, customizations are loaded, and labeling is completed.

**Final QA/Packaging** – Final manual verification of the device, then packaging.

**Random sample testing** – A specified number of devices are removed from packaging and tested. If failures reach a certain threshold, the entire line may be recalled.

## Manufacturing process options

Each manufacturer has different techniques and tooling that they use to manufacture Windows 10 Mobile devices. Two options are described here, but the OEM is encouraged to combine approaches and innovate as needed. The best technical expertise regarding manufacturing resides with those who built the OEM manufacturing line. Select the guidance that works for your manufacturing processes and business.

A summary of two example manufacturing processes is provided here.

### Manufacturing process option 1: boot from WIM MMOS image

You can temporarily copy a WIM (Windows Imaging) Microsoft Manufacturing OS (MMOS) image over to a device and then boot to that image that is running in volatile RAM memory. For more information about MMOS, see [Microsoft Manufacturing OS](#).

Because the test WIM MMOS image that will never ship is used on the device, this approach enables the provisioning of additional manufacturing tools (transports, extenders). Also, OEMs can use any functional native APIs in the test image. Additional factory test-only drivers or other software can be included in the WIM MMOS image.

One advantage of using this approach is that booting from a WIM Image in RAM is faster than flashing the image.

Simple factory flow



WIM MMOS factory flow



For more info, see [Working with WIM MMOS images](#).

### Manufacturing process option 2: reflash the device

This process uses two images: one for factory testing, and a final golden image for the final shipping device. Like the MMOS WIM image, the test image that will never ship is used on the device. This means that additional manufacturing tools (transports, extenders) can be included and manufacturing only native APIs can be called.

Simple factory flow



Reflash factory flow



This approach makes it possible, if appropriate, to use a different version of the operating system for testing. This

may be a useful interim measure as test applications are being migrated to Windows 10 Mobile.

One tradeoff in this approach is that the manufacturing line must be designed to accommodate the reflashing time that occurs near the end of the manufacturing process.

For more info on working with the flashing tools, see [Flashing tools](#).

## Example test area by manufacturing phase

Test area by manufacturing phase is provided only as an example; each manufacturer may wish to sequence the tests differently.

### **Modem testing (RF / cellular testing)**

Modem RF calibration

Wi-Fi RX/TX Power

Bluetooth RX/TX Power

### **Device testing**

Display

Keypad

SIM interface

Storage card

Camera

RTC

Speaker

Microphone

Sensor – ALS

Sensor – Magnetometer

Sensor – Proximity

Sensor – Accelerometer

Handset interface

Power - stand by current

## **Device provisioning**

IMEI

SIM Lock

Bluetooth MAC

Sensors calibration

Security provisioning

MO provisioning

# Mobile deployment and imaging

1/25/2019 • 2 minutes to read

Windows 10 Mobile brings the features available in Windows 10 to mobile devices. In addition, on devices that meet the required hardware components, Windows 10 Mobile also adds features like Continuum for Phones, which allows users to connect their Windows phone to a monitor, and even to a mouse and keyboard, to make the phone work like a desktop.

## Intended audience

The entirety, or sections, of this walkthrough is intended for use by:

- New or experienced Windows OEMs and ODMs who want to build and deploy a customized Windows 10 Mobile image.
- Mobile operators who want to know the process of building and deploying a customized mobile image.

## Overview

This mobile deployment and imaging guide is organized based on the two ways that you can customize, build, and flash a Windows image to a mobile device:

**Step 1: Prepare for Windows mobile development** provides information about the prerequisites, tools, and how to set up your development environment.

**Step 2: Create mobile packages** provides step-by-step information on how to create a mobile package using a sample driver and when declaring an MCSF setting by using the MCSF settings schema within a .pkg.xml.

**Step 3: Configure the Start layout** shows how you can customize the Start layout on mobile devices to include Web links, secondary tiles, folders, and so on. Start layout uses a new unified schema in win\_threshold so it doesn't matter if you are using the Start MCSF settings or the Start Windows Provisioning settings to add the layout to your image. Also note that the default layout on mobile devices can only be customized as part of the imaging process.

**Step 4:** Once you've done all the preparation steps, you're ready to start customizing and building your image. We recommend learning the classic mobile deployment process first because MCSF still provides a more robust set of customizations that OEMs and ODMs can configure for their image. However, Windows Provisioning offers many enterprise policies, enrollment and enterprise settings not available through MCSF so we recommend becoming familiar with the Windows Provisioning deployment process too.

- [Part 1: Classic mobile deployment](#)

Configure customizations that are only available through the Managed Centralized Settings Framework (MCSF) and use the classic Windows mobile command-line tools to build packages and a customized image, and then flash the image to a device.

- [Part 2: Mobile deployment using Windows Provisioning](#)

Use the Windows Provisioning answer file (WPAF) to configure customization settings that are only available through the Windows Provisioning framework. Use the WPAF with an MCSF CAF as inputs to the Windows Imaging and Configuration Designer (ICD) command-line interface to build a customized mobile image.

# Prepare for Windows mobile development

1/25/2019 • 7 minutes to read

Here's what you'll need to start customizing, testing, and deploying Windows on mobile devices.

## Step 1: Meet all the prerequisites

Before you can get started on your Windows mobile development, make sure you meet these requirements:

- You have access to the Microsoft Connect site where mobile partners can download the latest mobile OS kits and packages.

If you don't have access or need more information, contact your Microsoft representative.

- A development workstation or technician computer

This PC will run the tools needed during the development process. The PC must be running one of the following operating systems:

- Windows 10 32-bit (x86) or 64-bit (x64)
- Windows 8.1 32-bit (x86) or 64-bit (x64)
- Windows 7 32-bit (x86) or 64-bit (x64)

You must install all Windows critical updates to avoid any issues when using the mobile kits.

- Uninstall earlier versions of the tools and kits

If you are using a different version of the tools and kits other than the ones listed in the *Pairing information* section of the kit release notes, you first need to uninstall all of the programs associated with these components.

- Download the kits and OS packages

Download the latest kits and tools for Windows 10 Mobile from the Microsoft Connect site. See Contents of the mobile build to learn more about the contents of the MobileOS kit for different silicon architectures.

- Reference mobile hardware

This mobile device should represent all the mobile devices in a single model line; for example, the Contoso Windows Phone. For more information about detailed hardware requirements for any device that runs Windows 10 Mobile, see *Section 2.0 - Minimum hardware requirements for Windows 10 Mobile* in [Minimum hardware requirements](#).

- Board support package (BSP) prerequisites

Make sure you have all the necessary BSP files for your reference hardware. A BSP is a set of files and drivers that Windows 10 Mobile uses to communicate with the hardware on the device, to launch the OS, and to create an OS image that can run on your reference hardware. The SoC vendor provides the BSP for your reference hardware.

## Step 2: Install the tools and development kits

The following kits and tools are used for Windows mobile development:

- Visual Studio 2015, which is your primary development environment for writing apps and drivers.

- Windows 10 Mobile OS, which is contained in the MobileOS package.
- Windows Assessment and Deployment Kit (ADK), which contains the tools you can use for building and customizing your image as well as several other deployment tools that you can use to help you automate a large-scale deployment of Windows.
- Windows Driver Kit (WDK) and Windows 10 Standalone SDK, which you can install separately or install the Enterprise WDK (EWDK), which contains a version of the driver kit and SDK.
- Windows Hardware Lab Kit (HLK), which is a test framework you can use to test hardware devices for Windows.

The following table specifies which development scenarios require each of the kits and tools.

DEVELOPMENT SCENARIO	MOBILEOS KIT	WINDOWS ADK	WINDOWS STANDALONE SDK	WDK	VISUAL STUDIO 2015
Compile code that runs on the mobile device (for example, drivers, services, and apps)	Required	Required	Required	Required	Required
Build packages	Not required	Required	Not required	Not required	Not required
Sign binaries and packages	Not required	Required	Not required	Not required	Not required
Build and customize mobile images on the command line using ImgGen	Required	Not required	Required	Required	Not required
Build and customize mobile images using Windows ICD	Required	Required	Not required	Not required	Not required

## To install the kits and tools

1. Follow the instructions for downloading and installing Visual Studio 2015, the [Windows Driver Kit \(WDK\) 10](#), and the [Windows 10 SDK](#).

**Note** Visual Studio 2015 is only required if you are compiling code that will run on the mobile device, such as drivers and apps.

To confirm that the Windows SDK was properly installed, make sure that the subdirectories listed in the following table exist on your technician PC. Some of these subdirectories may not appear in the kit install directory if you didn't select them from the Windows SDK install wizard.

```
<table>
<colgroup>
<col width="50%" />
<col width="50%" />
</colgroup>
<thead>
<tr class="header">
<th align="left">Windows installation directory tree</th>
<th align="left">Subdirectories within the directory tree</th>
```

```

</tr>
</thead>
<tbody>
<tr class="odd">
<td align="left"><p>For a 32-bit OS: <strong>C:\Program Files\Windows Kits\10</strong></p>
<p>For a 64-bit OS: <strong>C:\Program Files (x86)\Windows Kits\10</strong></p></td>
<td align="left"><ul>
<li><strong>Bin</strong></li>
<li><strong>Catalogs</strong></li>
<li><strong>Debuggers</strong></li>
<li><strong>DesignTime</strong></li>
<li><strong>Include</strong></li>
<li><strong>Lib</strong></li>
<li><strong>Redist</strong></li>
<li><strong>References</strong></li>
<li><strong>Shortcuts</strong></li>
<li><strong>Testing</strong></li>
</ul></td>
</tr>
</tbody>
</table>

```

To confirm that the WDK was properly installed, make sure that the subdirectories listed in the following table exist on your technician PC.

Windows installation directory tree	Subdirectories within the directory tree
<p>For a 32-bit OS: <strong>C:\Program Files\Windows Kits\10</strong></p> <p>For a 64-bit OS: <strong>C:\Program Files (x86)\Windows Kits\10</strong></p>	<ul> <li><strong>Build</strong></li> <li><strong>BuildLabSupport</strong></li> <li><strong>CodeAnalysis</strong></li> <li><strong>CrossCertificates</strong></li> <li><strong>Debug</strong></li> <li><strong>Help</strong></li> <li><strong>Remote</strong></li> <li><strong>Tools</strong></li> </ul>

## 2. Install the Windows 10 ADK.

Ensure that the **Install Path** is set to the kit install directory, **C:\Program Files\Windows Kits\10** (for a 32-bit OS) or **C:\Program Files (x86)\Windows Kits\10** (for a 64-bit OS).

In the **Select the features you want to install** page, select the following:

- **Deployment Tools**
- **Windows Preinstallation Environment (Windows PE)**
- **Imaging and Configuration Designer (ICD)**

- **Configuration Designer**
- **User State Migration Tool (USMT)**

To confirm that the Windows ADK was properly installed, make sure that the **Assessment and Deployment Kit** appears under your Windows installation directory.

3. Download the latest mobile OS kits and packages from the Microsoft Connect site and copy the contents to a local directory on your development workstation.

The kits and packages contain the files and tools you need to build a Windows 10 Mobile image.

4. Extract the packages and files to the kit install location.

- a. Unzip the MobileOS-arm-fre.zip package.
- b. Open the extracted package and copy all the subfolders and their contents to the kit install directory, **C:\Program Files\Windows Kits\10** (for a 32-bit OS) or **C:\Program Files (x86)\Windows Kits\10** (for a 64-bit OS). The folders should be: FieldMedic, FMFiles, MS Packages, OEMCustomization, and OEMInputSamples.

5. In the **OEM** folder, double-click **Setup.exe** to install the following mobile components:

- Windows Phone Driver Kit (WPDK)
- Debugger symbols
- Windows Hardware Lab Kit content for phone
- Test Shell (TShell)
- Core OS packages

## Step 3: Install other tools

You can install some tools separately from the mobile kit. These tools, which are contained in the IHV\_Tools folder, are listed below.

INSTALLER PACKAGE	DESCRIPTION
WP_CPTT_NT-x86-fre.msi	The Windows phone common packaging and test tools.

INSTALLER PACKAGE	DESCRIPTION
WP8KDConn.msi	<p>The KDBG Connectivity package, which contains the following components:</p> <ul style="list-style-type: none"> <li>• Virtual Ethernet tool (VirtEth.exe)</li> <li>• Virtual network driver (the Virtual PC 2007 filter, VmNetSrv), which is required for the Virtual Ethernet tool unless Virtual PC 2007 is already installed; the user will be prompted to install two unsigned drivers.</li> <li>• USB serial drivers (usb2ethernet and usbcompcom)</li> <li>• USB debugger drivers (usb2dbg); the user will be prompted to install one signed driver.</li> </ul> <p>The virtual network driver is installed in the %ProgramFiles%\Microsoft Virtual PC\Utility\VMNetSrv (or %ProgramFiles(x86)%\Microsoft Virtual PC\Utility\VMNetSrv for a 64-bit OS) directory.</p> <p>The other three components are installed in the %ProgramFiles%\Microsoft Windows Phone 8 KDBG Connectivity (or %ProgramFiles(x86)%\Microsoft Windows Phone 8 KDBG Connectivity for a 64-bit OS) directory tree, with the virtual Ethernet tool in the bin subdirectory, the USB serial drivers in the drivers\Usb2Eth subdirectory, and the USB debugger drivers in the drivers\Usb2Dbg subdirectory.</p>
TShell.msi	<p>The Test Shell, which is a set of command-line tools for use in the bring-up and testing of a mobile device. You can use TShell for tasks such as copying files to the mobile device and to view logs.</p>

## Step 4: Set up environment variables

Follow the steps below to set up the environment variables that are required for a working build environment.

### To set up a build environment in Visual Studio 2015

1. Open a **Developer Command Prompt for VS2015** window.
2. Set the WPDKCONTENTROOT environment variable to the location of the Windows 10 Mobile kit installation directory.

- For computers running a 32-bit version of Windows, by default this is %ProgramFiles%\Windows Kits\10.

```
set WPDKCONTENTROOT=%ProgramFiles%\Windows Kits\10
```

- For computers running a 64-bit version of Windows, by default this is %ProgramFiles(x86)%\Windows Kits\10.

```
set WPDKCONTENTROOT=%ProgramFiles(x86)%\Windows Kits\10
```

3. Set the WDKCONTENTROOT environment variable to the location of the WDK kit installation directory.

  - For computers running a 32-bit version of Windows, by default this is %ProgramFiles%\Windows

Kits\10.

```
set WDKCONTENTROOT=%ProgramFiles%\Windows Kits\10
```

- For computers running a 64-bit version of Windows, by default this is %ProgramFiles(x86)%\Windows Kits\10.

```
set WDKCONTENTROOT=%ProgramFiles(x86)%\Windows Kits\10
```

4. Add the x86 tools directory for the Windows kits and the Windows kit tools directory to the PATH environment variable.

```
set PATH=%PATH%;%WDKCONTENTROOT%\bin\x86;%WPKCONTENTROOT%\Tools\bin\i386
```

## Step 5: Install OEM test certs

To sign binaries and packages, you must install the OEM test certificates.

### To install OEM test certs

- After ensuring that WPKCONTENTROOT is set to the path of the kit install location, run InstallOEMCerts.cmd by using the following command:

```
%WPKCONTENTROOT%\tools\bin\i386\InstallOEMCerts.cmd
```

For more information, see [Set up the signing environment](#).

# Create mobile packages

1/25/2019 • 10 minutes to read

In Windows 10 Mobile, packages make up the building blocks for the OS and they can contain all the files, libraries, registry settings, executables, and data on the mobile device. Every OS component, from device drivers to system files, must be contained in a package.

As an OEM, if you have device-specific drivers or customizations that you want to add to the OS, you need to create a package. This section shows you how to add a fictitious driver and an MCSF customization setting as part of the package.

For more detailed information about mobile packages, including specifying other components and package merging, see [Adding mobile packages](#).

## Add a driver component

To learn more about drivers and get started on writing your own, see [Device and Driver Technologies](#).

In this walkthrough, we're downloading the sample echo KMDF driver, converting it to a universal Windows driver, and then using Visual Studio to create a package file.

### To add a driver

1. Download the echo universal driver sample.
  - a. Download the master.zip file and save it to your local hard drive:  
<https://github.com/microsoft/windows-driver-samples/archive/master.zip>.
  - b. Extract all the contents of the master.zip file. Specify a new folder, such as C:\DriverSamples, or browse to an existing one to store the extracted files.
  - c. After the files are extracted, go to the general\echo\kmdf subfolder within the folder where you saved the extracted files to find the driver solution for the echo driver sample. For example, if you created a C:\DriverSamples folder in the previous step, you can locate the driver solution under C:\DriverSamples\general\echo\kmdf.
2. Convert the existing echo driver project to a universal Windows driver project.
  - a. In Visual Studio 2015, open the existing kmdfecho driver project, kmdfecho.sln.
  - b. Location the Solution Explorer in Visual Studio. In the Solution Explorer pane, right-click on **Solution 'kmdfecho' (3 projects)** and choose **Configuration Manager**.
  - c. In the **Configuration Manager** window, set the target operating system to **Windows 10**.
  - d. Right-click on the driver project and then choose **Properties**. Under **Configuration Manager > Driver**, verify that the **Target Platform** is set to **Universal**.
3. From Visual Studio 2015, use PkgGen to generate a package file.
  - a. Right-click on the driver project and choose **Add->New Item**.
  - b. Under **Visual C++ > Windows Driver**, choose **Package Manifest**, and then click **Add**.

Visual Studio adds a file called Package.pkg.xml to your driver project. You can right-click on the file

and choose **Properties** to verify that the item type is set to **PkgGen**. On the same property page, you can set **Excluded from Build** to **Yes** if you decide later that you want to build this driver project and not generate a package file.

- c. Click **OK**.
- d. Right-click on the driver project and choose **Properties**. Under **Configuration Properties**, open the **PackageGen** node and change the **Version** to a any value that you like. For example, you can set the version to *1.0.0.0*.
- e. Set the package **Owner**, **Component**, and **SubComponent** values.

4. Right-click on the driver project and choose **Properties**. Set the test certificates to one of the OEM test certificates that was installed when you ran `installoemcerts.cmd`.

For example, you can use CN=Windows Phone OEM Root 2013 (TEST ONLY).

5. Save your work and then restart Visual Studio as an administrator.
6. Build your driver. Visual Studio links against the required libraries and generates a .cat file, an .inf file, a driver binary, and an .spkg file.

Once the driver is built, you should see a new folder named *ProjectName.spkg*, which contains the .cab and the .spkg.

We will use the driver .spkg that you built in this walkthrough and combining it with the .spkg from the next walkthrough to build a mobile OS image.

To learn about how to run PkgGen.exe outside of Visual Studio, see the next section on adding a customization setting. Also see the *Run the pkggen.exe tool* section in [Creating mobile packages](#).

### Add an MCSF customization setting

In Windows 10 Mobile, there are two supported customization frameworks: Managed Centralized Settings Framework (MCSF) and Windows provisioning. For more information about these frameworks, see [Customizations for mobile devices](#). When it comes to adding a custom OEM customization setting, only MCSF is extendable and available for OEMs to declare their own various mobile OS settings in a simple and consistent way that's similar to the Microsoft customization settings.

To learn how to use MCSF and the package XML schema to declare and access custom OEM settings, see the sections in [Managed Centralized Settings Framework \(MCSF\)](#).

In this walkthrough, we're extending the [Configure Quick actions](#) customization to use the MCSF schema to create a policy setting that exposes the various slots so that they can be easily configured later without having to remember the registry keys and values. To do this, we add the policy settings declarations in a .pkg.xml file and then build this file to create a package.

### To add a customization setting

1. Write the MCSF policy setting that corresponds to the following registry key:

```

$(HKLM.SOFTWARE)\Microsoft\Shell\OEM\QuickActions\Slot\X
Type: REG_SZ
Possible values:
    Microsoft.QuickAction.AllSettings
    Microsoft.QuickAction.Connect
    Microsoft.QuickAction.Note
    Microsoft.QuickAction.Flashlight
    Microsoft.QuickAction.RotationLock
    Microsoft.QuickAction.BatterySaver
    Microsoft.QuickAction.Bluetooth
    Microsoft.QuickAction.WiFi
    Microsoft.QuickAction.AirplaneMode
    Microsoft.QuickAction.Vpn
    Microsoft.QuickAction.Cellular
    Microsoft.QuickAction.MobileHotspot
    Microsoft.QuickAction.Camera
    Microsoft.QuickAction.Brightness
    Microsoft.QuickAction.QuietHours
    Microsoft.QuickAction.Location

```

The X in the registry key should be replaced with the slot number being configured (beginning with 1). Slots are ordered right-to-left. There is a maximum of 5 slots on a large screen mobile device while mobile devices without a large screen have 4 slots.

The following code example shows how the MCSF policy settings for the quick actions can be declared:

```

<SettingsGroup Path="Notifications/QuickActions">
    <!-- Default Quick actions configuration -->
    <Setting Name="QuickActionSlot1" Description="App to place in quick action slot 1.">
        <RegistrySource Path="$(hkml.software)\Microsoft\Shell\OEM\QuickActions\Slot\1" Type="REG_SZ"
Default="Microsoft.QuickAction.AllSettings" />
        <Validate>
            <!-- Shows the available options for the quick action slot -->
            <Option Value="Microsoft.QuickAction.AllSettings" FriendlyName="All settings" />
            <Option Value="Microsoft.QuickAction.Connect" FriendlyName="Connect" />
            <Option Value="Microsoft.QuickAction.Note" FriendlyName="Note" />
            <Option Value="Microsoft.QuickAction.Flashlight" FriendlyName="Flashlight" />
            <Option Value="Microsoft.QuickAction.RotationLock" FriendlyName="Rotation lock" />
            <Option Value="Microsoft.QuickAction.BatterySaver" FriendlyName="Battery saver" />
            <Option Value="Microsoft.QuickAction.Bluetooth" FriendlyName="Bluetooth" />
            <Option Value="Microsoft.QuickAction.WiFi" FriendlyName="Wi-Fi" />
            <Option Value="Microsoft.QuickAction.AirplaneMode" FriendlyName="Airplane mode" />
            <Option Value="Microsoft.QuickAction.Vpn" FriendlyName="VPN" />
            <Option Value="Microsoft.QuickAction.Cellular" FriendlyName="Cellular" />
            <Option Value="Microsoft.QuickAction.MobileHotspot" FriendlyName="Mobile hotspot" />
            <Option Value="Microsoft.QuickAction.Camera" FriendlyName="Camera" />
            <Option Value="Microsoft.QuickAction.Brightness" FriendlyName="Brightness" />
            <Option Value="Microsoft.QuickAction.QuietHours" FriendlyName="Quiet hours" />
            <Option Value="Microsoft.QuickAction.Location" FriendlyName="Location" />
        </Validate>
    </Setting>
    <Setting Name="QuickActionSlot2" Description="App to place in quick action slot 2.">
        <RegistrySource Path="$(hkml.software)\Microsoft\Shell\OEM\QuickActions\Slot\2" Type="REG_SZ"
Default="Microsoft.QuickAction.RotationLock" />
        <Validate>
            <!-- Shows the available options for the quick action slot -->
            <Option Value="Microsoft.QuickAction.AllSettings" FriendlyName="All settings" />
            <Option Value="Microsoft.QuickAction.Connect" FriendlyName="Connect" />
            <Option Value="Microsoft.QuickAction.Note" FriendlyName="Note" />
            <Option Value="Microsoft.QuickAction.Flashlight" FriendlyName="Flashlight" />
            <Option Value="Microsoft.QuickAction.RotationLock" FriendlyName="Rotation lock" />
            <Option Value="Microsoft.QuickAction.BatterySaver" FriendlyName="Battery saver" />
            <Option Value="Microsoft.QuickAction.Bluetooth" FriendlyName="Bluetooth" />
            <Option Value="Microsoft.QuickAction.WiFi" FriendlyName="Wi-Fi" />
            <Option Value="Microsoft.QuickAction.AirplaneMode" FriendlyName="Airplane mode" />

```

```

<Option Value="Microsoft.QuickAction.Vpn" FriendlyName="VPN" />
<Option Value="Microsoft.QuickAction.Cellular" FriendlyName="Cellular" />
<Option Value="Microsoft.QuickAction.MobileHotspot" FriendlyName="Mobile hotspot" />
<Option Value="Microsoft.QuickAction.Camera" FriendlyName="Camera" />
<Option Value="Microsoft.QuickAction.Brightness" FriendlyName="Brightness" />
<Option Value="Microsoft.QuickAction.QuietHours" FriendlyName="Quiet hours" />
<Option Value="Microsoft.QuickAction.Location" FriendlyName="Location" />
</Validate>
</Setting>
<Setting Name="QuickActionSlot3" Description="App to place in quick action slot 3.">
    <RegistrySource Path="$(hkml.software)\Microsoft\Shell\OEM\QuickActions\Slot\3" Type="REG_SZ"
Default="Microsoft.QuickAction.Bluetooth" />
    <Validate>
        <!-- Shows the available options for the quick action slot -->
        <Option Value="Microsoft.QuickAction.AllSettings" FriendlyName="All settings" />
        <Option Value="Microsoft.QuickAction.Connect" FriendlyName="Connect" />
        <Option Value="Microsoft.QuickAction.Note" FriendlyName="Note" />
        <Option Value="Microsoft.QuickAction.Flashlight" FriendlyName="Flashlight" />
        <Option Value="Microsoft.QuickAction.RotationLock" FriendlyName="Rotation lock" />
        <Option Value="Microsoft.QuickAction.BatterySaver" FriendlyName="Battery saver" />
        <Option Value="Microsoft.QuickAction.Bluetooth" FriendlyName="Bluetooth" />
        <Option Value="Microsoft.QuickAction.WiFi" FriendlyName="Wi-Fi" />
        <Option Value="Microsoft.QuickAction.AirplaneMode" FriendlyName="Airplane mode" />
        <Option Value="Microsoft.QuickAction.Vpn" FriendlyName="VPN" />
        <Option Value="Microsoft.QuickAction.Cellular" FriendlyName="Cellular" />
        <Option Value="Microsoft.QuickAction.MobileHotspot" FriendlyName="Mobile hotspot" />
        <Option Value="Microsoft.QuickAction.Camera" FriendlyName="Camera" />
        <Option Value="Microsoft.QuickAction.Brightness" FriendlyName="Brightness" />
        <Option Value="Microsoft.QuickAction.QuietHours" FriendlyName="Quiet hours" />
        <Option Value="Microsoft.QuickAction.Location" FriendlyName="Location" />
    </Validate>
</Setting>
<Setting Name="QuickActionSlot4" Description="App to place in quick action slot 4.">
    <RegistrySource Path="$(hkml.software)\Microsoft\Shell\OEM\QuickActions\Slot\4" Type="REG_SZ"
Default="Microsoft.QuickAction.WiFi" />
    <Validate>
        <!-- Shows the available options for the quick action slot -->
        <Option Value="Microsoft.QuickAction.AllSettings" FriendlyName="All settings" />
        <Option Value="Microsoft.QuickAction.Connect" FriendlyName="Connect" />
        <Option Value="Microsoft.QuickAction.Note" FriendlyName="Note" />
        <Option Value="Microsoft.QuickAction.Flashlight" FriendlyName="Flashlight" />
        <Option Value="Microsoft.QuickAction.RotationLock" FriendlyName="Rotation lock" />
        <Option Value="Microsoft.QuickAction.BatterySaver" FriendlyName="Battery saver" />
        <Option Value="Microsoft.QuickAction.Bluetooth" FriendlyName="Bluetooth" />
        <Option Value="Microsoft.QuickAction.WiFi" FriendlyName="Wi-Fi" />
        <Option Value="Microsoft.QuickAction.AirplaneMode" FriendlyName="Airplane mode" />
        <Option Value="Microsoft.QuickAction.Vpn" FriendlyName="VPN" />
        <Option Value="Microsoft.QuickAction.Cellular" FriendlyName="Cellular" />
        <Option Value="Microsoft.QuickAction.MobileHotspot" FriendlyName="Mobile hotspot" />
        <Option Value="Microsoft.QuickAction.Camera" FriendlyName="Camera" />
        <Option Value="Microsoft.QuickAction.Brightness" FriendlyName="Brightness" />
        <Option Value="Microsoft.QuickAction.QuietHours" FriendlyName="Quiet hours" />
        <Option Value="Microsoft.QuickAction.Location" FriendlyName="Location" />
    </Validate>
</Setting>
<Setting Name="QuickActionSlot5" Description="App to place in quick action slot 5.">
    <RegistrySource Path="$(hkml.software)\Microsoft\Shell\OEM\QuickActions\Slot\5" Type="REG_SZ"
Default="Microsoft.QuickAction.Camera" />
    <Validate>
        <!-- Shows the available options for the quick action slot -->
        <Option Value="Microsoft.QuickAction.AllSettings" FriendlyName="All settings" />
        <Option Value="Microsoft.QuickAction.Connect" FriendlyName="Connect" />
        <Option Value="Microsoft.QuickAction.Note" FriendlyName="Note" />
        <Option Value="Microsoft.QuickAction.Flashlight" FriendlyName="Flashlight" />
        <Option Value="Microsoft.QuickAction.RotationLock" FriendlyName="Rotation lock" />
        <Option Value="Microsoft.QuickAction.BatterySaver" FriendlyName="Battery saver" />
        <Option Value="Microsoft.QuickAction.Bluetooth" FriendlyName="Bluetooth" />
        <Option Value="Microsoft.QuickAction.WiFi" FriendlyName="Wi-Fi" />
        <Option Value="Microsoft.QuickAction.AirplaneMode" FriendlyName="Airplane mode" />

```

```

<Option Value="Microsoft.QuickAction.Vpn" FriendlyName="VPN" />
<Option Value="Microsoft.QuickAction.Cellular" FriendlyName="Cellular" />
<Option Value="Microsoft.QuickAction.MobileHotspot" FriendlyName="Mobile hotspot" />
<Option Value="Microsoft.QuickAction.Camera" FriendlyName="Camera" />
<Option Value="Microsoft.QuickAction.Brightness" FriendlyName="Brightness" />
<Option Value="Microsoft.QuickAction.QuietHours" FriendlyName="Quiet hours" />
<Option Value="Microsoft.QuickAction.Location" FriendlyName="Location" />
</Validate>
</Setting>
</SettingsGroup>

```

2. Create a .pkg.xml file and add the policy settings in the previous step to the .pkg.xml file. The following example shows how to do this.

```

<?xml version="1.0" encoding="utf-8">
<Package xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  Owner=""
  Component=""
  SubComponent=""
  OwnerType="OEM"
  ReleaseType="">
  <Components>
    <SettingsGroup Path="Notifications/QuickActions">
      <!-- Default Quick actions configuration -->
      <Setting Name="QuickActionSlot1" Description="App to place in quick action slot 1.">
        <RegistrySource Path="$(hkml.software)\Microsoft\Shell\OEM\QuickActions\Slot\1" Type="REG_SZ"
Default="Microsoft.QuickAction.AllSettings" />
        <Validate>
          <!-- Shows the available options for the quick action slot -->
          <Option Value="Microsoft.QuickAction.AllSettings" FriendlyName="All settings" />
          <Option Value="Microsoft.QuickAction.Connect" FriendlyName="Connect" />
          <Option Value="Microsoft.QuickAction.Note" FriendlyName="Note" />
          <Option Value="Microsoft.QuickAction.Flashlight" FriendlyName="Flashlight" />
          <Option Value="Microsoft.QuickAction.RotationLock" FriendlyName="Rotation lock" />
          <Option Value="Microsoft.QuickAction.BatterySaver" FriendlyName="Battery saver" />
          <Option Value="Microsoft.QuickAction.Bluetooth" FriendlyName="Bluetooth" />
          <Option Value="Microsoft.QuickAction.WiFi" FriendlyName="Wi-Fi" />
          <Option Value="Microsoft.QuickAction.AirplaneMode" FriendlyName="Airplane mode" />
          <Option Value="Microsoft.QuickAction.Vpn" FriendlyName="VPN" />
          <Option Value="Microsoft.QuickAction.Cellular" FriendlyName="Cellular" />
          <Option Value="Microsoft.QuickAction.MobileHotspot" FriendlyName="Mobile hotspot" />
          <Option Value="Microsoft.QuickAction.Camera" FriendlyName="Camera" />
          <Option Value="Microsoft.QuickAction.Brightness" FriendlyName="Brightness" />
          <Option Value="Microsoft.QuickAction.QuietHours" FriendlyName="Quiet hours" />
          <Option Value="Microsoft.QuickAction.Location" FriendlyName="Location" />
        </Validate>
      </Setting>
      <Setting Name="QuickActionSlot2" Description="App to place in quick action slot 2.">
        <RegistrySource Path="$(hkml.software)\Microsoft\Shell\OEM\QuickActions\Slot\2" Type="REG_SZ"
Default="Microsoft.QuickAction.RotationLock" />
        <Validate>
          <!-- Shows the available options for the quick action slot -->
          <Option Value="Microsoft.QuickAction.AllSettings" FriendlyName="All settings" />
          <Option Value="Microsoft.QuickAction.Connect" FriendlyName="Connect" />
          <Option Value="Microsoft.QuickAction.Note" FriendlyName="Note" />
          <Option Value="Microsoft.QuickAction.Flashlight" FriendlyName="Flashlight" />
          <Option Value="Microsoft.QuickAction.RotationLock" FriendlyName="Rotation lock" />
          <Option Value="Microsoft.QuickAction.BatterySaver" FriendlyName="Battery saver" />
          <Option Value="Microsoft.QuickAction.Bluetooth" FriendlyName="Bluetooth" />
          <Option Value="Microsoft.QuickAction.WiFi" FriendlyName="Wi-Fi" />
          <Option Value="Microsoft.QuickAction.AirplaneMode" FriendlyName="Airplane mode" />
          <Option Value="Microsoft.QuickAction.Vpn" FriendlyName="VPN" />
          <Option Value="Microsoft.QuickAction.Cellular" FriendlyName="Cellular" />
          <Option Value="Microsoft.QuickAction.MobileHotspot" FriendlyName="Mobile hotspot" />
          <Option Value="Microsoft.QuickAction.Camera" FriendlyName="Camera" />
          <Option Value="Microsoft.QuickAction.Brightness" FriendlyName="Brightness" />
        </Validate>
      </Setting>
    </SettingsGroup>
  </Components>
</Package>

```

```

        <Option Value="Microsoft.QuickAction.QuietHours" FriendlyName="Quiet hours" />
        <Option Value="Microsoft.QuickAction.Location" FriendlyName="Location" />
    </Validate>
</Setting>
<Setting Name="QuickActionSlot3" Description="App to place in quick action slot 3.">
    <RegistrySource Path="$(hkml.software)\Microsoft\Shell\OEM\QuickActions\Slot\3" Type="REG_SZ"
Default="Microsoft.QuickAction.Bluetooth" />
    <Validate>
        <!-- Shows the available options for the quick action slot -->
        <Option Value="Microsoft.QuickAction.AllSettings" FriendlyName="All settings" />
        <Option Value="Microsoft.QuickAction.Connect" FriendlyName="Connect" />
        <Option Value="Microsoft.QuickAction.Note" FriendlyName="Note" />
        <Option Value="Microsoft.QuickAction.Flashlight" FriendlyName="Flashlight" />
        <Option Value="Microsoft.QuickAction.RotationLock" FriendlyName="Rotation lock" />
        <Option Value="Microsoft.QuickAction.BatterySaver" FriendlyName="Battery saver" />
        <Option Value="Microsoft.QuickAction.Bluetooth" FriendlyName="Bluetooth" />
        <Option Value="Microsoft.QuickAction.WiFi" FriendlyName="Wi-Fi" />
        <Option Value="Microsoft.QuickAction.AirplaneMode" FriendlyName="Airplane mode" />
        <Option Value="Microsoft.QuickAction.Vpn" FriendlyName="VPN" />
        <Option Value="Microsoft.QuickAction.Cellular" FriendlyName="Cellular" />
        <Option Value="Microsoft.QuickAction.MobileHotspot" FriendlyName="Mobile hotspot" />
        <Option Value="Microsoft.QuickAction.Camera" FriendlyName="Camera" />
        <Option Value="Microsoft.QuickAction.Brightness" FriendlyName="Brightness" />
        <Option Value="Microsoft.QuickAction.QuietHours" FriendlyName="Quiet hours" />
        <Option Value="Microsoft.QuickAction.Location" FriendlyName="Location" />
    </Validate>
</Setting>
<Setting Name="QuickActionSlot4" Description="App to place in quick action slot 4.">
    <RegistrySource Path="$(hkml.software)\Microsoft\Shell\OEM\QuickActions\Slot\4" Type="REG_SZ"
Default="Microsoft.QuickAction.WiFi" />
    <Validate>
        <!-- Shows the available options for the quick action slot -->
        <Option Value="Microsoft.QuickAction.AllSettings" FriendlyName="All settings" />
        <Option Value="Microsoft.QuickAction.Connect" FriendlyName="Connect" />
        <Option Value="Microsoft.QuickAction.Note" FriendlyName="Note" />
        <Option Value="Microsoft.QuickAction.Flashlight" FriendlyName="Flashlight" />
        <Option Value="Microsoft.QuickAction.RotationLock" FriendlyName="Rotation lock" />
        <Option Value="Microsoft.QuickAction.BatterySaver" FriendlyName="Battery saver" />
        <Option Value="Microsoft.QuickAction.Bluetooth" FriendlyName="Bluetooth" />
        <Option Value="Microsoft.QuickAction.WiFi" FriendlyName="Wi-Fi" />
        <Option Value="Microsoft.QuickAction.AirplaneMode" FriendlyName="Airplane mode" />
        <Option Value="Microsoft.QuickAction.Vpn" FriendlyName="VPN" />
        <Option Value="Microsoft.QuickAction.Cellular" FriendlyName="Cellular" />
        <Option Value="Microsoft.QuickAction.MobileHotspot" FriendlyName="Mobile hotspot" />
        <Option Value="Microsoft.QuickAction.Camera" FriendlyName="Camera" />
        <Option Value="Microsoft.QuickAction.Brightness" FriendlyName="Brightness" />
        <Option Value="Microsoft.QuickAction.QuietHours" FriendlyName="Quiet hours" />
        <Option Value="Microsoft.QuickAction.Location" FriendlyName="Location" />
    </Validate>
</Setting>
<Setting Name="QuickActionSlot5" Description="App to place in quick action slot 5.">
    <RegistrySource Path="$(hkml.software)\Microsoft\Shell\OEM\QuickActions\Slot\5" Type="REG_SZ"
Default="Microsoft.QuickAction.Camera" />
    <Validate>
        <!-- Shows the available options for the quick action slot -->
        <Option Value="Microsoft.QuickAction.AllSettings" FriendlyName="All settings" />
        <Option Value="Microsoft.QuickAction.Connect" FriendlyName="Connect" />
        <Option Value="Microsoft.QuickAction.Note" FriendlyName="Note" />
        <Option Value="Microsoft.QuickAction.Flashlight" FriendlyName="Flashlight" />
        <Option Value="Microsoft.QuickAction.RotationLock" FriendlyName="Rotation lock" />
        <Option Value="Microsoft.QuickAction.BatterySaver" FriendlyName="Battery saver" />
        <Option Value="Microsoft.QuickAction.Bluetooth" FriendlyName="Bluetooth" />
        <Option Value="Microsoft.QuickAction.WiFi" FriendlyName="Wi-Fi" />
        <Option Value="Microsoft.QuickAction.AirplaneMode" FriendlyName="Airplane mode" />
        <Option Value="Microsoft.QuickAction.Vpn" FriendlyName="VPN" />
        <Option Value="Microsoft.QuickAction.Cellular" FriendlyName="Cellular" />
        <Option Value="Microsoft.QuickAction.MobileHotspot" FriendlyName="Mobile hotspot" />
        <Option Value="Microsoft.QuickAction.Camera" FriendlyName="Camera" />
        <Option Value="Microsoft.QuickAction.Brightness" FriendlyName="Brightness" />

```

```
<Option Value="Microsoft.QuickAction.QuietHours" FriendlyName="Quiet hours" />
<Option Value="Microsoft.QuickAction.Location" FriendlyName="Location" />
</Validate>
</Setting>
</SettingsGroup>
</Components>
</Package>
```

3. Add values for the **Owner**, **Component**, **SubComponent**, and **ReleaseType** elements. For example:

- **Owner**="Contoso"
- **Component**="Customization.Notifications"
- **SubComponent**="QuickActions"
- **ReleaseType**="Test"

To learn more about the elements and attributes, see [Primary elements and attributes of a package project file](#).

4. Name and save the .pkg.xml file as "QuickActions.pkg.xml".

5. Generate a package, or .spkg file, using the "QuickActions.pkg.xml" as input. For more information, see *Run the pkggen.exe tool* in [Adding mobile packages](#).

#### To generate a package using QuickActions.pkg.xml

- a. Open a command line with administrator privileges.
- b. Enter the following command to build a package from QuickActions.pkg.xml.

```
PkgGen QuickActions.pkg.xml
/config:"%WPKCONTENTROOT%\Tools\bin\i386\pkggen.cfg.xml"
```

This command will generate a package called Customization.Notifications.QuickActions.spkg. In the next section, we will use this package and add it to a feature manifest file.

# Adding mobile packages

1/25/2019 • 8 minutes to read

Packages are the logical building blocks of Windows 10 Mobile. They contain all the files, libraries, registry settings, executables, and data on the device. From device drivers to system files, every component must be contained in a package. This modular architecture allows for precise control of updates: a package is the smallest serviceable unit on the device. Packages usually represent a specific feature or functionality in the operating system. Individual packages can be grouped together to form more manageable groups of components and used to create images for flashing or updating a device.

## Related topics

The following list shows additional topics related to package creation:

- [Primary elements and attributes of a package project file](#)
- [Specifying components in a package project file](#)
- [Specifying files and registry entries in a package project file](#)
- [Command-line arguments for package generator](#)
- [Merging packages before imaging](#)
- [Merging packages using FeatureMerger](#)

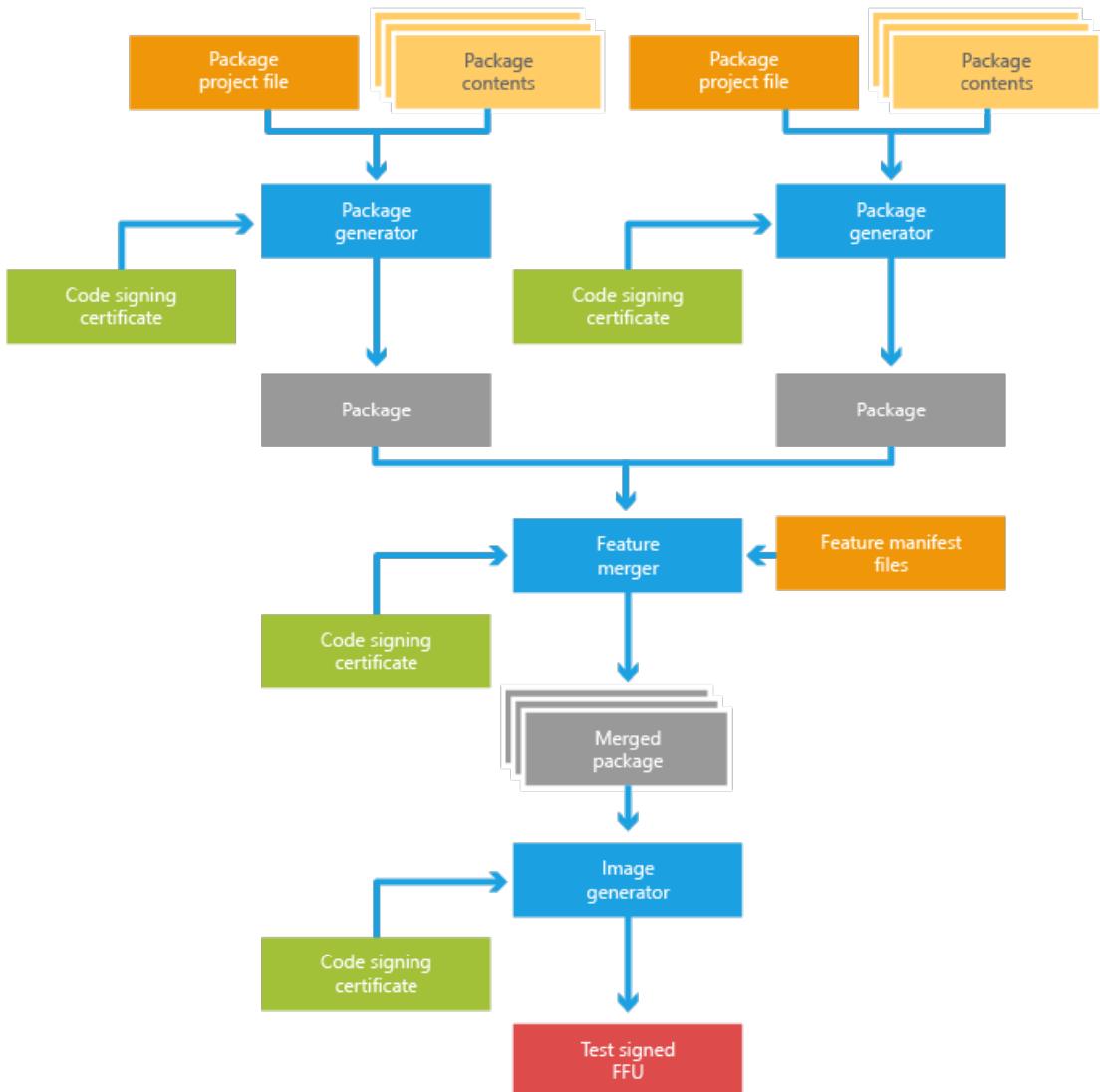
## Packaging process during development

It is important to follow the packaging model to ensure the operating system behaves properly. If OEM packages are incorrectly defined, a device that functions normally during development may stop working correctly when a software update is applied.

Packages fall into three primary categories:

- Microsoft prebuilt packages that are delivered with mobileos and contain the core operating system
- SoC vendor prebuilt packages that are delivered by the vendor and contain the drivers and firmware that support the device's chipset
- OEM packages that are built by partners to contain their device-specific drivers and customizations

The following diagram illustrates the packaging process during development.



# Create a package project with no content

A package project XML file generally starts with no content. The following is an example of an empty package project.

## **Important**

The package project XML file must use the "pkg.xml" extension.

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
    Owner="OEMName"
    OwnerType="OEM"
    ReleaseType="Test"
    Platform="PlatformName"
    Component="ComponentName"
    SubComponent="SubName">
</Package>
```

If you run the package generator (pkgen.exe) against this project file, a package with no contents will be created.

## Add content to a package

The contents of a package are organized as a list of XML elements in the package project XML file. To add

contents to a package, add a **Components** element with the appropriate child elements as shown in the following excerpt that demonstrates the addition of some files and registry settings to a package.

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
    Owner="OEMName"
    OwnerType="OEM"
    ReleaseType="Test"
    Platform="PlatformName"
    Component="ComponentName"
    SubComponent="SubName">
    <Components>
        <OSComponent>
            <Files>
                <File Source="$( _RELEASEDIR )\test_file1.dll" />
                <File Source="$( _RELEASEDIR )\toBeRenamed.dat"
                    DestinationDir="$( runtime.system32 )\test" Name="test.dat" />
            </Files>
            <RegKeys>
                <RegKey KeyName="$( hk1m.software )\OEMName\test">
                    <RegValue Name="StringValue" Value="Test string" Type="REG_SZ" />
                    <RegValue Name="DWordValue" Value="12AB34CD" Type="REG_DWORD" />
                    <RegValue Name="BinaryValue" Value="12,AB,CD,EF" Type="REG_BINARY" />
                </RegKey>
                <RegKey KeyName="$( hk1m.software )\OEMName\EmptyKey" />
            </RegKeys>
        </OSComponent>
    </Components>
</Package>
```

This example demonstrates how to specify the file source path and override the default path and name on the device. It also demonstrates how to specify different registry values. If you run the package generator (pkggen.exe) against this project file, it will create a package that contains the specified files and registry values. For more info about how to run package generator, see [Command-line arguments for package generator](#).

You can also add other objects such as COM servers and drivers. For additional schema and attribute information for each type of object, see [Specifying components in a package project file](#).

Relative DestinationDir references using a "." or ".." are not supported. Use absolute directory references instead.

## Run the pkggen.exe tool

The package generator tool (pkggen.exe) provided with the Windows Driver Kit and is used to build a package and generate a .spkg file for inclusion in the operating system image. This tool is installed by default to %WPDKCONTENTROOT%\Tools\bin\i386.

### To use the package generator tool

1. Configure your environment to specify a certificate for package signing using the guidance provided in [Set up the signing environment](#) and [Sign binaries and packages](#).
2. Add %WPDKCONTENTROOT%\Tools\bin\i386 to your Path environment variable.
3. From an elevated command prompt, enter the following command for each of the pkg.xml files you want to build.

```
PkgGen [package project file name] /config:"%WPDKCONTENTROOT%\Tools\bin\i386\pkggen.cfg.xml"
```

**Note** When a package is changed the /version field should always be incremented when running package generator.

For additional info about package generator options and capabilities, see [Command-line arguments for package generator](#).

## View the contents of a package

Packages use Windows cabinet file technology to store a set of files.

### To view the contents of a package

1. On a Windows PC, locate the generated package and add a ".cab" file extension to the package name.
2. Double-click the renamed file to view the files that are stored in the .cab file.
3. The package files are renamed in the .cab file. To view the original filenames, extract the man.dsm.xml file to the Desktop and open it.
4. There are multiple file entries. For each file entry, the <DevicePath> element shows the full file name. The <CabPath> entry shows the file name that is used in the package.

```
<FileEntry>
<FileType>Regular</FileType>
<DevicePath>\Windows\Packages\CustomMetadata\Contoso.PhoneTest.TestApp.meta.xml</DevicePath>
<CabPath>4_Contoso.xml</CabPath>
<Attributes>Normal</Attributes>
</FileEntry>
```

5. After you are done viewing the files, remove the ".cab" file extension from the package file name.

## Add a language-specific content to a package

In the preceding example, all the files and registry values are language neutral. You can use the package project XML file to add language-related files and registry values to a package. Special flags are used to notify the package generator of language-specific content. The following XML example demonstrates how to designate language-specific content.

```
<Components>
<OSComponent>
...
<Files Language="*">
    <File DestinationDir="$(runtime.default)\mui\$(langid)"
        Source="$_RELEASEDIR\$(LANGID)\test.dll.mui"/>
</Files>
<Files Language="(zh-CN;zh-TW)">
    <File DestinationDir="$(runtime.default)\mui\$(langid)"
        Source="$_RELEASEDIR\$(LANGID)\testZH.dll.mui"/>
</Files>
<RegKeys Language="(zh-CN;zh-TW)">
    <RegKey KeyName="$(hklm.software)\microsoft\testZH\$(LANGID)">
        <RegValue Name="ZHConfig_\$(LANGID)" Value="$(LANGID)" Type="REG_SZ"/>
        <RegValue Name="ZHConfig_\$(LANGID)_Test" Value="$(LANGID)"
            Type="REG_EXPAND_SZ"/>
    </RegKey>
</RegKeys>
...
</OSComponent>
</Components>
```

By specifying the **Language** attribute of **File** or **RegKey** elements, the package generator is notified that the enclosed contents are language related and need to be expanded for all (or just the specified) languages. Inside the element, you can use \$(LANGID) to reference the actual language.

## Note

While only **File** and **RegKey** elements have **Language** attributes, most of the package objects can contain **File** and **RegKey** child elements. For more info about these elements, see [Specifying files and registry entries in a package project file](#).

## Add a resolution-specific content to a package

Similar to including language-specific content, packages can also include resources (files or registry settings) that are specific to a resolution. This is done by creating a "resolution package." The way to specify resolution-specific content is also similar to that of language-specific content; the only difference is that you must use the **Resolution** attribute instead of the **Language** attribute. The following XML example demonstrates how to designate certain files and registry keys for a specific resolution.

```
<Components>
    <OSComponent>
        ...
        <Files Resolution="">
            <File Source="$_RELEASEDIR\$(RESID)\testA.jpg" Name="testA.\$(resid).jpg"/>
        </Files>
        <Files Resolution="(320X480;240x320)">
            <File Source="$_RELEASEDIR\$(RESID)\testB.jpg"
                  DestinationDir="$(runtime.system32)\$(resid)"/>
        </Files>
        <Files Resolution="(480x240)">
            <File Source="$_RELEASEDIR\$(RESID)\testC.jpg"
                  DestinationDir="$(runtime.system32)\$(resid)" Name="testC_\$(resid).jpg"/>
        </Files>
        <RegKeys Resolution="">
            <RegKey KeyName="$(hkLM.software)\microsoft\ResRelatedSettings\$(RESID)">
                <RegValue Name="Config" Value="\$RESID" Type="REG_SZ"/>
            </RegKey>
        </RegKeys>
        ...
    </OSComponent>
</Components>
```

## Add a driver component

Driver injection is supported by using the **Driver** element in the package project XML file. The following XML example demonstrates how to specify the .inf and .sys file for a driver. For more info about the elements and attributes available for drivers, including those for security, see [Specifying components in a package project file](#).

```
<Components>
    <Driver InfSource="$_RELEASEDIR\testDriver.inf">
        <Reference Source="$_RELEASEDIR\testDriver.sys"/>
        <Files>
            <File Source="$_RELEASEDIR\testDriver.sys"/>
        </Files>
    </Driver>
</Components>
```

## Package a driver

The default file location for driver installation is "\$(runtime.drivers)". The staging of the driver object requires access to the Mobile Core hive files. For a package that includes a driver, it is necessary to also set the variable HIVE\_ROOT to the directory with those hives, which should be %WPDKCONTENTROOT%\CoreSystem. Under these circumstances, the command for package generator would be the following:

```
PkgGen SampleDriver.pkg.xml /config:"%WPDKCONTENTROOT%\Tools\bin\i386\pkggen.cfg.xml"
/variables:"HIVE_ROOT=%WPDKCONTENTROOT%\CoreSystem"
```

## Note

If the driver uses the **Include** INF directive to reference other drivers that are part of the Mobile Core subset of the operating system, use the **WIM\_ROOT** variable instead of the **HIVE\_ROOT** variable. The default directory for the staging WIM image is the same as the hives.

For Windows 10 Mobile, you must use both the **HIVE\_ROOT** and **WIM\_ROOT** parameters. If you use only **WIM\_ROOT**, the package might not be complete.

## Add a service component

The **Service** element in the package project XML file, its child elements, and its attributes can be used to define and package a system service. For more detailed info about the attributes and child elements of the **Service** element, see [Specifying components in a package project file](#).

## Add customization settings

Managed Centralized Settings Framework (MCSF) provides a standard way to describe settings that are customizable within packages. A **SettingsGroup** element represents a settings group in the customization answer file.

## Add binary partitions

The **BinaryPartition** element is a special element that represents a binary dump of a partition on the device. A package project can only specify one **BinaryPartition** element. Additionally, a BinaryPartition element can't coexist with any other objects and requires that the **BinaryPartition** attribute of the **Package** element be set to **true**. The following example demonstrates this.

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
    Owner="OEMName"
    OwnerType="OEM"
    ReleaseType="Test"
    Component="ComponentName"
    SubComponent="SubName">
    <Partition>
        <BinaryPartition BinaryPartition="true">
            <Components>
                <BinaryPartition ImageSource="$_RELEASEDIR)\testDPP.bin"/>
            </Components>
        </BinaryPartition>
    </Partition>
</Package>
```

## Project scope macros

Package projects can utilize macros to simplify the XML creation process. Some macros are already globally defined, in which case they can't be changed or modified, but you can also define local macros for use within your own package project XML file. This local macro definition is embedded in the specific package project file through the **Macros** element. The following example demonstrates creating a local macro for use in a package project file.

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
  ...
  >
  <Macros>
    <Macro Id="testName" Value="testValue"/>
  ...
  </Macros>

  <Components>
    ...
  </Components>
</Package>
```

After defining the macro shown here, you could use \$(testName) to reference the value "testValue" in your project. For more info, see "The Macros element and local project macros" in [Primary elements and attributes of a package project file](#).

[Send comments about this topic to Microsoft](#)

# Primary elements and attributes of a package project file

1/25/2019 • 5 minutes to read

The root XML element in the **package** project XML file is the Package element. This element is the base container element for all other package-related elements in a package project XML file. It must occur only once, and it contains all the package information in the project file as attributes and child elements. The information in the **Package** element can be divided into four key areas: attributes, macros, security capabilities, and components.

The following XML example shows the schema definition of the **Package** element.

```
<xs:element name="Package">
  <xs:complexType>
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element minOccurs="0" maxOccurs="1" ref="CustomMetadata" />
      <xs:element minOccurs="0" maxOccurs="1" ref="Macros" />
      <xs:element minOccurs="0" maxOccurs="1" ref="Capabilities" />
      <xs:element minOccurs="0" maxOccurs="1" ref="Components" />
      <xs:element minOccurs="0" maxOccurs="1" ref="Authorization" />
    </xs:sequence>
    <xs:attribute name="Owner" type="xs:string" use="required" />
    <xs:attribute name="Component" type="xs:string" use="required" />
    <xs:attribute name="SubComponent" type="xs:string" use="optional" />
    <xs:attribute name="BinaryPartition" type="xs:boolean" use="optional" />
    <xs:attribute name="OwnerType" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Microsoft" />
          <xs:enumeration value="OEM" />
          <xs:enumeration value="SiliconVendor" />
          <xs:enumeration value="MobileOperator" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="ReleaseType" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Production" />
          <xs:enumeration value="Test" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="Partition" type="xs:string" use="optional" />
    <xs:attribute name="Platform" type="xs:string" use="optional" />
    <xs:attribute name="GroupingKey" type="xs:string" use="optional" />
    <xs:attribute name="Description" type="xs:string" use="optional" />
  </xs:complexType>
</xs:element>
```

## Package attributes

Attributes of the target package(s) are described by using the XML attributes of the **Package** element. This element currently supports the following attributes.

ATTRIBUTE	DEFINITION
<b>Owner</b>	<p>Required. String that specifies the owner of the package. For example, Contoso.</p>
<b>Component</b>	<p>Required. String that specifies the component represented by the package. This value is used as a portion of the package ID.</p>
<b>SubComponent</b>	<p>Optional. String that specifies the subcomponent represented by the package.</p>
<b>ReleaseType</b>	<p>Required. String that specifies the type of release for which the package should be included. Valid values are the following:</p> <ul style="list-style-type: none"> <li>• Production</li> <li>• Test</li> </ul> <div data-bbox="817 848 1436 1080" style="border: 1px solid black; padding: 10px;"> <p><b>Important</b></p> <p>If a package is intended to be included in a retail image, it must be marked as <b>Production</b>. Packages marked as <b>Test</b> will fail retail signing and are not allowed in retail images.</p> </div>
<b>OwnerType</b>	<p>Required. String that specifies the category of owner for the package. Valid values are the following:</p> <ul style="list-style-type: none"> <li>• Microsoft</li> <li>• OEM</li> <li>• SiliconVendor</li> <li>• MobileOperator</li> </ul>
<b>Partition</b>	<p>Optional. String identifier for the target partition for the package. By default, packages are installed to the MainOS partition unless another is explicitly specified.</p> <div data-bbox="817 1612 1436 1866" style="border: 1px solid black; padding: 10px;"> <p><b>Warning</b></p> <ul style="list-style-type: none"> <li>• Packages that are intended to be updated must not target the data partition. The reason for this restriction is that the data partition is formatted when the device reset feature is used.</li> <li>• Packages must only target a single partition.</li> </ul> </div>

ATTRIBUTE	DEFINITION
<b>Platform</b>	<p>Required. String, such as "QC8960", that specifies the target platform for the package if the package is targeting a specific platform. By default, this value is NULL.</p> <div data-bbox="825 291 1428 494" style="border: 1px solid black; padding: 10px;"> <p><b>Note</b></p> <p>Although this attribute is specified as optional in the schema, it is required for all packages where the <b>OwnerType</b> attribute is not set to Microsoft.</p> </div>
<b>GroupingKey</b>	<p>Optional. OEM-defined string that specifies an identifier for grouping packages together. By default, this value is NULL.</p>
<b>BinaryPartition</b>	<p>Optional. Boolean value that indicates whether the package contains a binary partition object. By default, this value is set to False.</p>
<b>Description</b>	<p>Optional. String that can be used to provide additional details about the package. This field can contain any string and is not read by any Microsoft components once contained in the manifest.</p>

The following XML example demonstrates how to specify the package attributes.

```
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
    Owner="OEMName"
    OwnerType="OEM"
    ReleaseType="Test"
    Platform="PlatformName"
    Component="ComponentName"
    SubComponent="SubName">
</Package>
```

#### Note

The package attributes shown in this example support the use of macros.

## The Macros element and local project macros

To simplify the creation of the package project XML, macros can be used in the project file. A macro is essentially a unique identifier in the project file that is later replaced by a defined value. Each macro definition is represented by a **Macro** element within the parent **Macros** element. Each macro is defined by the attributes **Id** and **Value**. The following XML example shows the use of the **Macros** element to define a macro.

```
<Macros>
    <Macro Id="TypeLibId" Value="{F5078F18-C551-11D3-89B9-0000F81FE221}" />
    <Macro Id="ProxyStubClsId"
        Value="{00020424-0000-0000-C000-000000000046}" />
</Macros>
```

The following XML example shows the schema definition of the **Macros** element.

```

<xss:element name="Macros">
  <xss:complexType>
    <xss:sequence>
      <xss:element name="Macro" minOccurs="1" maxOccurs="unbounded">
        <xss:annotation>
          <xss:documentation>
            A Macro element defines a text substitution macro that can be used
            in other elements. Macros are referenced using NMAKE syntax, i.e.
            $(runtime.windows).
          </xss:documentation>
        </xss:annotation>
      </xss:element>
    </xss:sequence>
  </xss:complexType>
  <xss:attribute name="Id" type="MacroIdType" use="required">
    <xss:annotation>
      <xss:documentation>
        Required. The Id for this macro, used in macro references.
        For example, if the Id for this macro is "runtime.windows",
        the macro would be referenced as $(runtime.windows).
      </xss:documentation>
    </xss:annotation>
  </xss:attribute>
  <xss:attribute name="Value" type="MacroValueTypeStringType" use="required">
    <xss:annotation>
      <xss:documentation>
        Required. The value that will be substituted for macro
        references in macro- enabled XML attributes.
      </xss:documentation>
    </xss:annotation>
  </xss:attribute>
</xss:complexType>
</xss:element>
</xss:sequence>
</xss:complexType>
</xss:element>

```

ATTRIBUTE	DEFINITION
<b>Id</b>	<p>Required. String that specifies the name of the macro. This identifier should always start with either letters or a “<i>character and then be followed by letters, digits, the “.” character, or the “.” character.</i> No other characters are allowed in the <b>Id</b> attribute.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <b>Note</b>            The <b>Id</b> for a macro is case sensitive and must be unique for each defined Macro element in the package project XML file.         </div>
<b>Value</b>	<p>Required. String that specifies the expansion of the macro identifier. This value can include most ANSI characters and must match the following regular expression pattern: [a-zA-Z0-9-_!@#%^.,:=+~`{}[]\$]*.</p>

Macros can be used throughout the project file by using the syntax \$(MacroName). Although macros are valid in most elements of the project file, they are not supported in all. Macros cannot be referenced in a nested fashion—for example \$(Macro1\_\$(Macro2)—and they cannot be used in the definition of package attributes. Some other items of note about macros are:

- A macro definition can include the use of a macro. For example, the following is valid.

```
<Macros>
  <Macro Id="Windows" Value="\windows"/>
  <Macro Id="System32" Value="$(Windows)\System32"/>
</Macros>
```

- The order macros are defined does not matter as long as the definitions are not recursive.

There are additionally global macros that can be used in a package project XML file, but they cannot be redefined within the package. Redefining a global macro causes a failure when the package generator tries to build the package.

## The Components element

The **Components** element is the most important part of a package. It defines for the package generator what files and settings are included in the target package(s). Files, registry settings, and other information are grouped into different kinds of objects for packaging. You can include multiple objects into the target package(s) by listing them in this XML section. Currently the following objects are defined:

- OSComponent
- AppResource
- Application
- BCDStore
- BinaryPartition
- ComServer
- Driver
- Service
- SvcHostGroup
- WinRTHost
- FullTrust
- InboxApp
- SettingsGroup

For additional details about these components, refer to [Specifying components in a package project file](#). You can have any number of these types of objects in your package project (with the exception of some restrictions for the **BinaryPartition** object).

[Send comments about this topic to Microsoft](#)

# Specifying components in a package project file

1/25/2019 • 14 minutes to read

This section provides more detailed information about the supported children of the **Components** element in a package project XML file.

## OSComponent element

The **OSComponent** element is a container for **Files** and **RegKeys** elements. **OSComponent** has no attributes and is usually used to include system components such as shared DLLs, data files, and registry settings. For additional information about the **Files** and **RegKeys** elements, see [Specifying files and registry entries in a package project file](#).

### Note

The **OSComponent** element must contain at least one **Files** element or **RegKeys** element.

The following XML example uses the **OSComponents** element to include various system files.

```
<Components>
  <OSComponent>
    <Files>
      <File Source="$_RELEASEDIR)\testdll.dll"/>
      <File Source="$_RELEASEDIR)\testzh.dll"/>
      <File Source="$_RELEASEDIR)\testEA.dll"/>
    </Files>
    <Files Language="*">
      <File DestinationDir="$(runtime.default)\mui\$(langid)"
            Source="$_RELEASEDIR)\$(LANGID)\testdll.dll.mui"/>
    </Files>
    <Files Language="(zh-CN;zh-TW)">
      <File DestinationDir="$(runtime.default)\mui\$(langid)"
            Source="$_RELEASEDIR)\$(LANGID)\testZH.dll.mui"/>
    </Files>
    <Files Language="(zh-CN;zh-TW;jp-jp;kr-kr)">
      <File DestinationDir="$(runtime.default)\mui\$(langid)"
            Source="$_RELEASEDIR)\$(LANGID)\testEA.dll.mui"/>
    </Files>
    <RegKeys Language="(zh-CN;zh-TW)">
      <RegKey KeyName="$(hklm.software)\microsoft\testZH\$(LANGID)">
        <RegValue Name="ZHConfig_$(LANGID)" Value="$(LANGID)" Type="REG_SZ"/>
        <RegValue Name="ZHConfig_$(LANGID)_Test" Value="$(LANGID)"
                  Type="REG_EXPAND_SZ"/>
      </RegKey>
    </RegKeys>
  </OSComponent>
</Components>
```

The files and registry keys can represent language-neutral or language-specific components.

## Driver element

Windows 10 Mobile uses some of the same driver model as Windows 10 for desktop editions (Home, Pro, Enterprise, and Education). You must import an .inf file for your driver into a package by using the **Driver** element. When an .inf file is specified, the packaging infrastructure calls the driver installation code to simulate the driver installation and determine the necessary registry change for the driver.

The following table describes the attributes of the **Driver** element.

ATTRIBUTE	DESCRIPTION
<b>InfSource</b>	Required. Specifies the .inf file for your driver to import into the driver package.

The following table describes the child elements of the **Driver** element.

ELEMENT	DESCRIPTION
<b>Reference</b>	Optional. Specifies any additional files that are required when installing the driver (such as the driver's .sys file). Adding a <b>Reference</b> element doesn't actually add the file to the package, however; to do that, you must still use a <b>Files</b> element.
<b>Files</b>	Optional. Specifies a file to include in the driver package (such as the driver's .sys file). For more info, see <a href="#">Specifying files and registry entries in a package project file</a> .
<b>Security</b>	Optional. Specifies how the driver is accessible applications and services. For more info, see <a href="#">Security element</a> below.

The following example adds a driver to a package.

```
<Components>
  <Driver InfSource=
    "$(_WINPHONEROOT)\Sample.inf">
    <Reference Source="$(_RELEASEDIR)\Sample.sys" />
    <Files>
      <File Source="$(_RELEASEDIR)\Sample.sys"/>
    </Files>
  </Driver>
</Components>
```

The default file location for driver installation is "\$(runtime.drivers)". The staging of the driver object requires access to the Mobile Core hive files. Therefore, it is necessary to set the variable **HIVE\_ROOT** to the directory that contains those hives, which by default is %WPDKCONTENTROOT%\CoreSystem. The following example shows a package generator (PkgGen.exe) command that sets the **HIVE\_ROOT** variable.

```
Pkggen.exe <other arguments> /variables:"HIVE_ROOT=%WPDKCONTENTROOT%\CoreSystem";<other variables> <other arguments>
```

## Note

If the driver uses the **Include** INF directive to reference other drivers that are part of the Mobile Core subset of the operating system, use the **WIM\_ROOT** variable instead of the **HIVE\_ROOT** variable. The default directory for the staging WIM image is the same as the hives.

For Windows 10 Mobile, you must use both the **HIVE\_ROOT** and **WIM\_ROOT** parameters. If you use only **WIM\_ROOT**, the package might not be complete.

For more information about command-line arguments for PkgGen.exe, see [Command-line arguments for package generator](#).

## Security element

Under the default security policy for drivers, drivers are accessible to other TCB components (including other drivers and the kernel) and to services. Some device experiences require a driver that can be accessed by applications as well as services. To enable this scenario, OEMs may need to add a **Security** element under the **Driver** element.

The **Security** element has the following attribute and child elements.

ATTRIBUTE	DESCRIPTION
<b>InfSectionName</b>	Required. This attribute specifies the name of an <b>AddReg</b> directive section in the driver's .inf file. This specification allows for the injection of a Security Descriptor Definition Language (SDDL) string into the .inf file as part of package generation.
ELEMENT	DESCRIPTION
<b>AccessedByCapability</b>	<p>Optional. This element is used to specify a capability ID that is required for non-TCB code to access the driver, as well as the rights granted to that capability. The capability ID is specified through the <b>Id</b> attribute, which accepts a string that is the capability name. The rights are specified through the <b>Rights</b> attribute, which accepts one or more of the following strings:</p> <ul style="list-style-type: none"><li>• \$(DEVICE_READ)</li><li>• \$(DEVICE_WRITE)</li><li>• \$(DEVICE_ALL)</li><li>• \$(DEVICE_EXECUTE)</li></ul>
<b>AccessedByApplication</b>	<p>Optional. This element is used to specify an app that can access the driver, as well as the rights granted to that application. The app is specified through the <b>Name</b> attribute, which accepts a string that is the app's product ID GUID. The rights are specified through the <b>Rights</b> attribute, which accepts one or more of the following strings:</p> <ul style="list-style-type: none"><li>• \$(DEVICE_READ)</li><li>• \$(DEVICE_WRITE)</li><li>• \$(DEVICE_ALL)</li><li>• \$(DEVICE_EXECUTE)</li></ul>

### Note

The product ID GUID must be specified including brackets, for example

```
{263AF644-C573-4e00-BB49-740DD4C69664}.
```

ELEMENT	DESCRIPTION
<b>AccessedByService</b>	<p>Optional. This element is used to specify a service that can access the driver, as well as the rights granted to that service. The service is specified through the <b>Name</b> attribute, which accepts a string that is the name of the service. This value must match the name of the service as it was declared in the <b>Name</b> attribute of the <b>Service</b> element. The rights are specified through the <b>Rights</b> attribute, which accepts one or more of the following strings:</p> <ul style="list-style-type: none"> <li>• \$(DEVICE_READ)</li> <li>• \$(DEVICE_WRITE)</li> <li>• \$(DEVICE_ALL)</li> <li>• \$(DEVICE_EXECUTE)</li> </ul>

## Service element

The **Service** element describes a service in the system and is used for the packaging and configuration of partner services.

The following table describes the attributes of the **Service** element.

ATTRIBUTE	DESCRIPTION
<b>Name</b>	Required. The name of the service. The string is case sensitive.
<b>DisplayName</b>	Optional. The displayable name of the service.
<b>Description</b>	Optional. A description of the service.
<b>Start</b>	<p>Required. An enumeration value that defines when the service will start. Valid values are the following:</p> <ul style="list-style-type: none"> <li>• <b>Demand</b> – The service starts after the OS has finished booting and the shell is available. Microsoft strongly recommends using this value for services in Windows 10 Mobile wherever possible.</li> <li>• <b>Auto</b> – The service starts automatically during system startup.</li> </ul>
<b>Type</b>	<p>Required. An enumeration value that defines the service type. Valid values are the following:</p> <ul style="list-style-type: none"> <li>• <b>Win32OwnProcess</b> – A service that runs in its own process.</li> <li>• <b>Win32ShareProcess</b> – A shared service that runs in a host process with other services.</li> </ul>

ATTRIBUTE	DESCRIPTION
<b>DependOnService</b>	Optional. A service on which the service being defined depends.
<b>ErrorControl</b>	<p>Optional. An enumeration value that defines the severity of the error and the action to take if the service fails to start. Valid values are the following:</p> <ul style="list-style-type: none"> <li>• <b>Ignore</b> –The startup program should ignore the error and continue the startup operation.</li> <li>• <b>Normal</b> – The startup program should log the error in the event log but continue the startup operation.</li> <li>• <b>Severe</b> – The startup program should log the error in the event log. If the last-known-good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known-good configuration.</li> </ul>
<b>SvcHostGroupName</b>	Required for shared services. A unique OEM-defined string that identifies a particular service group. All services shared in the same process must assign the <b>SvcHostGroupName</b> attribute to the same value.

The following table describes the child elements of the **Service** element.

ELEMENT	DESCRIPTION

ELEMENT	DESCRIPTION
<b>Executable</b>	<p>Required for services that run in their own process. This element defines the service executable through the following four attributes.</p> <ul style="list-style-type: none"> <li>• <b>Name</b> - Optional. The name for the executable on the device. If a name is not specified, the file name in the source path is used.</li> <li>• <b>Source</b> – Required. The source path of the service executable to be included in the package. This path can be an absolute or relative path. It is also valid to use macro references in the path.</li> </ul> <div style="border: 1px solid black; padding: 5px;"> <p><b>Note</b> If the executable file does not exist, package generation will fail.</p> </div> <ul style="list-style-type: none"> <li>• <b>DestinationDir</b> – Optional. The destination folder on the device for the service executable. This path must start with a globally defined macro for a directory. If this attribute is not specified, the default location for services is used.</li> <li>• <b>Attributes</b> – Optional. The file system attributes for this file on the device. The value can be a combination of the following, separated by a space character. <ul style="list-style-type: none"> <li>◦ Archive</li> <li>◦ Hidden</li> <li>◦ Offline</li> <li>◦ ReadOnly</li> <li>◦ System</li> <li>◦ Temporary</li> <li>◦ SparseFile</li> <li>◦ NotContentIndexed</li> </ul> <p>When not specified, a predefined default value is used.</p> <div style="border: 1px solid black; padding: 5px;"> <p><b>Note</b> The file system might not support all of these attributes.</p> </div> </li> </ul>

ELEMENT	DESCRIPTION
<b>FailureActions</b>	<p>Optional. This element defines the actions taken on the failure of the service. It has one child element (<b>Action</b>) and two attributes (<b>Command</b> and <b>ResetPeriod</b>).</p> <p>The <b>Action</b> element specifies the action to be performed. At least one <b>Action</b> element is required when <b>FailureActions</b> is included, and each <b>Action</b> element must contain two attributes. The first attribute is <b>Type</b>, which is an enumeration value that specifies the type of action to take. The second attribute is <b>Delay</b>, which is a non-negative integer that specifies the time, in milliseconds, to wait before performing the action. Valid values for <b>Type</b> are the following:</p> <ul style="list-style-type: none"> <li>• <b>None</b> – No action.</li> <li>• <b>RestartService</b> – Restart the service.</li> <li>• <b>RunCommand</b> – Run a command.</li> </ul> <p>The following list describes the two attributes of the <b>FailureActions</b> element.</p> <ul style="list-style-type: none"> <li>• <b>Command</b> – Optional. The command line of the process for the <b>CreateProcess</b> function to execute in response to a failure, when the <b>RunCommand</b> attribute is specified through the <b>Action</b> element. This process runs under the same account as the services.</li> </ul> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p><b>Note</b></p> <p>If the value is NULL, the command is unchanged. If the value is an empty string (""), the command is deleted and no command is run when the service fails.</p> </div> <ul style="list-style-type: none"> <li>• <b>ResetPeriod</b> – Required. The time, in seconds, after which to reset the failure count to zero if there are no failures. Specify INFINITE to indicate that the count should never be reset.</li> </ul>

ELEMENT	DESCRIPTION
<b>Files</b>	<p>Optional. This element, through its child element <b>File</b>, defines supporting files to include with the service. The <b>File</b> element has the following four attributes:</p> <ul style="list-style-type: none"> <li>• <b>Name</b> - Optional. The name for the file on the device. If a name is not specified, the file name in the source path is used.</li> <li>• <b>Source</b> – Required. The source path of the file to be included in the package. This path can be an absolute or relative path. It is also valid to use macro references in the path.</li> </ul> <div style="border: 1px solid black; padding: 10px;"> <p><b>Note</b> If the file does not exist, package generation will fail.</p> </div> <ul style="list-style-type: none"> <li>• <b>DestinationDir</b> – Optional. The destination folder on the device for the file. This path must start with a globally defined macro for a directory. If this attribute is not specified, the default location for files is used.</li> <li>• <b>Attributes</b> – Optional. The file system attributes for this file on the device. The value can be a combination of the following, separated by a space character. <ul style="list-style-type: none"> <li>◦ Archive</li> <li>◦ Hidden</li> <li>◦ Offline</li> <li>◦ ReadOnly</li> <li>◦ System</li> <li>◦ Temporary</li> <li>◦ SparseFile</li> <li>◦ NotContentIndexed</li> </ul> <p>When not specified, a predefined default value is used.</p> <div style="border: 1px solid black; padding: 10px;"> <p><b>Note</b> The file system might not support all of these attributes.</p> </div> </li> </ul>
<b>RegKeys</b>	<p>Optional. Specifies registry entries for the service through the child element <b>RegKey</b>. For additional information about the <b>RegKey</b> element, see <a href="#">Specifying files and registry entries in a package project file</a>.</p>

ELEMENT	DESCRIPTION
<b>RequiredCapabilities</b>	<p>Optional. Specifies platform capabilities required by the service. When used, at least one child <b>RequiredCapability</b> element must be provided. The <b>RequiredCapability</b> element has one attribute, <b>CapId</b>, which specifies a valid platform capability.</p>
<b>ServiceDll</b>	<p>Required for shared services that run in a host executable. Specifies information about the DLL that implements the service. The <b>ServiceDll</b> element has the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>Source</b> – Required. The full path of the service DLL on the development computer.</li> <li>• <b>HostExe</b> – Required. The full path of the service host executable for the service on the image. For example, "\$(runtime.system32)\OEM_SampleHost.exe".</li> </ul>

## BinaryPartition element

The **BinaryPartition** element is used to create binary partition packages. If a package contains a **BinaryPartition** object, it can contain no other objects, including other **BinaryPartition** objects.

There is also a special package-level Boolean attribute called **BinaryPartition**. When this attribute is set, one and only one **BinaryPartition** element must be added.

The **BinaryPartition** element has only one attribute, **ImageSource**, which points to a file that contains a binary dump of the target partition. You must specify the appropriate value for the package-level **Partition** attribute.

## ComServer element

The **ComServer** element describes a DLL and all of the COM classes and related items derived from the DLL. It is designed to simplify how CLSID-related registry settings are specified. The **ComServer** element is derived from **OSComponent** and includes three additional child elements:

- **Dll**: This element specifies the DLL that exports all the COM classes in the **ComServer** object. The path of the DLL will also be used for the path stored under the **InprocServer32** registry key.
- **Classes**: This element specifies the classes exported from the **Dll** element. It contains multiple **Class** elements, each of which can have the following attributes.

ATTRIBUTE	DESCRIPTION
<b>ID</b>	Required. A string that specifies the class ID. For example, "{a3079dc1-e685-4e37-af40-057ed6d0e252}"
<b>TypeLib</b>	Optional. A string that specifies the <b>TypeLib</b> class ID.
<b>AppId</b>	Optional. String.

ATTRIBUTE	DESCRIPTION
<b>ProgId</b>	Optional. String.
<b>Description</b>	Optional. String.
<b>VersionIndependentProgId</b>	Optional. String.
<b>Version</b>	Optional. String.
<b>DefaultIcon</b>	Optional. String.
<b>ThreadingModel</b>	<p>Optional. A string that specifies the threading model. Valid values are:</p> <ul style="list-style-type: none"> <li>• Apartment</li> <li>• Free</li> <li>• Both</li> <li>• Neutral</li> </ul>

In addition to these attributes, you can also specify zero or more **RegKey** child elements under a **Class** element. When registry information is included here, the built-in macro \$(hkcr.clsid) (which is mapped to the string "HKCR\Classes\\$\{CLSID\}\<currentClsId\*>") can be used to add more settings for this class without mentioning the full key name. The following example demonstrates the definition of a **Class** object.

```
``` syntax
<Class
  Id="{2933BF90-7B36-11D2-B20E-00C04F983E60}"
  Version="1.0"
  TypeLib="{D63E0CE2-A0A2-11D0-9C02-00C04FC99C8E}"
  ThreadingModel="Both"
  ProgId="Microsoft.XMLDOM.1.0"
  VersionIndependentProgId="Microsoft.XMLDOM"
  Description="XML DOM Document">

  <RegKey KeyName="$(hkcr.clsid)\SideBySide">
    <RegValue Name="RefCount" Type="REG_DWORD" Value="00000001" />
    <RegValue Name="Version30RefCount" Type="REG_DWORD" Value="00000001" />
  </RegKey>
  <RegKey KeyName="$(hkcr.clsid)\VersionList">
    <RegValue Name="3.0" Type="REG_EXPAND_SZ"
      Value="%SystemRoot%\System32\msxml3.dll" />
  </RegKey>
</Class>
```

```

- **Interfaces:** This element contains a list of **Interface** elements that describe the interfaces implemented by the **ComServer** object. Similarly to **Class** elements, the following attributes and zero or more **RegKey** child elements can be specified.

| ATTRIBUTE               | DEFINITION  |
|-------------------------|---|
| <b>ID</b>               | Required. A string that specifies the class ID. For example, "{a3079dc1-e685-4e37-af40-057ed6d0e252}" |
| <b>TypeLib</b>          | Optional. String that specifies the <b>TypeLib</b> class ID.  |
| <b>Name</b>             | Optional. String.   |
| <b>ProxyStubClSID</b>   | Optional. String.   |
| <b>ProxyStubClSID32</b> | Optional. String.   |

For the **Interface** element, there is a built-in macro \$(hkcr.iid) that maps to "HKCR\\Classes\\Interface\\<interfaceId\*>". The following example demonstrates the definition of an **Interface** object.

```
``` syntax
<Interface
  Id="{D4D4A0FC-3B73-11D1-B2B4-00C04FB92596}"
  TypeLib="$(TypeLibId)"
  Name="IXMLAttribute"
  ProxyStubClSID="{00020424-0000-0000-C000-000000000046}"
  ProxyStubClSID32="$(ProxyStubClSID)">
  <RegKey KeyName="$(hkcr.iid)\TypeLib">
    <RegValue Name="Version" Type="REG_SZ" Value="3.0" />
  </RegKey>
</Interface>
```

```

The default device directory for this object is \$(runtime.system32).

## SettingsGroup element

A **SettingsGroup** element represents a settings group in the customization answer file. Managed Centralized Settings Framework (MCSF) provides a standard way to describe settings that are customizable within packages.

## Custom OEM package metadata

OEMs have the ability to insert custom XML metadata into packages through the package project XML file (.pkg.xml). When packages are generated, this XML data is written to a separate file (.meta.xml) inside of the package. When the package is added to the image, the custom metadata will be available on the device.

### Adding custom metadata to package files

The **CustomMetadata** element is used to add custom metadata under the Package node. The XML within CustomMetadata is a series of key-value pairs using the **Field** element as shown below.

```
<Package>
  <CustomMetadata>
    <Field Name="Key">Value</Field>
  </CustomMetadata>
  <Macros>
  </Macros>
  <Capabilities>
  </Capabilities>
  <Components>
  </Components>
  <Authorization>
  </Authorization>
</Package>
```

PkgGen writes out the content of CustomMetadata tag to the .meta.xml file in the package. The layout of this file is nearly identical to the input.

```
<?xml version="1.0" encoding="utf-8"?>
<CustomMetadata xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00">
  <Field Name="Key">Value</Field>
</Field>
</CustomMetadata>
```

The output metadata file is included in the package, and is installed on the phone in \Windows\Packages\CustomMetadataFiles, as *PackageName.meta.xml*. This folder is a sibling of the DsmFiles folder that is used for packages.

The packaging and device update tools treat the .meta.xml file like any other file on the phone.

### Walkthrough – adding custom metadata

Use the following process to add custom data to an existing package.

#### To add custom data to an existing package

1. Locate the desired package project XML file and open it in a text editor such as Notepad.
2. Add the desired metadata tags right after the <Package> element. For example, to track a targeted version, model, and development team, add the following metadata value pairs.

```
<CustomMetadata>
  <Field Name="TargetVersion">8.1</Field>
  <Field Name="PrimaryPhoneModel">DCD6000</Field>
  <Field Name="DevelopmentTeam">Alpha Team</Field>
</CustomMetadata>
```

3. Use PkgGen to generate a package. For more info, see [Creating packages](#).

### Walkthrough – viewing custom metadata

Use the following process to confirm that the metadata file is present in the generated package.

#### To confirm that the metadata file is in the generated package

1. On a Windows PC, locate the generated package and add a ".cab" file extension to the package name.
2. Double-click the renamed file to view files that are stored in the .cab file.
3. The package files are renamed in the .cab file. To locate the filename of the metadata file that will be used on the device, click the `man.dsm.xml` file, extract it, and open it. One of the file entries will show the .meta.xml package name and the .cab file name that is used. In this example, `4_Contoso.xml` is the name of

the metadata XML file.

```
...<FileEntry>
<FileType>Regular</FileType>
<DevicePath>\Windows\Packages\CustomMetadata\Contoso.TestApp.meta.xml</DevicePath>
<CabPath>4_Contoso.xml</CabPath>
<Attributes>Normal</Attributes>
</FileEntry>
```

4. Extract the `4_Contoso.xml` and open it. Confirm that it contains the expected metadata.

```
<?xml version="1.0" encoding="utf-8"?>
<CustomMetadata xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00">
<Field Name="TargetVersion">8.1</Field>
<Field Name="PrimaryPhoneModel">DCD6000</Field>
<Field Name="DevelopmentTeam">Alpha Team</Field>
</CustomMetadata>
```

## Related topics

[Creating packages](#)

[Send comments about this topic to Microsoft](#)

# Specifying files and registry entries in a package project file

1/25/2019 • 5 minutes to read

Because files and registry entries are a key component of each object in a package, it is important to understand how files and registry entries are specified in package project files.

## Files

To include a file, use the **File** element, which is a child of the **Files** element. The following XML example shows the schema definition of the **File** element.

```
<xss:complexType name="fileType">
  <xss:attribute name="Name" type="xs:string" use="optional"/>
  <xss:attribute name="Source" type="xs:string" use="required"/>
  <xss:attribute name="DestinationDir" type="DevicePathType" use="optional"/>
  <xss:attribute name="Attributes" type="attributesType" use="optional" />
  <xss:attribute name="EmbeddedSigningCategory" type="xs:string" use="optional" />
</xss:complexType>
```

This element has the following attributes.

ATTRIBUTE	DEFINITION
<b>Source</b>	<p>Required. Specifies the source path of the file to be included in to the package. This path can be an absolute or relative path. It is also valid to use macro references within the path.</p> <div style="border: 1px solid black; padding: 5px;"><p><b>Note</b></p><p>The file must exist, or package generation will fail.</p></div>
<b>DestinationDir</b>	<p>Optional. Specifies the destination folder on the device for the specified file. This path must start with a globally defined macro for a folder. If this attribute is not specified, the default location depends on the owned object.</p>
<b>Name</b>	<p>Optional. Specifies the name for this file on the device. If a name is not specified, the file name in the source path is used.</p>

ATTRIBUTE	DEFINITION
<p><b>Attributes</b></p>	<p>Optional. Specifies the file system attributes for this file on the device. This value can be a combination of the following separated by a space character</p> <ul style="list-style-type: none"> <li>• Archive</li> <li>• Hidden</li> <li>• Offline</li> <li>• ReadOnly</li> <li>• System</li> <li>• Temporary</li> <li>• SparseFile</li> <li>• NotContentIndexed</li> </ul> <p>When not specified, a predefined default value is used.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p><b>Note</b></p> <p>Although the above attributes can be specified, the file system might not support all of them.</p> </div>
<p><b>EmbeddedSigningCategory</b></p>	<p>Optional. Specifies the type of code signing that should be used for the file during package generation.</p> <p>The string that is provided for EmbeddedSigningCategory is passed unchanged to the sign.cmd tool that is used in package generation. For info on the available signing options for sign.cmd, see <a href="#">Sign binaries and packages</a>.</p> <p>For example, to specify that HAL binaries should be signed using the -oem and -hal option, add the EmbeddedSigningCategory attribute as shown.</p> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <pre>&lt;Files&gt;   &lt;File Source="MyHalFileLocation\HalExt1.dll"     EmbeddedSigningCategory="-oem -hal" /&gt;   &lt;File Source="MyHalFileLocation\HalExt2.dll"     EmbeddedSigningCategory="-oem -hal" /&gt;   &lt;File Source="MyHalFileLocation\HalExt3.dll"     EmbeddedSigningCategory="-oem -hal" /&gt; &lt;/Files&gt;</pre> </div> <p>For general information about code signing, see <a href="#">Code signing</a>.</p>

## Registry keys and values

In Windows 10 Mobile, registry keys are instead specified in the package project XML file by using the **RegKey** element.

The **RegKey** element has only one required attribute, **KeyName**, which is the full path of the registry key. **KeyName** must start with a globally defined macro, but it does not require the trailing slash at the end of the key name.

Under the **RegKey** element can be zero or more **RegValue** elements. Each of these elements represents a value

under the registry key. The **RegValue** element has the following attributes.

### Note

The MUI\_SZ registry key type has been replaced with the REG\_SZ or REG\_EXPAND\_SZ type. Developers should use either of these and change the code to use the RegLoadMUIString function to do the same. The format of the value is "@[path]\dllname,-strID" and an example would be "@%SystemRoot%\System32\msxml3r.dll,-2". This change occurred because the Windows kernel does not support a MUI\_SZ type.

ATTRIBUTE	DEFINITION									
<b>Name</b>	Required. String that specifies the name of the value. The symbol "@" can be used to specify the default value. Additionally, macros can be used in the attribute value.									
<b>Value</b>	Required. String that represents the value of the key. This attribute is dependent on the value type.									
<b>Type</b>	Required. The type of the value. The following list shows the possible type values and the required format for the paired <b>Value</b> attribute. <table border="1"><thead><tr><th>TYPE VALUE</th><th>DEFINITION</th><th>VALUE ATTRIBUTE</th></tr></thead><tbody><tr><td>REG_SZ</td><td>A plain text string.<div style="border: 1px solid black; padding: 5px;"><p><b>Note</b> A NULL terminated character is appended to the end of this string automatically. If for some reason you don't want the null terminator, you must use REG_HEX with a raw hex dump.</p></div></td><td>"test value"</td></tr><tr><td>REG_MULTI_SZ</td><td>Multiple text strings separated by '.'. See note.</td><td>"value1;value2;value3"</td></tr></tbody></table>	TYPE VALUE	DEFINITION	VALUE ATTRIBUTE	REG_SZ	A plain text string. <div style="border: 1px solid black; padding: 5px;"><p><b>Note</b> A NULL terminated character is appended to the end of this string automatically. If for some reason you don't want the null terminator, you must use REG_HEX with a raw hex dump.</p></div>	"test value"	REG_MULTI_SZ	Multiple text strings separated by '.'. See note.	"value1;value2;value3"
TYPE VALUE	DEFINITION	VALUE ATTRIBUTE								
REG_SZ	A plain text string. <div style="border: 1px solid black; padding: 5px;"><p><b>Note</b> A NULL terminated character is appended to the end of this string automatically. If for some reason you don't want the null terminator, you must use REG_HEX with a raw hex dump.</p></div>	"test value"								
REG_MULTI_SZ	Multiple text strings separated by '.'. See note.	"value1;value2;value3"								

ATTRIBUTE	DEFINITION	DEFINITION	VALUE ATTRIBUTE
	REG_DWORD	32-bit number represented in hex format, with no "0x" prefix.	"1234ABCD"
	REG_QWORD	64-bit number in hex format, with no "0x" prefix.	"1234ABCD1234ABCD"
	REG_BINARY	Byte array in hex format, separated by ',', with no "0x" prefix.	"1A,2B,3D"
	REG_EXPAND_SZ	Same as REG_SZ.	
	REG_HEX	A byte array (same as REG_BINARY) with a type prefix in the form of "hex(x)." where x is the hex number for the type.	"hex(7):1A,2B,3D"

**Note**  
 Given that the ',' character is the delimiter for REG\_MULTI\_SZ, the ';' character cannot be used inside the string because it would be parsed as the separator of two strings. A workaround for this is to use the REG\_HEX type instead and specify the string with the hex dump, e.g.: "hex(7):<exact hex dump of the string>".

## File and registry key groups, language/resolution expansion, and CPU filtering

For most objects, **File** and **RegKey** elements cannot be directly specified. Instead, they must be grouped under the **Files** and **RegKeys** elements. A **Files** or **RegKeys** element must have at least one **File** or **RegKey** element, respectively.

Both **Files** and **RegKeys** have some special attributes used for language/resolution expansion and CPU filtering.

ATTRIBUTE	DEFINITION
<b>CpuFilter</b>	<p>Optional. Restricts the elements (<b>File</b> or <b>RegKey</b>) to a specific CPU architecture. Currently this attribute supports the following two values:</p> <ul style="list-style-type: none"> <li>● ARM</li> <li>● X86</li> </ul> <p>The attribute is case sensitive and values must be specified in uppercase letters.</p> <div data-bbox="826 485 1429 642" style="border: 1px solid black; padding: 10px;"> <p><b>Note</b></p> <p>This attribute can be used along with the <b>Language</b> or <b>Resolution</b> attribute.</p> </div>
<b>Language</b>	<p>Optional. Indicates to the package generator that the contents under this group are language specific and must be distributed to the corresponding language package. This attribute can be specified with the following values:</p> <ul style="list-style-type: none"> <li>● "": The "" character means that the elements under this group are language specific and are supported for every language that is being built. For each language built, the contents are inserted into the correspondent language packages. To target different files for each language, you can use the built-in global macro <code>\$(LANGID)</code>, which is expanded to the language identifier of the target language package.</li> <li>● "(en-US;zh-CN)": This syntax is used to indicate to the package generator that the contents are language specific, but are only available for the given languages. The contents will then be inserted only into the specified language packages.</li> <li>● "!(en-US;zh-CN)": A '!' in front of the language list explicitly excludes the listed languages from receiving the contents. The contents will not be inserted to those listed languages.</li> </ul> <div data-bbox="826 1574 1429 1720" style="border: 1px solid black; padding: 10px;"> <p><b>Note</b></p> <p>This attribute can be used along with the <b>CpuFilter</b>, but it cannot be used at the same time as <b>Resolution</b>.</p> </div>

ATTRIBUTE	DEFINITION
<b>Resolution</b>	<p>Optional. This attribute is similar in usage to the <b>Language</b> attribute, with the exception that it requires the use of a resolution identifier instead of a language identifier. A resolution identifier is in the form of "&lt;width&gt;x&lt;height&gt;", for example "320x240". A built-in macro \$(RESID) is available for the use of all child elements.</p> <div data-bbox="826 422 1429 579" style="border: 1px solid black; padding: 10px;"><p><b>Note</b> This attribute can be used along with the <b>CpuFilter</b>, but it cannot be used at the same time as <b>Language</b>.</p></div>

Send comments about this topic to Microsoft

# Command-line arguments for package generator

1/25/2019 • 2 minutes to read

The package generator tool (PkgGen.exe) is used to create a package from a package project XML file. The tool is designed to process one project XML file at a time and by default is located at %WPKCONTENTROOT%\Tools\bin\i386. The following section outlines the command-line arguments.

## Warning

Package generation requires catalog files to be signed. To perform this task, a certificate must be specified for use with pkggen.exe. If a certificate is not provided, usage of pkggen.exe will return the following error message:

`Failed to sign package integrity catalog file`. To specify a signing certificate for pkggen.exe, follow the steps in [Set up the signing environment](#) and [Sign binaries and packages](#).

## Usage

**PkgGen** [Project File] [Options] **/config:**[Configuration File]

## Note

When using the package generator tool, make sure to use quote marks for input paths that contain spaces. For example, if you defined a %WPKCONTENTROOT% environment variable that is set to C:\Program Files (x86)\Windows Kits\10, then the invocation of package generator should be

`"%WPKCONTENTROOT%\tools\bin\i386\pkggen.exe"` with a leading and ending quote mark. This guidance should also be followed for the path to the project file and the configuration file. For any parameter values, such as variables, that start with the `/<param_name>:` syntax make sure to quote the entire string after the : symbol if there are any spaces. An example would be

`/variables:"WPKCONTENTROOT=C:\Program Files (x86)\Windows Kits\10;MyVar=TestValue"`.

NAME	DESCRIPTION	USAGE
[Project File]	Path to the project XML file.	Required
<b>/output:</b> [Output Directory]	Specifies the output directory for the processed package. By default this value is ".", meaning the current directory.	Optional
<b>/version:</b> [Version String]	Specifies the version of the package using the format "<major>.<minor>.<hotfix>.<build>". By default this value is "0.0.0.0". <b>Note:</b> The version field must be incremented whenever a package is changed.	Optional
<b>/build:</b> [Build Type]	Specifies the build type of the operating system image the package is being created for (FRE or CHK). By default FRE is used.	Optional

NAME	DESCRIPTION	USAGE
<b>/cpu:</b> [CPU Type]	Specifies the platform CPU type for the package (ARM or x86). By default, ARM is used.	Optional
<b>/language:</b> [Language ID List]	Specifies a list of ";" separated language identifiers for which language packages should be generated. By default this list is empty.	Optional
<b>/resolution:</b> [Resolution ID list]	Specifies a list of ";" separated resolution identifiers for which resolution packages should be generated. By default this list is empty.	Optional
<b>/config:</b> [Configuration File]	Specifies the path to the global macro file PkgGen.cfg.xml that is included with the Windows Driver Kit (WDK). By default, this file is located at %ProgramFiles(x86)%\Windows Kits\10\Tools\bin\i386 (or the corresponding path under %ProgramFiles% on computers running a 32-bit version of Windows).	Required
<b>/variables:</b> [Variable Definition List]	Specifies additional variables for use during package project file processing. The format is "[variable name]=[value];[variable 2 name]=[value];".	Optional
<b>{+ -}diagnostic</b>	Precede with either "+" or "-" to indicate true or false. This option enables debugging messages, which are disabled by default. Syntax for enabling debugging messages would be <code>+diagnostic</code> .	Optional
<b>{+ -}compress</b>	Precede with either "+" or "-" to indicate true or false. The default is false. This value allows for the compression of the package. When compress is true, the package generator tool run time increases, but the package is optimized for storage size. This setting has no impact on imaging.	Optional

Send comments about this topic to Microsoft

# Merging packages before imaging

1/25/2019 • 5 minutes to read

Before you create an image, we recommend that you first merge your packages by using the FeatureMerger.exe tool. This tool takes a feature manifest file as an input and merges your referenced packages into a small, well-defined set of packages that adhere to a predictable and consistent naming system.

This topic provides general guidance about the package merging process.

## Important

For retail images, you must generate merged packages that conform to the *Windows Standard Package Configuration* (WSPC). WSPC is a set of rules that define the package naming requirements for retail devices. For more info, see [Windows Standard Packaging Configuration \(WSPC\) requirements for retail images](#).

## Understanding the package merging process

To create merged packages, create a feature manifest file that lists the packages you want to merge and pass this feature manifest file to the FeatureMerger.exe tool. FeatureMerger.exe merges your packages based on the package groups under which the packages are listed in the feature manifest file and additional package metadata. Based on this information, one or more packages that represent the minimal set of packages based on the metadata are generated. The new merged packages are placed in the output directory specified when the tool was invoked. For instructions about using FeatureMerger.exe, see [Merging packages using FeatureMerger](#).

The following table summarizes how packages are merged based on the feature manifest package group they are listed under.

PACKAGE GROUP ELEMENT IN THE FEATURE MANIFEST	MERGING BEHAVIOR
<b>BasePackages</b>	All packages listed under the <b>BasePackages</b> element with the same partition, language, and resolution will be merged together into a package with the following naming convention:  <code>&lt;Owner&gt;.Base.&lt;FeatureManifestID&gt;.&lt;Partition&gt;.spkg</code>
<b>SOCPackages</b>	All packages listed under the <b>SOCPackages</b> element with the same <b>SOC</b> attribute value, partition, language, and resolution will be merged together into a package with the following naming convention.  <code>&lt;Owner&gt;.SOC_&lt;SOC name&gt;.&lt;FeatureManifestID&gt;.&lt;Partition&gt;.spkg</code>
<b>OEMDevicePlatformPackages</b>	All packages listed under the <b>OEMDevicePlatformPackages</b> and
<b>DeviceSpecificPackages</b>	<b>DeviceSpecificPackages</b> elements with the same <b>Device</b> attribute value, partition, language, and resolution will be merged together into a package with the following naming convention.  <code>&lt;Owner&gt;.DEVICE_&lt;device name&gt;.&lt;FeatureManifestID&gt;.&lt;Partition&gt;.spkg</code>

PACKAGE GROUP ELEMENT IN THE FEATURE MANIFEST	MERGING BEHAVIOR
<b>Features</b>	All packages listed under the <b>Features</b> element with the same <b>FeatureID</b> , partition, language, and resolution will be merged together into a package with the following naming convention.  <code>&lt;Owner&gt;.OEM_&lt;feature ID&gt;.&lt;FeatureManifestID&gt;.&lt;Partition&gt;.spkg</code>
<b>ReleasePackages</b>	All packages listed under the <b>ReleasePackages</b> element with the same <b>ReleaseType</b> attribute value, partition, language, and resolution will be merged together into a package with the following naming convention.  <code>&lt;Owner&gt;.RELEASE_&lt;TEST   PRODUCTION&gt;.&lt;FeatureManifestID&gt;.&lt;Partition&gt;.spkg</code>
<b>SVPackages</b>	All packages listed under the <b>SVPackages</b> element with the same <b>SV</b> attribute value, partition, language, and resolution will be merged together into a package with the following naming convention.  <code>&lt;Owner&gt;.SV_&lt;SV name&gt;.&lt;FeatureManifestID&gt;.&lt;Partition&gt;.spkg</code>
<b>PrereleasePackages</b>	All packages listed under the <b>PrereleasePackages</b> element with the same <b>Type</b> attribute value, partition, language, and resolution will be merged together into a package with the following naming convention.  <code>&lt;Owner&gt;.PRERELEASE_&lt;REPLACEMENT   PROTECTED&gt;.&lt;FeatureManifestID&gt;.&lt;Partition&gt;.spkg</code>

In the preceding table, the partition, language, and resolution metadata used for package merging are derived from the following attributes in the package project XML file used to generate each package:

- **Partition** attribute of the **Package** element.
- **Language** attribute of the **Files** element.
- **Resolution** attribute of the **Files** element.

#### Note

Language and resolution-specific merged packages (packages with the language or resolution suffix in the package file name) do not comply with WSPC. For retail images, these packages should be merged into a WSPC-compliant base package by referencing the packages under the **BasePackages** element in the feature manifest.

For more info about these attributes, see [Primary elements and attributes of a package project file](#) and [Specifying files and registry entries in a package project file](#).

## Naming convention for merged packages

The merged packages generated by FeatureMerger.exe have the following basic naming convention:

`<Owner>.<MergedPackageType>.<FeatureManifestID>.<Partition>.spkg`

The `<MergedPackageType>` substring is derived from the grouping element in the feature manifest, as described

in the previous section. The following table describes where the other substrings are derived from.

SUBSTRING	ORIGIN OF THE SUBSTRING
<i>Owner</i>	<p>The <i>Owner</i> substring is derived from one of the following locations:</p> <ul style="list-style-type: none"><li>• If you pass a single feature manifest XML file to FeatureMerger.exe, this is derived from the <b>/OwnerName</b> command line parameter.</li><li>• If you pass an FMFileList XML file to FeatureMerger.exe, this is derived from the <b>OwnerName</b> attribute of the <b>FM</b> element in the FMFileList XML file.</li></ul>
<i>FeatureManifestID</i>	<p>The <i>FeatureManifestID</i> substring is derived from one of the following locations:</p> <ul style="list-style-type: none"><li>• If you pass a single feature manifest XML file to FeatureMerger.exe, this is derived from the <b>/FMID</b> command line parameter.</li><li>• If you pass an FMFileList XML file to FeatureMerger.exe, this is derived from the <b>ID</b> attribute of the <b>FM</b> element in the FMFileList XML file.</li></ul> <div style="border: 1px solid black; padding: 5px;"><p><b>Important</b> When building merged packages for retail images, OEMs must specify either <b>Phone</b> or <b>Variant</b> for the <i>FeatureManifestID</i> substring. For more info, see <a href="#">Windows Standard Packaging Configuration (WSPC) requirements for retail images</a>.</p></div>
<i>Partition</i>	<p>The <i>Partition</i> substring is derived from the <b>Partition</b> attribute of the <b>Package</b> element in the package project XML files that were used to build the packages that are being merged. For more information, see <a href="#">Primary elements and attributes of a package project file</a>.</p>

## General guidelines for package merging

You should structure your feature manifest files to control how groups of packages are merged. The following list is a set of general guidelines for package merging:

- Plan for how you want your packages to be merged. For example, you may only want certain sets of packages to be available on phones that are targeted for specific mobile operators or markets.
- Either create new feature manifest files, or leverage existing feature manifest files, to control how packages are merged.
- Use the **SOC****Packages**, **OEM****Device****Platform****Packages** and **Device****Specific****Packages** elements to manage hardware-specific packages. For example, all packages associated with a specific SoC element in a feature manifest should have the same **SOC** attribute. FeatureMerger.exe uses each of these categories as a criteria for merging packages. The packages will be merged with the other packages when any combination of the elements matches. If some packages match on some of the attributes and not others, then only the

packages that match on all elements will be contained in a merged package.

- When generating merged packages for retail images, make sure you generate packages that conform to the WSPC. For more info, see [Windows Standard Packaging Configuration \(WSPC\) requirements for retail images](#).
- Use merged packages for all image generation by using the feature manifests that are generated by FeatureMerger.exe. For more info, see [Merging packages using FeatureMerger](#).
- Verify that all packages have the proper metadata to ensure that merging is performed as expected.

For more info about feature manifests, see [Feature manifest file contents](#). For more info about the OEMInput file elements, see [OEMInput file contents](#).

## Related topics

[Merging packages using FeatureMerger](#)

[Windows Standard Packaging Configuration \(WSPC\) requirements for retail images](#)

[Send comments about this topic to Microsoft](#)

# Merging packages using FeatureMerger

1/18/2019 • 9 minutes to read

The package feature merger tool (FeatureMerger.exe) combines packages into a small, well-defined set of merged packages that adhere to a predictable and consistent naming system. For retail images, OEMs must use FeatureMerger.exe to merge their packages into a specific set of packages that conform to the *Windows Standard Package Configuration* (WSPC).

For an overview of package merging, see [Merging packages before imaging](#). For more information about the WSPC requirements for retail images, see [Windows Standard Packaging Configuration \(WSPC\) requirements for retail images](#).

## FeatureMerger.exe overview

FeatureMerger.exe can be used in two ways: by passing a single feature manifest XML file, or by passing an FMFileList XML file that describes multiple feature manifests. The tool generates a set of merged packages and a new feature manifest file that references the merged packages. The new feature manifest file can be referenced in an OEMInput file that is passed to ImgGen.cmd to build an image from the new merged packages.

To generate merged packages that comply with the (WSPC) for retail images, you must organize all your packages into a single feature manifest as described in [Windows Standard Packaging Configuration \(WSPC\) requirements for retail images](#) and pass this feature manifest to FeatureMerger.exe. If you are starting with packages that are described in multiple feature manifest files, you can generate merged packages that comply with WSPC in a multi-step process:

1. Generate an initial set of merged packages and feature manifests by running FeatureMerger.exe with an FMFileList XML file that references the feature manifest files.
2. Manually create a new feature manifest file that references all of the merged packages generated in the first step. Make sure that the merged packages are referenced in the WSPC-compliant sections of the feature manifest file, as described in [Windows Standard Packaging Configuration \(WSPC\) requirements for retail images](#).
3. Generate merged packages by running FeatureMerger.exe with the consolidated feature manifest file.

## Using FeatureMerger.exe with a single feature manifest XML file

FeatureMerger.exe has the following syntax when used with a single feature manifest XML file.

```
FeatureMerger InputFile OutputPackageDir OutputPackageVersion OutputFMDir /FMID:<feature manifest id>
/Languages:<language identifier list>
/Resolutions:<resolution identifier list> /OwnerName:<owner name> /Variables:<variables> [/OwnerType:<owner
type>] [/MergePackageRootReplacement:<replacement directory>
[{-+}Incremental] [{+-}Compress]
```

### Required arguments and switches

The following table lists the required arguments and switches when using FeatureMerger.exe with a single feature manifest XML file.

Argument/Switch	Description	Value
<i>InputFile</i>	The path to the feature manifest XML file.	Text string
<i>OutputPackageDir</i>	The path to the directory where the merged packages will be placed.	Text string
<i>OutputPackageVersion</i>	The version of the merged package that is created using the format " <major>.<minor>.<hotfix>. <build>".	Text string
	<p><b>Note</b></p> <p>The package version must be incremented whenever a package is changed. For more info about package versioning and updates, see <a href="#">Update requirements</a>.</p>	
<i>OutputFMDir</i>	The path to the directory where the generated feature manifests will be placed. Feature manifests are generated with the same file name as the original feature manifest file name. The updated version of the feature manifest file contains the names of the merged packages and the provided version number.	Text string
<b>/FMID:</b> <i>Feature manifest ID</i>	A required short FM ID that describes the feature manifest file that is used in the merged package name to ensure that features from different FM files don't collide. The ID should be eight characters or less and be different from all other FM IDs. When generating merged packages for retail images, this ID must have the value of either <b>Phone</b> or <b>Variant</b> . For more info, see <a href="#">Windows Standard Packaging Configuration (WSPC) requirements for retail images</a> .	Text string
<b>/Languages:</b> <i>Language identifier list</i>	A semicolon "," separated list of supported UI language identifiers. The following is an example of a language identifier list.	Text string
	<pre>en-us;de-de</pre>	

ARGUMENT/SWITCH	DESCRIPTION	VALUE
<b>/Resolutions:</b> <i>Resolution identifier list</i>	A semicolon ";" separated list of supported resolution identifiers. The following is an example of a resolutions identifier list.  480x800;720x1280;768x1280;1080x1920	Text string
<b>/OwnerName:</b> <i>Owner name</i>	The name of the package owner, for example "Contoso".	Text string
<b>/Variables:</b> <i>variables</i>	<p>Additional variables that are used in the project file using the following syntax:&lt;name&gt; = &lt;value&gt;;&lt;name&gt; = &lt;value&gt;;</p> <p>The following variables are pre-defined:</p> <ul style="list-style-type: none"> <li>• <b>_cpotype:</b> Required. Specifies the CPU architecture. For Windows 10 Mobile, this variable must be set to "arm".</li> <li>• <b>buildtype:</b> Required. Specifies the build type. This variable must be set to either "fre" or "chk".</li> <li>• <b>releasetype:</b> Optional. Specifies the image release type. This variable can be set to either "test" or "production". If this variable is not specified, the default value is "test".</li> </ul> <p>For example, to set the CPU, build type, and release type, specify the variables like this:</p> <pre>/Variables:_cpotype=arm;buildtype=fre;releasetype:producti on</pre>	Text string

## Optional and reserved switches

The following table lists the optional switches when using FeatureMerger.exe with a single feature manifest XML file.

SWITCH	DESCRIPTION	VALUE
<b>/OwnerType:</b> <i>Owner type</i>	Specifies the resulting package owner type.	Text string – For OEM packages, this value must be set to "OEM".

SWITCH	DESCRIPTION	VALUE
<b>/MergePackageRootReplacement</b> <i>Replacement directory</i>	Specifies a string to be used in the generated FM file for packages. Replaces the OutputPackageDir in the package paths.	Text string
<b>{+ -}Incremental</b>	Specifies to only re-merge existing merged packages when one of the sources packages has changed. If this value is not set, the default is False. This means that all packages will be rebuilt each time FeatureMerger is called.	Boolean value: Precede this switch with either [+] to set to true or [-] to set to false
<b>{+ -}Compress</b>	This value allows for the compression of merged packages. When <b>compress</b> is true, the merge tool run time increases, but the package is optimized for storage size. This setting has no impact on imaging.	Boolean value: Precede this switch with either [+] to set to true or [-] to set to false

The following table lists the switches that are reserved for use by Microsoft when using FeatureMerger.exe with a single feature manifest XML file.

SWITCH	DESCRIPTION	VALUE
<b>/Critical</b>	Reserved for use by Microsoft.	Text string

## Examples

The following examples demonstrate FeatureMerger.exe syntax when specifying a feature manifest file.

```
FeatureMerger C:\FM\OEMSampleFM.xml D:\FM\MergedPackages 8.0.0.1 D:\FM\MergedFMs /FMID:Phone /Languages:en-us
/Resolutions:480x800;720x1280;768x1280;1080x1920 /OwnerName:Contoso /variables:_cpotype=arm;buildtype=fre
```

## Using FeatureMerger.exe with an FMFileList XML file

FeatureMerger.exe has the following syntax when used with a single feature manifest XML file.

```
FeatureMerger InputFile OutputPackageDir OutputPackageVersion OutputFMDir /InputFMDirPath:<path> /Variables:<variables>
[/MergePackageRootReplacement:<replacement directory>
[{+|-}Incremental] [{+|-}Compress]]
```

## Required arguments and switches

The following table lists the required arguments and switches when using FeatureMerger.exe with an FMFileList XML file.

ARGUMENT/SWITCH	DESCRIPTION	VALUE
-----------------	-------------	-------

ARGUMENT/SWITCH	DESCRIPTION	VALUE
<i>InputFile</i>	The path to the FMFileList XML file. For more info about the contents of the FMFileList XML file, see <a href="#">Contents of the FMFileList XML file</a> in this topic.	Text string
<i>OutputPackageDir</i>	The path to the directory where the merged packages will be placed.	Text string
<i>OutputPackageVersion</i>	The version of the merged package that is created using the format "<major>.<minor>.<hotfix>.<build>".	Text string
<div style="border: 1px solid black; padding: 10px;"> <p><b>Note</b></p> <p>The package version must be incremented whenever a package is changed. For more info about package versioning and updates, see <a href="#">Update requirements</a>.</p> </div>		
<i>OutputFMDir</i>	The path to the directory where the generated feature manifests will be placed. Feature manifests are generated with the same file name as the original feature manifest file name. The updated version of the feature manifest file contains the names of the merged packages and the provided version number.	Text string
<b>/InputFMDir</b> : <i>path</i>	The directory where the source feature manifest files are located.	Text string

Argument/Switch	Description	Value
/Variables: <i>variables</i>	<p>Additional variables used in the project file using the following syntax:&lt;name&gt;=&lt;value&gt;;&lt;name&gt;=&lt;value&gt;;</p> <p>The following variables are pre-defined:</p> <ul style="list-style-type: none"> <li>• <b>_cpotype</b>: Required. Specifies the CPU architecture. For Windows 10 Mobile, this variable must be set to "arm".</li> <li>• <b>buildtype</b>: Required. Specifies the build type. This variable must be set to either "fre" or "chk".</li> <li>• <b>releasetype</b>: Optional. Specifies the image release type. This variable can be set to either "test" or "production". If this variable is not specified, the default value is "test".</li> </ul> <p>For example, to set the CPU, build type, and release type, specify the variables like this:</p> <pre>/Variables:_cpotype=arm;build type=fre;releasetype:producti on</pre>	Text string

## Optional and reserved switches

The following table lists the optional switches when using FeatureMerger.exe with FMFileList XML file.

Switch	Description	Value
/MergePackageRootReplaceme nt: <i>Replacement directory</i>	Specifies a string to be used in the generated feature manifest for packages. Replaces the OutputPackageDir in the package paths.	Text string
{+ -}Incremental	Specifies to only remerge existing merged packages when one of the sources packages has changed. If this value is not set, the default is False. This means that all packages will be rebuilt each time FeatureMerger.exe is called.	Boolean value: Precede this switch with either [+] to set to true or [-] to set to false

SWITCH	DESCRIPTION	VALUE
{+ -}Compress	This value allows for the compression of merged packages. When <b>compress</b> is true, the merge tool run time increases, but the package is optimized for storage size. This setting has no impact on imaging.	Boolean value: Precede this switch with either [+] to set to true or [-] to set to false

The following table lists the switches that are reserved for use by Microsoft when using FeatureMerger.exe with an FMFileList XML file.

SWITCH	DESCRIPTION	VALUE
/Critical	Reserved for use by Microsoft.	Text string

## Examples

The following example demonstrates FeatureMerger.exe syntax when specifying an FMFileList XML file.

```
FeatureMerger.exe C:\FM\OEMSampleFMFileList.xml C:\FM\MergedPackages 8.0.0.1 C:\FM\MergedFMs
/variables:_cputype=arm;builddtype=fre
```

## Contents of the FMFileList XML file

An FMFileList XML file contains the following elements.

ELEMENT	DESCRIPTION
FM	The <b>FM</b> element contains one or more <b>FM</b> elements.

ELEMENT	DESCRIPTION
<b>FM</b>	<p>The <b>FM</b> element is used to point to feature manifest files.</p> <pre data-bbox="853 253 1361 339">&lt;FM Path="C:\MyDir\OEMSampleFM.xml" ReleaseType="Test" OwnerType="OEM" ID="Phone" OwnerName="Contoso" /&gt;</pre> <p>The FM element has the following attributes.</p> <ul style="list-style-type: none"> <li>• <b>Path:</b> The path to the feature manifest files. The variable <b>\$(FMDirectory)</b> can be used to specify the InputFMDir parameter in the path.</li> <li>• <b>ReleaseType:</b> The release type must be set to either "Test" or "Production". For FMs used in final retail images, this should be set to "Production". In addition, all packages defined in a Production FM must have the ReleaseType set to "Production". For more info about working with packages, see <a href="#">Primary elements and attributes of a package project file</a>.</li> <li>• <b>OwnerType:</b> This value should be set to "OEM".</li> <li>• <b>ID:</b> A required short FM ID that describes the feature manifest file that is used in the merged package name to ensure that features from different FM files don't collide. The ID should be eight characters or less and be different from all other FM IDs. When generating merged packages for retail images, this ID must have the value of either <b>Phone</b> or <b>Variant</b>. For more info, see <a href="#">Windows Standard Packaging Configuration (WSPC) requirements for retail images</a>.</li> <li>• <b>OwnerName:</b> The owner name should be set to the OEM name, for example "Contoso".</li> </ul>
<b>SupportedResolutions</b>	<p>The <b>SupportedResolutions</b> element is used to list the resolutions supported in the packages that will be merged.</p> <pre data-bbox="853 1500 1218 1522">&lt;Resolution&gt;480x800&lt;/Resolution&gt;</pre>
<b>SupportedLanguages</b>	<p>The <b>SupportedLanguages</b> element is used to list the languages supported in the packages that will be merged.</p> <pre data-bbox="853 1758 1148 1781">&lt;Language&gt;en-US&lt;/Language&gt;</pre>
<b>SupportedLocales</b>	<p>The <b>SupportedLocales</b> element is used to list the locales supported in the packages that will be merged.</p> <pre data-bbox="853 1994 1101 2016">&lt;Locale&gt;en-US&lt;/Locale&gt;</pre>

For example, the following sample file specifies two feature manifest files, a list of supported resolutions, a

supported language, and a supported locale.

```
<?xml version="1.0" encoding="utf-8"?>
<FMCollectionManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate">

  <FMs>
    <FM Path="D:\FeatureMergeTest\OEMSampleFM1.xml" ReleaseType="Test" OwnerType="OEM" ID="FM1"
    OwnerName="Contoso"/>
    <FM Path="D:\FeatureMergeTest\OEMSampleFM2.xml" ReleaseType="Test" OwnerType="OEM" ID="FM2"
    OwnerName="Contoso"/>
  </FMs>
  <SupportedResolutions>
    <Resolution>480x800</Resolution>
    <Resolution>720x1280</Resolution>
    <Resolution>768x1280</Resolution>
  </SupportedResolutions>
  <SupportedLanguages>
    <Language>en-US</Language>
  </SupportedLanguages>
  <SupportedLocales>
    <Locale>en-US</Locale>
  </SupportedLocales>
</FMCollectionManifest>
```

## Paths that contain spaces

When you are using FeatureMerger.exe, be sure to use quotation marks around paths that contain spaces. For example, if you define a %WPDKCONTENTROOT% environment variable that is set to C:\Program Files (x86)\Windows Kits\10, the invocation of a package merger should be enclosed in quotation marks:

```
"%WPDKCONTENTROOT%\tools\bin\i386\FeatureMerger.exe"
```

This guidance also applies to the parameters that take paths, such as **/InputFMDirpath**.

## Package code signing

Like all package generation, merged package generation uses catalog signing. For catalog signing to work, a certificate must be specified. To specify a signing certificate for package generation, follow the steps in [Set up the signing environment](#) and [Sign binaries and packages](#).

## Related topics

[Merging packages before imaging](#)

[Windows Standard Packaging Configuration \(WSPC\) requirements for retail images](#)

# Windows Standard Packaging Configuration (WSPC) requirements for retail images

1/18/2019 • 19 minutes to read

For mobile retail images, OEM firmware packages (sometimes referred to as BSP submissions) must comply with the Windows Standard Package Configuration (WSPC).

WSPC is designed to achieve a better customer experience:

- Smaller download sizes.
- Reduced imaging and update errors.
- Faster updates: Reduced system overhead needed to deliver and install updates.
- Fewer updates: Device updates can make a "single hop" update from any image baseline.

## Note

Microsoft requires that all OEM retail firmware packages (BSP submissions) conform to the WSPC. While there are no exceptions to this policy, Microsoft has included tools and processes to ease the your transition to WSPC compliance.

## WSPC overview

The purpose of WSPC is to define a firmware package set and update rules for that package set where it is easy to determine differences between any two arbitrary versions of the package set. The ability to easily and definitively determine the changes between any two package set versions allows Microsoft to use more efficient methods to update the device. The end result is a better customer experience as the user will experience only a single update session and associated reboot as opposed to several.

### Previous WSPC implementation

Microsoft's previous implementation of WSPC mandated a static firmware package set using static package names. The static package set and names applied globally to all of an OEM's POPs. These packages were tightly coupled to the processor's (SOC's) the file system partition layout. While this implementation fulfilled the goal described in the summary above, it had two serious drawbacks. First, it put a high burden on you to achieve compliance due to its complex implementation. Second, the implementation's tight coupling to the SOC's file system partition layout does not accommodate new SOC's with different partition configurations.

### New WSPC implementation

The new WSPC implementation based on a simpler generalization of our previous implementation. As long as we don't remove or rename packages between firmware versions, determining the differences between any two arbitrary firmware versions is straight forward. We simply compare the source and target firmware package sets, and update any packages that have changed between the two. This includes the addition of packages from one firmware version to another.

Complying with the new model is simple and requires less overhead. Instead of Microsoft prescribing the exact contents of the firmware package set, the you will do this by declaring one of your firmware package sets as a baseline. This baseline will apply to a specific set of POPs defined by the OEM. In addition, since this model is a generalized form of the previous WSPC implementation, Windows Phone 8.1 WSPC compliance firmware submissions are automatically compliant with the new WSPC implementation.

This implications of this model are:

1. You can define multiple baselines on a per POP basis.
2. Once a package is added to a firmware package set, it cannot be removed.

If necessary, you must include an empty package.

3. You cannot rename a package in a firmware package set.

This is effectively the removal of an existing and the addition of a new package.

4. You must be mindful to ensure the firmware package set does not grow beyond reasonable size.

Feature merge should be used to manage firmware package set growth.

## WSPC requirements

### Windows Phone Ingestion Client

You must upgrade to the version x.x.x of the Windows Phone Ingestion Client (WPIC). There are breaking changes in this version that necessitate the upgrade, including:

1. A **Baseline** parameter has been added to the [Initialize-FirmwareSubmission cmdlet](#). This will be used to identify a firmware submission as a baseline for WSPC compliance validation.
2. The **nonWSPCCompliant** parameter of the [Initialize-FirmwareSubmission cmdlet](#) has been deprecated.
3. All requests for all cmdlets from previous versions of the WPIC will be rejected with the following error:

The Windows Phone Ingestion Client version you are using (Version {0}) is no longer supported.  
Please make sure you are using the current version of the Windows Phone Ingestion Client, Version {1}.

### Feature merge

Feature Merge continues to play a critical role in the new WSPC implementation. Packages must be feature merged to greatest extent possible to smallest resultant set possible. Feature Merging firmware packages will help manage package set growth as well avoid situations where you must work around deprecated packages for WSPC compliance.

More information on using feature merge can be found at the links below:

- [Merging packages before imaging](#)
- [Merging packages using FeatureMerger](#)

## Firmware submissions

Firmware updates are determined by comparing the baseline firmware submission with any subsequent submissions.

### Baseline firmware submission

To prepare a firmware submission for updating to retail devices by using a Request For Update (RFU), it must first be declared as a baseline. By declaring a firmware submission as a baseline, the package set comprising firmware submission will be used as the basis for WSPC compliance checks.

Prior to declaring a firmware submission as a baseline, you should have performed sufficient testing on the firmware so that they are confident that the package set will experience minimal and/or manageable variance in subsequent versions. Also, as discussed above, the package set should be feature merged to the greatest extent possible.

To declare the firmware submission as a baseline, you will run the [Initialize-FirmwareSubmission cmdlet](#) with

the **baseline** parameter as shown below.

```
Initialize-FirmwareSubmission -TypeOfSubmission Image -UpdateHistoryPath [path] -OemInputPath [path] -OutputFilePath [path] -Baseline
```

### Subsequent firmware submissions

After a baseline firmware submission has been declared, subsequent firmware submissions targeting the same POPs will be compared with the previous baseline firmware version for WSPC compliance.

**Important** Special care must be taken when targeting firmware submissions for multiple POPs with different baselines. Achieving WSPC compliance across **all baselines** can be difficult.

The WSPC compliance check will ensure that the package set of the new firmware version contains the same packages as the previous baseline or is a superset of the previous baseline. Consider the following package sets as an example:

#### Baseline firmware submission

Package1.spkg version 1

Package2.spkg version 1

Package3.spkg version 1

Package4.spkg version 1

#### Firmware submission v2

Package1.spkg version 2

Package2.spkg version 2

Package3.spkg version 2

Package4.spkg version 2

#### Firmware submission v3

Package1.spkg version 2

Package2.spkg version 2

Package3.spkg version 2

Package4.spkg version 2

Package5.spkg version 1

#### Firmware submission v4

Package1.spkg version 2

Package2.spkg version 2

Package3.spkg version 2

Package5.spkg version 1

Firmware submission v2 is WSPC-compliant with the baseline because the package set is the same between versions (even though the package versions are different).

Firmware submission v4 is not WSPC-compliant with the baseline because Package4.spkg does not exist (has been removed) in version 4.

Microsoft will process all firmware submissions regardless of their WSPC compliance status as long as you have not submitted it as a baseline.

```
Initialize-FirmwareSubmission -TypeOfSubmission Image -UpdateHistoryPath [path] -OemInputPath [path] -OutputFilePath [path]
```

If the new firmware submission is not WSPC-compliant, the following warning will be returned:

```
The firmware submission is not WSPC compliant with the baseline submission {1}. This is because the packages {0} are missing from the new submission. Since this is not a new baseline, the firmware submission is allowed. However, since this submission not WSPC compliant and is not a baseline, it will not be available for a retail servicing update. Retail servicing updates are only allowed for the following scenarios: between an RTM source submission and an RTM target submission, or between a legacy non-WSPC compliant source submission and an RTM target submission. To make this submission WSPC compliant, please ensure the missing packages are included in the submission.
```

Microsoft will reject baseline firmware submissions that are not WSPC-compliant. The following error will be shown:

```
<path>:\ Initialize-FirmwareSubmission -TypeOfSubmission Image -UpdateHistoryPath [path] -OemInputPath [path] -OutputFilePath [path] -Baseline
```

```
The firmware submission is not WSPC compliant with the baseline submission {1}. This is because the packages {0} are missing from the new submission. To make this submission WSPC compliant, please ensure the missing packages are included in the submission.
```

### \*.SPKG and \*.CAB firmware package names

Starting with Windows 10 Mobile, you'll be able to build firmware using Microsoft's Component Based Servicing Model (CBS). CBS packages use the \*.CAB extension where legacy Windows Phone packages use the \*.SPKG extension. Once pushed to the device, SPKG packages are converted to CAB packages on device so package names are not unique by extension.

As a result, when conducting WSPC compliance checks between firmware submissions, packages names differing only by SPKG or CAB extension will be treated as the same package.

For example, package1.spkg will be treated as the same package as package1.cab.

### Using empty packages for WSPC compliance

In some cases, it may be necessary for you to discontinue the use of a particular package that was part of a previous **baseline** submission. However, to maintain WSPC compliance for subsequent **baseline** submissions, the deprecated package must continue to exist.

To support this scenario, you can submit an empty package of the same package name in the new baseline submission.

### Removal packages

Because the purpose of the new WSPC implementation is to remove the use of and need for removal packages, they are no longer supported for firmware submissions.

### Out of order firmware submissions

Over the course of a typical development lifecycle for firmware, we expect that the version number of new firmware will always be larger than the previous version number. However, we understand that there may be exceptions. As a result, Microsoft will also support cases where you submit firmware with versions that are less than the max firmware version submitted.

For example, assume the following baseline versions of firmware have been submitted :

Baseline firmware submission v1

Baseline firmware submission v3

Baseline firmware submission v5

Baseline firmware submission v7

Now, you submit new firmware with version 4. WSPC compliance will need to run against:

- Baseline firmware submission v3 to ensure that the v4 package set is a superset of the v3 package set (no removals between v3 and v5)
- Baseline firmware submission v5 to ensure that the v5 package set is a superset of the v4 package set (no removals between v4 and v5)

If either of the checks fail and the firmware submission v4 was submitted without the **baseline** flag, the firmware will be accepted, but the following warning will be returned:

```
<path>:\ Initialize-FirmwareSubmission -TypeOfSubmission Image -UpdateHistoryPath [path] -OemInputPath [path] -OutputFilePath [path]
```

The firmware submission is not WSPC compliant with the baseline submission {1}. This is because the packages {0} are missing from the new submission. Since this is not a new baseline, the firmware submission is allowed. However, since this submission not WSPC compliant and is not a baseline, it will not be available for a retail servicing update. Retail servicing updates are only allowed for the following scenarios: between an RTM source submission and an RTM target submission, or between a legacy non-WSPC compliant source submission and an RTM target submission. To make this submission WSPC compliant, please ensure the missing packages are included in the submission.

If either of the checks fail and the firmware submission v4 was submitted with the **baseline** parameter, the firmware will be rejected with the following error:

```
<path>:\ Initialize-FirmwareSubmission -TypeOfSubmission Image -UpdateHistoryPath [path] -OemInputPath [path] -OutputFilePath [path] -baseline
```

The firmware submission is not WSPC compliant with the baseline submission {1}. This is because the packages {0} are missing from the new submission. To make this submission WSPC compliant, please ensure the missing packages are included in the submission.

## Partial submissions

Partial firmware submissions with the [Initialize-FirmwareSubmission cmdlet](#) using the **PartialImage** parameter will still be allowed, but the submission must not be a baseline submission.

## NonWSPCCompliant parameter

The **NonWspcCompliant** parameter is not supported.

For more info, see [Initialize-FirmwareSubmission cmdlet](#).

## Request for update (RFU)

As we mentioned previously, retail servicing must be between baseline firmware submissions. By retail servicing, we mean using the [New-RequestForUpdate cmdlet](#) with the retail servicing type. Example is shown below:

```
$result = New-RequestForUpdate
-FirmwareSubmissionTicketId <Target Firmware Submission>
-RequestForUpdateType RetailServicing
-SourceFirmwareSubmissionTicketId <Source Firmware Submission>
-OemDeviceName xxx
-MOID 000-yy
```

RFUs used for retail servicing must be:

- Between two successful baseline firmware submissions
- The version of the target firmware submission must be greater than the source firmware submission

If the target firmware submissions is not a baseline, the request will be rejected and the following error will be shown:

The target firmware submission is not a Baseline submission, which is required for a retail servicing update. A retail servicing update is only allowed for the following scenarios: between a Baseline source submission and a Baseline target submission, or between a legacy non-WSPC compliant source submission and a Baseline target submission. Please also make sure you are using the current version of the Windows Phone Ingestion Client, Version {0}. Submission status in this request for update: [Target submission: {1}, {2}], [Source submission: {3}, {4}].

If neither the source nor target firmware submissions are baseline but the source firmware submission is WSPC compliant, the request will be rejected and the following error will be shown:

The source firmware submission is WSPC compliant, however it is not a Baseline submission, which is required for a retail servicing update. A retail servicing update is only allowed for the following scenarios: between a Baseline source submission and a Baseline target submission, or between a legacy non-WSPC compliant source submission and a Baseline target submission. Please also make sure you are using the current version of the Windows Phone Ingestion Client, Version {0}.

If the target firmware submissions is not a baseline, but the request is not for retail servicing, the request will be accepted and processed with a warning similar to the following:

```
$result = New-RequestForUpdate
-FirmwareSubmissionTicketId <Target Firmware Submission>
-RequestForUpdateType Trial
-SourceFirmwareSubmissionTicketId <Source Firmware Submission>
-OemDeviceName xxx
-MOID 000-yy
```

The target firmware submission is not a Baseline submission, which is required for a retail servicing update. A retail servicing update is only allowed for the following scenarios: between a Baseline source submission and a Baseline target submission, or between a legacy non-WSPC compliant source submission and a Baseline target submission. Since this is a non-retail RFU, it is being allowed, however the RFU should be made WSPC compliant before submitting the retail RFU. RFU status in this request for update: [Target submission: {0}, {1}], [Source submission: {2}, {3}].

If neither the source nor target firmware submissions are baseline but the source firmware submission is WSPC compliant, but the request is not for retail servicing, the request will be accepted and processed with a warning similar to the follow:

The source firmware submission is WSPC compliant, however it is not a Baseline submission, which is required for a retail servicing update. A retail servicing update is only allowed for the following scenarios: between a Baseline source submission and a Baseline target submission, or between a legacy non-WSPC compliant source submission and a Baseline target submission. Since this is a non-retail RFU, it is being allowed, however the RFU should be made WSPC compliant before submitting the retail RFU.

# Support for legacy WSPC compliance

Because the new WSPC compliance is a more general implementation of the previous WSPC model, legacy WSPC compliant submissions will automatically be compliant with the new model.

All existing WSPC submissions will automatically be marked as baseline as part of the rollout of this change.

## Migrating from a legacy non-compliant submission

Because our customers already have devices already with non-WSPC compliant firmware, Microsoft will provide an upgrade path for these devices to WSPC-compliant versions.

To accommodate this, Microsoft will allow retail servicing from non baseline (non-WSPC compliant) firmware submissions to baseline firmware submissions provided the following conditions are met:

- The source firmware submission version is less than the minimum version of the baseline firmware version.
- The baseline target firmware submission is WSPC-compliant with the source firmware submission.

The implication of these conditions is that the package set of the existing source non-compliant submissions will effectively become the baseline package set for WSPC compliance going forward.

For more info, see [New-RequestForUpdate cmdlet](#).

## Tools

The Microsoft Windows Phone Ingestion Client (WPIC) and Phone Image Inspector (ImgVal) have both been updated to determine if firmware submissions are WSPC-compliant or not. However, because WSPC compliance is now determined by comparison to previously submitted baseline firmware, both tools must have access to Microsoft's UTS systems. As a result, ImgVal will not be capable of running in a standalone mode. During installation, ImgVal will prompt you for your Microsoft Connect credentials as it does currently for WPIC. For more info about WPIC, see [Ingestion Client for Windows Phone](#).

### Standard validation

In addition to WSPC validation, WPIC and ImgVal will also run a new set of standard validations against firmware submissions. These standard validation rules are designed to catch common errors in firmware submissions that may cause issues during retail servicing.

Previously, this validation was bypassed for NonWspcCompliant submissions. Going forward they will be applied to all submissions. Standard validation errors are blocking and must be fixed for successful firmware submissions.

The standard validation rules are listed below:

	VALIDATION	RULE
1	Removal packages	Removal packages are not allowed. This includes SPKRs or CBS-Rs.
2	Package version	Packages cannot have a 0.0.0.0 package version.
3	Binary packages in MainOS	Packages in MainOS partition cannot be flagged as binary partition package.
4	Binary packages in EFIESP	Packages in EFIESP partition cannot be flagged as binary partition package.

	VALIDATION	RULE
5	Binary packages in UpdateOS	Packages in UpdateOS partition cannot be flagged as binary partition package.
6	PhoneFirmwareRevision	Firmware version must be in x.x.x.x format where x is between 0 and 65535 (inclusive).
7	PhoneManufacturer	Phone manufacturer name must be alphanumerical and less than or equal to 32 characters.
8	PhoneManufacturerModelName	OEM device name must be alphanumerical with -,_ and less than or equal to 32 characters.
9	PhoneMobileOperatorName	Mobile operator name must conform to XXX-XX format.

**Note** Firmware created through the Qualcomm Windows Phone Customization Tool (QWPCT) will automatically conform to the standard compliance rules above.

### Building merged packages

To conform to WSPC, OEMs should use FeatureMerger.exe to reduce the total number of packages in a firmware submissions to the greatest extent possible. For more info about the package merging process and how the merged package names are derived, see [Merging packages before imaging](#).

### Creating empty packages

Achieving WSPC compliance for a new firmware submission requires that it contain a superset of the packages contained in the previous baseline submission. An impact of this WSPC compliance rule is that packages may not be removed from new firmware submissions as compared to the previous baseline.

In some cases, it may be necessary to reconfigure your firmware in such a way that a package in the previous baseline is no longer used. In order to achieve WSPC compliance in this case, you will need to create and include an empty package in the firmware.

The following steps summarize the process of creating an empty feature merged package called Contoso.BASE.MainOS.spkg:

1. Build one empty package for the partition targeted by the package using PkgGen.exe. Ensure the <Components> element is empty. For example, the following package XML generates an empty project for the MainOS partition.

```
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
    Owner="Contoso" OwnerType="OEM" Component="EmptyPackage" SubComponent="ForMainOS"
    ReleaseType="Production" Platform=<PLAT>" Partition="MainOS">
    <Components>
    </Components>
</Package>
```

2. Reference the empty packages in the feature manifest. For example, the following feature manifest generates merged base packages and includes a combination of empty packages for the MainOS partition.

```

<FeatureManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate">
  <BasePackages>
    <PackageFile Path="C:\Packages" Name="Contoso.BASE.MainOS.spkg" Partition="MainOS" />
    <PackageFile Path="C:\EmptyPackage" Name="Contoso.EmptyPackage.ForMainOS.spkg"/>
    <PackageFile Path="C:\EmptyPackage" Name="Contoso.EmptyPackage.ForMainOS2.spkg"/>
    <PackageFile Path="C:\EmptyPackage" Name="Contoso.EmptyPackage.ForMainOS3.spkg"/>
  </BasePackages>
</FeatureManifest>

```

3. Generate merged packages by running FeatureMerger.exe with the feature manifests that reference the empty packages. The generated merged packages will contain both the empty packages and, where applicable, packages with content from the OEM.

Alternately, you can generate an empty package for the partition targeted by the package using PkgGen.exe without using feature merge. Ensure the <component> element is empty. For example, the following package XML generates an empty project for the MainOS partition called Contoso.BASE.MainOS.spkg.

```

<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
  Owner="Contoso" OwnerType="OEM" Component="BASE" SubComponent="MainOS" ReleaseType="Production"
  Platform="" Partition="MainOS">
  <Components>
  </Components>
</Package>

```

## Troubleshooting

	ACTIVITY	MESSAGE	RESOLUTION
1	Any WPIC command	The Windows Phone Ingestion Client version you are using (Version {0}) is no longer supported. Please make sure you are using the current version of the Windows Phone Ingestion Client, Version {1}.	Download and install the latest Windows Phone Ingestion Client from <a href="#">Microsoft Connect</a> .

	ACTIVITY	MESSAGE	RESOLUTION
2	Request for update	<p>WSPC compliant with the baseline submission {1}. This is because the packages {0} are missing from the new submission. Since this is not a new baseline, the firmware submission is allowed.</p> <p>However, since this submission not WSPC compliant and is not a baseline, it will not be available for a retail servicing update. Retail servicing updates are only allowed for the following scenarios: between an RTM source submission and an RTM target submission, or between a legacy non-WSPC compliant source submission and an RTM target submission. To make this submission WSPC-compliant, please ensure the missing packages are included in the submission.</p>	<p>No action is necessary if this firmware version is not going to be used for a retail update.</p> <p>However, Microsoft recommends that all new firmware submission be WSPC-compliant with the previous baseline.</p> <p>To fix the WSPC compliance issue, ensure the missing package (or empty package as appropriate) is included in the submission.</p>
3	Request for update	<p>The firmware submission is not WSPC-compliant with the baseline submission {1}. This is because the packages {0} are missing from the new submission.</p> <p>To make this submission WSPC-compliant, please ensure the missing packages are included in the submission.</p>	<p>Since this submission is a baseline, it must be WSPC-compliant with the previous baseline.</p> <p>Ensure the missing package (or empty package as appropriate) is included in the submission.</p>
4	Firmware submission	<p>To be WSPC-compliant, you cannot remove a package. The following packages were removed '{0}'.</p>	<p>Ensure the missing package (or empty package as appropriate) is included in the submission to be WSP-compliant.</p>
5	Firmware submission	<p>Using a cached package list for validation because a connection to the server cannot be established. '{0}'</p>	<p>Ensure the Image Validation tool or Windows Phone Ingestion Client have access to Microsoft's code signing and publishing systems.</p>
6	Firmware submission	<p>Could not retrieve the baseline package list while running on the client. A WSPC compliance check cannot be performed.</p>	<p>A check will be performed on the server. If error persists, contact Microsoft.</p>

	ACTIVITY	MESSAGE	RESOLUTION
7	Firmware submission	Could not retrieve the baseline package list while running on a Windows server. A WSPC compliance check cannot be performed.	Contact Microsoft.
8	Firmware submission	An exception was thrown while loading a package. {0} : {1}	Contact Microsoft.
9	Firmware submission	Could not retrieve the firmware version of the submission.	Contact Microsoft.
10	Firmware submission	There was an incorrect number of baseline submissions returned. Count: {0}	Contact Microsoft.
11	Firmware submission	No submission results were returned so a cached copy is being used.	Ensure the Image Validation tool or Windows Phone Ingestion Client have access to Microsoft's code signing and publishing systems.
12	Firmware submission	The following variant is not WSPC compliant. OEM: {0}, Mobile operator: {1}, OEM device name: {2}.	Ensure the firmware submission is targeting the correct variants. Also check the previous baseline for the offending variant and ensure all packages are present in the new submission.
13	Firmware submission	Could not get baseline package lists online for validation because connection to server cannot be established. Error: '{0}'	Ensure the Image Validation tool or Windows Phone Ingestion Client have access to Microsoft's code signing and publishing systems.
14	Firmware submission	No baseline submission result returned, this must be the first and new baseline submission for this POP.	No baseline submission result returned, this must be the first and new baseline submission for this POP. If first baseline submission, this is the correct behavior. Otherwise ensure the firmware submission is targeting the correct variants.
15	Firmware submission	Could not retrieve the latest baseline package list through partner service. Client local cache is used instead.	Ensure the Image Validation tool or Windows Phone Ingestion Client have access to Microsoft's code signing and publishing systems.

## Related topics

[Merging packages before imaging](#)

[Merging packages using FeatureMerger](#)

# Configure the Start layout

1/25/2019 • 3 minutes to read

You can now easily configure the default Start layout to include Web links, secondary tiles, folders, and apps. The converged Windows 10 Start layout requires that you create a LayoutModification.xml file, which we'll create in this walkthrough.

**Note** The schema for the LayoutModification.xml file is different from the MCSF customization answer file schema or the Windows provisioning answer file schema. You will need to use the LayoutModification.xml to fully take advantage of the Start customization in Windows 10 and you can use the Start settings in either MCSF or Windows Provisioning to reference LayoutModification.xml.

If you are not new to Windows mobile development and were using pre-existing MCSF settings pertaining to the Start layout, we highly recommend that you switch to a LayoutModification.xml to take full advantage of the Start experience. Also note that not all pre-existing MCSF Start settings are supported in Windows 10.

In this walkthrough, we will:

- Create two versions of the Start layout, one with a folder and another without a folder.
- Configure the MCSF Start layout settings to specify that we are using the new layout modification XML for our layout, creating variants in the CAF file, and specifying which Start layout applies to the variants we've created.

**Note** Make sure you've read [Start layout for Windows 10 mobile editions](#) before doing this walkthrough. The topic provides more detailed information about each element in LayoutModification.xml, which is not covered in this walkthrough, as well as limitations and restrictions that you need to be aware of.

## To configure the Start layout modification file

1. Create a file called LayoutModification1.xml.
2. Add the XML code to:
  - Pin a medium-sized tile on row 6, column 0. The tile is for the MSN News app.
  - Pin a medium-sized tile on row 6, column 2. The tile is a Web link for the Contoso home page.
  - Pin a small-sized folder on row 6, column 4. The folder name is "Contoso apps" and it contains the following:
    - A medium-sized tile for the MSN Apps app.
    - A medium-sized tile for the MSN Money app.

The following XML example shows what you need to add to the LayoutModification.xml file.

```

<?xml version="1.0" encoding="utf-8"?>
<LayoutModificationTemplate
    xmlns="http://schemas.microsoft.com/Start/2014/LayoutModification"
    xmlns:defaultlayout="http://schemas.microsoft.com/Start/2014/FullDefaultLayout"
    xmlns:start="http://schemas.microsoft.com/Start/2014/StartLayout"
    Version="1">
    <DefaultLayoutOverride>
        <StartLayoutCollection>
            <defaultlayout:StartLayout>
                <start:Group>
                    <start:Tile
                        AppUserModelID="Microsoft.BingNews_8wekyb3d8bbwe!ApplicationID"
                        Size="2x2"
                        Row="6"
                        Column="0"/>
                    <start:SecondaryTile
                        AppUserModelID="Microsoft.MicrosoftEdge_8wekyb3d8bbwe!MicrosoftEdge"
                        TileID="MyWeblinkTile"
                        Arguments="http://www.contoso.com"
                        DisplayName="Contoso Homepage"
                        Square150x150LogoUri="ms-appx:///Assets/MicrosoftEdgeSquare150x150.png"
                        Wide310x150LogoUri="ms-appx:///Assets/MicrosoftEdgeWide310x150.png"
                        ShowNameOnSquare150x150Logo="true"
                        ShowNameOnWide310x150Logo="false"
                        BackgroundColor="#FF112233"
                        Size="2x2"
                        Row="6"
                        Column="2"/>
                    <start:Folder
                        Name="Contoso apps"
                        Size="2x2"
                        Row="6"
                        Column="4">
                        <start:Tile
                            AppUserModelID="Microsoft.BingMaps_8wekyb3d8bbwe!ApplicationID"
                            Size="2x2"
                            Row="0"
                            Column="0"/>
                        <start:Tile
                            AppUserModelID="Microsoft.BingFinance_8wekyb3d8bbwe!ApplicationID"
                            Size="2x2"
                            Row="0"
                            Column="2"/>
                        <!-- Remove these comments if you have an app that you can preload and want to add
                            to the folder
                            <start:Tile
                                AppUserModelID="TBD"
                                Size="2x2"
                                Row="0"
                                Column="4"/>
                            -->
                    </start:Folder>
                </start:Group>
            </defaultlayout:StartLayout>
        </StartLayoutCollection>
    </DefaultLayoutOverride>
</LayoutModificationTemplate>

```

3. Save the xml file and note the location of the file; for example,  
*C:\Contoso\Customizations\LayoutModification1.xml*.
4. Using the same xml file, now save it as *LayoutModification2.xml*.
5. Modify the contents of the new *LayoutModification2.xml* file by deleting everything within the **start:Folder** element and replacing that folder tile location with the MSN Money app.

The contents of your XML file should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<LayoutModificationTemplate
    xmlns="http://schemas.microsoft.com/Start/2014/LayoutModification"
    xmlns:defaultlayout="http://schemas.microsoft.com/Start/2014/FullDefaultLayout"
    xmlns:start="http://schemas.microsoft.com/Start/2014/StartLayout"
    Version="1">
    <DefaultLayoutOverride>
        <StartLayoutCollection>
            <defaultlayout:StartLayout>
                <start:Group>
                    <start:Tile
                        AppUserModelID="Microsoft.BingNews_8wekyb3d8bbwe!ApplicationID"
                        Size="2x2"
                        Row="6"
                        Column="0"/>
                    <start:SecondaryTile
                        AppUserModelID="Microsoft.MicrosoftEdge_8wekyb3d8bbwe!MicrosoftEdge"
                        TileID="MyWeblinkTile"
                        Arguments="http://www.contoso.com"
                        DisplayName="Contoso Homepage"
                        Square150x150LogoUri="ms-appx:///Assets/MicrosoftEdgeSquare150x150.png"
                        Wide310x150LogoUri="ms-appx:///Assets/MicrosoftEdgeWide310x150.png"
                        ShowNameOnSquare150x150Logo="true"
                        ShowNameOnWide310x150Logo="false"
                        BackgroundColor="#FF112233"
                        Size="2x2"
                        Row="6"
                        Column="2"/>
                    <start:Tile
                        AppUserModelID="Microsoft.BingFinance_8wekyb3d8bbwe!ApplicationID"
                        Size="2x2"
                        Row="6"
                        Column="4"/>
                </start:Group>
            </defaultlayout:StartLayout>
        </StartLayoutCollection>
    </DefaultLayoutOverride>
</LayoutModificationTemplate>
```

6. Save the xml file and note the location of the file; for example,  
`C:\Contoso\Customizations\LayoutModification2.xml`.
7. Add the layout modification files and configure the Start layout settings in the MCSF CAF or Windows Provisioning answer file (WPAF).
  - For MCSF: See *Configure the Start layout* in [Configure customization settings](#).
  - For Windows Provisioning: If you are using the Windows Imaging and Configuration Designer (ICD) UI, see *Configure the Start layout* in [Use the Windows ICD UI to customize and build a mobile image](#). If you are using the Windows ICD CLI (hybrid method), see *Configure the Start layout* in [Use the Windows ICD CLI to customize and build a mobile image](#).

# Part 1: Classic mobile deployment

1/25/2019 • 2 minutes to read

In this section, we'll go through the process of customizing and building a mobile image using the classic, or Windows Phone 8.1, tools.

- [Configure customization settings](#)
- [Add a package to an OEM manifest file](#)
- [Configure the OEMInput file](#)
- [Build a mobile image using ImgGen](#)
- [Sign a mobile image](#)
- [Flash an image to a mobile device](#)

# Configure customization settings

1/25/2019 • 13 minutes to read

Customizations are ways that you can modify the Windows device UI, connectivity settings, and user experience to better reflect your brand and to meet the requirements of the network and market in which the device will ship. Customizations can include adding apps, modifying the Start layout, configuring network settings using device management, changing the default values in the Settings screen, or adding new wallpapers.

Windows 10 Mobile supports two customization frameworks: MCSF and Windows provisioning. For more information about each framework, see [Customizations for mobile devices](#).

In this section, we'll focus on adding the Start layout modification file, preloading an app, and configuring some customization settings using MCSF.

For more detailed information about the various customizations you can do, see <https://msdn.microsoft.com/library/windows/hardware/dn757433>.

## Create an MCSF customization answer file

You can use the MCSF [customization answer file](#) (CAF) to specify the settings and variants that you want to configure for a custom mobile OS image. Depending on the tools that you're using to build your image, you can use the MCSF CAF as input to ImgGen.cmd or the Windows Imaging and Configuration Designer (ICD) CLI. This answer file is based on the MCSF schema so if you decide to use another schema, such as the Windows provisioning schema, you need to write a different answer file that follows that schema instead.

If you don't have a pre-existing MCSF CAF, follow this walkthrough to learn how to create a basic MCSF CAF with multivariant support. Multivariant is the generic mechanism that lets you create a single image that can work for multiple markets by dynamically configuring the language, branding, apps, and networking settings during runtime based on the supported conditions, such as mobile operator and locale. If you are building a single variant image, you may skip this walkthrough.

### To create the MCSF CAF with multivariant support

1. Create an XML file and add the following content.

```
<?xml version="1.0" encoding="utf-8" ?>
<ImageCustomizations xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate"
    Name=""
    Description="Use to configure settings for custom mobile image."
    Owner=""
    OwnerType="OEM">

    <!-- Use to set up targets and conditions for the variants -->
    <Targets>
        <Target Id="">
            <TargetState>
                <Condition Name="" Value="" />
                <Condition Name="" Value="" />
            </TargetState>
        </Target>
        <Target Id="">
            <TargetState>
                <Condition Name="" Value="" />
                <Condition Name="" Value="" />
            </TargetState>
        </Target>
    </Targets>
</ImageCustomizations>
```

```

</Targets>

<!-- Use to specify the customizations for a single variant when used without the Variant element,
     or customizations that apply to all variants when used with the Variant element -->
<Static>
    <!-- Use to preload apps to install for all variants -->
    <Applications>
        <Application Source="" 
                      License="" 
                      ProvXML="" />
    </Applications>

    <!-- Section where you specify the settings you want to configure -->
    <Settings Path="">
        <Setting Name="" Value="" />
    </Settings>
</Static>

<!-- These settings in the Variant groups will only be applied if the associated target's
conditions are met. -->
<!-- The settings shown here will only be applied for this variant -->
<Variant Name="">
    <!-- Only one TargetRef can be used for each variant -->
    <TargetRefs>
        <TargetRef Id="" />
    </TargetRefs>

    <Settings Path="">
        <Setting Name="" Value="" />
    </Settings>

    <Settings Path="">
        <Setting Name="" Value="" />
    </Settings>
</Variant>

<!-- The settings shown here will only be applied for this variant -->
<Variant Name="">
    <!-- Only one TargetRef can be used for each variant -->
    <TargetRefs>
        <TargetRef Id="" />
    </TargetRefs>

    <Settings Path="">
        <Setting Name="" Value="" />
    </Settings>

    <Settings Path="">
        <Setting Name="" Value="" />
    </Settings>
</Variant>

</ImageCustomizations>

```

2. Specify the values for the following attributes, which determines the owner for the customizations:

- **Name** - A string that specifies the name of the customization or package. You can specify the name based on the device you're configures, such as "XDeviceCustomizations", or a generic name like "MobileCustomizations".
- **Description** - A string to help you identify the customizations defined within the CAF.
- **Owner** - A string specifying the owner, such as the OEM name.
- **OwnerType** - Set this to "OEM".

3. Specify the **Targets** to use for the variants. **Targets** describe keying for a variant and must be described or pre-declared before being referenced by the variant. To do this:

## To set up a Target

- a. Name the **Target Id**.
- b. Specify the **Condition(s)** for the target by providing the **Name** and **Value**.

For example, if we creating variants for two fictitious mobile operators, The Phone Company and Fabrikam, and we have their MCC and MNC, we can create a target for each operator using the MCC and MNC.

```
<!-- Use to set up targets and conditions for the variants -->
<Targets>
  <Target Id="Id_PhoneCo">
    <TargetState>
      <Condition Name="MCC" Value="410" />
      <Condition Name="MNC" Value="510" />
    </TargetState>
  </Target>
  <Target Id="Id_Fabrikam">
    <TargetState>
      <Condition Name="MCC" Value="310" />
      <Condition Name="MNC" Value="610" />
    </TargetState>
  </Target>
</Targets>
```

To learn more about all the supported **Condition Names** that you can use, see the section *Target, TargetState, Condition, and priorities* in [Create a provisioning package with multivariant settings](#). Note that in this walkthrough we are not using a provisioning package to declare our multivariant settings; instead, we are adding this directly into the MCSF CAF.

4. Set up the variant. To do this:

## To define a Variant

- a. Specify the **Variant Name**.
- b. Specify the **TargetRef Id** for the variant. This should match one of the **Target Ids** you specified in the **Targets** section of the CAF. Note that there should only be one **TargetRef Id** per variant.

Using our fictitious mobile operators, The Phone Company and Fabrikam, we can define the variants for each of these operators as shown in the following example:

```

<!-- The settings shown here will only be applied for The Phone Company -->
<Variant Name="ThePhoneCompany">
    <TargetRefs>
        <TargetRef Id="Id_PhoneCo" />
    </TargetRefs>

    <Settings Path="">
        <Setting Name="" Value="" />
    </Settings>
</Variant>

<!-- The settings shown here will only be applied for Fabrikam -->
<Variant Name="Fabrikam">
    <TargetRefs>
        <TargetRef Id="Id_Fabrikam" />
    </TargetRefs>

    <Settings Path="">
        <Setting Name="" Value="" />
    </Settings>

    <Settings Path="">
        <Setting Name="" Value="" />
    </Settings>
</Variant>

```

5. Name and save the XML file; for example, *C:\Contoso\Customizations\MobileCustomizations.xml*.

We'll add the static (common or variant-agnostic) and variant-specific customization settings and values in the next walkthroughs.

## Configure the Start layout

In this section, we'll use the Start MCSF settings to add the Start layout modification files that you created in [Configure the Start layout](#). We'll use one of the layout modification files as the common Start layout and we'll use the other layout modification file for one of the fictitious mobile operator variants.

1. Create and edit the MCSF CAF; for example, *C:\Contoso\Customizations\MobileCustomizations.xml*.
2. Set *C:\Contoso\Customizations\LayoutModification1.xml* as the common Start layout.

Within the **Static** section of the CAF, set `LayoutModificationFilePath` to the file path to the layout modification file.

```

<Static>
    <!-- Use to preload apps to install for all variants -->
    <Applications>
        <Application Source=""
                     License=""
                     ProvXML="" />
    </Applications>

    <!-- Section where you specify the settings you want to configure -->
    <Settings Path="StartLayoutModificationFilePath">
        <Setting Name="LayoutModificationFilePath"
Value="C:\Contoso\Customizations\LayoutModification1.xml" />
    </Settings>
</Static>

```

By specifying a new default location for the LayoutModification.xml, you are overriding the default Start layout that's in C:\Data\ProgramData\Microsoft\Start\Layouts.

- Set C:\Contoso\Customizations\LayoutModification2.xml as the default Start layout for the fictitious mobile operator, The Phone Company.

In the **Variant** section named ThePhoneCompany, set `LayoutModificationFilePath` to the file path for the second layout modification file.

```
<!-- The settings shown here will only be applied for The Phone Company -->
<Variant Name="ThePhoneCompany">
  <TargetRefs>
    <TargetRef Id="Id_PhoneCo" />
  </TargetRefs>

  <Settings Path="StartLayoutModificationFilePath">
    <Setting Name="LayoutModificationFilePath"
      Value="C:\Contoso\Customizations\LayoutModification2.xml" />
  </Settings>
</Variant>
```

**Note** `LayoutModificationFilePath` is a FirstVariationOnly setting, which means that it can only be modified during first variation, which is typically when the first valid configuration is found (such as when a SIM is inserted and a marked configuration is found for the SIM). If the configuration changes, such as during a SIM swap, the value for the FirstVariationOnly setting will not be changed again.

In this example, when the device boots after flashing a new mobile image and there is no SIM is inserted into the mobile device or a SIM for the fictitious Fabrikam mobile operator (MCC=310, MNC=610) is already in the device, LayoutModification1.xml (Start layout with a folder) will be used. If a SIM for the fictitious The Phone Company (MCC=410, MNC=510) is inserted after this, the Start layout will not change. However, if the device boots after flashing a new mobile image and there is already a SIM for The Phone Company inserted in the device, LayoutModification2.xml (Start layout with no folder) will be used instead.

## Preload apps

Partners can preload apps to be packaged and configured to install on mobile devices during the initial device setup process. In addition to preloading games, lifestyle apps, and other genres of apps, partners can preload system settings apps or partner account setup apps, just to name a few. For more information about preloading apps, see [Preinstallable apps for mobile devices](#).

**Important** In Windows 10, if you are working with an app developer or creating your own app to preload on the device, you must now request a preinstallation package for the app. For more information about this part of the process, see [Generate preinstall packages for OEMs](#). The .zip file that's returned as part of this process should contain the app's source file (such as an .appx, .appxbundle, or .xap), a provisioning file (.provxml), and a license file (.xml). If your preinstall package does not contain all of these files, you can't successfully preload the app.

### To preload an app

- Verify that the app preinstall package contains all the files you need to preload an app: source file, provisioning file, and license file.
- Add the app to the image. To do this, follow these steps:
  - Add an **Applications** section to the CAF.

```

<!-- Use to preload apps to install for all variants -->
<Applications>
    <Application Source="" 
        License="" 
        ProvXML="" />
</Applications>

```

**Note** You can add the **Applications** section within the **Static** section of the CAF, which means the app will be installed for all images regardless of the variant, or you can add it within a particular **Variant**, which means it will only be installed for a particular variant. If you are preloading more than one app, one can be common to all variants (or within the **Static** section), while another applies only to a particular variant (or within a **Variant** section). An example of the latter case can be when you have an app that you only need to install for one particular mobile operator, or country/region, and so on.

- b. Set **Source** to the location and name of your app source file; for example, *C:\Contoso\Customizations\Apps\SampleApp.appx*.
- c. Set **License** to the location and name of your app's license file; for example, *C:\Contoso\Customizations\Apps\SampleAppLicense.xml*.
- d. Set **ProvXML** to the location and name of your app's provisioning file; for example, *C:\Contoso\Customizations\Apps\mpap\_sampleapp\_001.provxml*.

The provXML file follows a prescribed naming convention. See [Preinstallable apps for mobile devices](#) for more information.

3. Save the CAF when you are done adding all the apps.

In some cases, you may need to preload an app that has dependencies on other packages or components. In this case, you need to make sure that the other packages or components are preinstalled first before your app. If the dependent packages or components are not installed first, your app preload will fail. We won't walk through this scenario here, but you can find this documented in [Preload an app with a dependency](#).

## Configure the DeviceTargetingInfo metadata for the device

In order to ship a mobile device, at a minimum, you must set the required settings described in [Phone metadata in DeviceTargetingInfo](#). Examples of required metadata include:

- OEM and mobile operator information, used for display strings in the UI, device update, connecting to the Microsoft Store, and so on.
- Hardware component versions and software versions, used for targeting updates to devices and for user support.
- The device's model name, the mobile operator's name, and the manufacturer's name, which appear in the **About** screen in **Settings**.

### To configure the DeviceTargetingInfo metadata

1. Edit the MCSF CAF; for example, *C:\Contoso\Customizations\MobileCustomizations.xml*.
2. Within the **Static** section of the CAF, add a **DeviceInfo/Static** settings group.

```

<Static>
  <!-- Other settings groups may already precede the DeviceInfo/Static group -->

  <Settings Path="DeviceInfo/Static">
    <Setting Name="PhoneManufacturer" Value="" />
    <Setting Name="PhoneManufacturerDisplayName" Value="" />
    <Setting Name="PhoneROMVersion" Value="" />
    <Setting Name="PhoneHardwareRevision" Value="" />
    <Setting Name="PhoneSOCVersion" Value="" />
    <Setting Name="PhoneFirmwareRevision" Value="" />
    <Setting Name="PhoneRadioHardwareRevision" Value="" />
    <Setting Name="PhoneRadioSoftwareRevision" Value="" />
    <Setting Name="PhoneBootLoaderVersion" Value="" />
    <Setting Name="PhoneROMLanguage" Value="" />
    <Setting Name="PhoneHardwareVariant" Value="" />
  </Settings>

</Static>

```

These settings are image-time only and will be put directly into the registry hive. Note that some settings in the `DeviceInfo/Static` group are optional so you may choose not to specify any values for them or remove them from the CAF. The following table summarizes which settings are required, optional, and which ones come from the silicon vendor.

REQUIRED SETTING	OPTIONAL SETTING	SOC VENDOR SETTING
PhoneManufacturer	PhoneManufacturerDisplayName	PhoneROMVersion
PhoneFirmwareRevision	PhoneRadioHardwareRevision	PhoneHardwareRevision
PhoneROMLanguage		PhoneSOCVersion
PhoneHardwareVariant		PhoneRadioSoftwareRevision PhoneBootLoaderVersion

\*\*Note\*\* You will need to contact your Microsoft representative to find out the value that you should use for `PhoneManufacturer`.

- Within the **Variant** section of the CAF, add a `DeviceInfo/Variant` settings group.

Using our example, we'll add the settings within the variant section for ThePhoneCompany

```

<!-- The settings shown here will only be applied for The Phone Company -->
<Variant Name="ThePhoneCompany">
  <TargetRefs>
    <TargetRef Id="Id_PhoneCo" />
  </TargetRefs>

  <!-- Other settings with the Variant section not included here for simplicity -->

  <Settings Path="DeviceInfo/Variant">
    <Setting Name="PhoneMobileOperatorName" Value="" />
    <Setting Name="PhoneManufacturerModelName" Value="" />
    <Setting Name="PhoneMobileOperatorDisplayName" Value="" />
    <Setting Name="PhoneSupportPhoneNumber" Value="" />
    <Setting Name="PhoneSupportLink" Value="" />
    <Setting Name="PhoneOEMSupportLink" Value="" />
    <Setting Name="PhonemodelName" Value="" />
    <Setting Name="RoamingSupportPhoneNumber" Value="" />
  </Settings>
</Variant>

```

These settings are first variation only and can be configured at runtime, so potentially may be based on the SIM value. Note that some settings in the `DeviceInfo/Variant` group are optional so you may choose not to specify values for them or remove them from the CAF. The following table summarizes which settings are required, optional, and which ones come from the silicon vendor.

REQUIRED SETTING	OPTIONAL SETTING
PhoneMobileOperatorName	PhoneMobileOperatorDisplayName
PhoneManufacturerModelName	PhoneSupportPhoneNumber
PhonemodelName	PhoneSupportLink PhoneOEMSupportLink PhoneRoamingSupportPhoneNumber

- Save the CAF when you are done adding all the values for the required settings or any optional settings you choose to set. Follow the guidance in [Phone metadata in DeviceTargetingInfo](#) to make sure you set the correct values and their formats.

## Configure other customization settings

There are a variety of other customization settings that you can configure for Windows mobile devices. For phones in particular, you'll also typically need to provision connectivity settings such as the MMS APN, MMS proxy, IMS services (if supported), and so on. For this walkthrough, we'll assume that you've already configured these settings so we won't cover how to configure these connectivity settings. The MCSF section of the partner documentation provides scenario-based documentation to help you identify the settings you need to configure based on the scenarios or areas you want to enable. For more information about these customizations, see:

- [Customizations for device management](#) for info on overriding the default CountryTable.xml, setting the UICC slot for branding configuration, and more.
- [Customizations for hardware components](#) for info on customizing the display, storage, touch, and so on.
- [Customizations for applications and Microsoft components](#) for info on adding a phone call/SMS filter app, active phone cover settings, and others.
- [Customizations for boot, initial setup, and shutdown](#) to learn about configuring the timezone confirmation page and language selection during device setup, and many more.

- [Customizations for browser](#) for info on ways you can customize Microsoft Edge.
- [Customizations for connectivity](#) to learn more about setting the custom percentages for signal strength bars, preferred data provider list, roaming filter, and so on.
- [Customizations for desktop experiences](#) to customize the device icon and default image that appears when the mobile device is connected to the desktop.
- [Customizations for email](#) to learn how to change the email app to always have a light background.
- [Customizations for maps](#) for info preloading map data in the user store or SD card, maps for phones shipped in China, and more.
- [Customizations for phone calls](#) to learn about branding for phone calls, setting up visual voicemail, enabling IMS services and RCS, and many more.
- [Customizations for photos, music, and videos](#) for info about audio volume limitation, adding OEM lens apps, and so on.
- [Customizations for Settings](#) to learn about all the many customizations you can configure for the settings that appear within the **Settings** app on mobile devices.
- [Customizations for SMS and MMS](#) for more info on adding encoding extension tables for SMS, maximum length for messages, intercept deny list, and many more.
- [Customizations for Start](#) to change the default behavior of the Microsoft Store live tile. Note that you may configure the Start layout too, but that's covered in [Start layout for Windows 10 mobile editions](#) and shown as an example in this walkthrough.

# Add a package to an OEM manifest file

1/25/2019 • 2 minutes to read

You can use a feature manifest (FM) file to define specific types of image builds that contain different sets of optional packages. To learn more about FM files, see [Feature manifest file contents](#). For more information about additional logic that you can add to the build system, see [Feature groupings and constraints](#).

In this walkthrough, we will add the packages you created in [Creating mobile packages](#), to an FM file to define new OEM features that you can later include to build a mobile OS image.

## To add a package to an FM file

1. Create a new FM file or modify an existing FM file to include the two packages that you created and define feature IDs for these packages. For an example of what the FM file looks like, see %WPKDCONTENTROOT%\FMFiles\arm\MSOptionalFeatures.xml in your kit installation folder.

The following example shows what your FM file may look like.

```
<?xml version="1.0" encoding="utf-8"?>
<FeatureManifest
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate">

    <!-- Other elements that you can configure
    <OEMDevicePlatformPackages>
        <PackageFile Name="Microsoft.Fake.OEMDevicePlatform.spkg"
        Path="$(mspackageroot)\firmware\Fake\$(cputype)\$(buildtype)" Device="Fake"/>
    </OEMDevicePlatformPackages>

    <BasePackages>
        <PackageFile Path="$(mspackageroot)\drivers\Fake\$(cputype)\$(buildtype)"
Name="Microsoft.Fake.AX88772.spkg"/>
    </BasePackages>
    -->

    <Features>
        <OEM>
            <PackageFile Path="SourceDirectoryA" Name="MyEchoDriver.spkg">
                <FeatureIDs>
                    <FeatureID>ECHO_DRIVER</FeatureID>
                </FeatureIDs>
            </PackageFile>
            <PackageFile Path="SourceDirectoryB"
Name="Contoso.Customization.Notifications.QuickActions.spkg">
                <FeatureIDs>
                    <FeatureID>QUICK_ACTIONS</FeatureID>
                </FeatureIDs>
            </PackageFile>
        </OEM>
    </Features>
</FeatureManifest>
```

The **OEMDevicePlatformPackages** element and **BasePackages** element are placeholders and are only there to show you what other options are available to configure in the FM file.

For more information about creating an FM file and other elements that you may need to fully define your

feature, see [Feature manifest file contents](#). For more information about additional logic that you can add to the build system, see [Feature groupings and constraints](#).

2. In the first **PackageFile** element within the **OEM** section, replace *SourceDirectoryA* with the location of the folder that contains the Echo driver package and replace *MyEchoDriver.spkg* with the real name of the .spkg containing the driver.
3. In the second **PackageFile** element within the **OEM** section, replace *SourceDirectoryB* with the location of the folder that contains the customization setting package and replace *Contoso.Customization.Notifications.QuickActions.spkg* with the real name of the .spkg containing the customization setting.
4. Save your FM file in the %WPDKCONTENTROOT%\FMFiles\arm folder. For this example, let's name the file as ContosoOptionalFeatures.xml.

# Configure the OEMInput file

1/25/2019 • 2 minutes to read

Now that you've configured your customization settings in the MCSF CAF and added custom OEM features in the OEM manifest file, you'll need to create an OEMInput file that specifies the device platform, the feature manifest files, the release type, device model, build type, languages you want to support, boot UI language, device resolutions, and other attributes like optional features that you want to include as part of your image.

For more information about each element in the OEMInput file, see [OEMInput file contents](#).

In this walkthrough, we will add the two features we defined in [Adding a package to an OEM manifest file](#), specifying the languages that we want to support, and defining the rest of our image.

## To configure the OEMInput file

1. Create a new OEMInput.xml file or modify an existing OEMInput file. For an example of what the OEMInput.xml file looks like, see %WPDKCONTENTROOT%\OEMInputSamples\Fake in your kit installation folder.

In the following example, we copied the contents of the %WPDKCONTENTROOT%\OEMInputSamples\Fake\TestOEMInput.xml file and made the following modifications:

- Added a new description.
- Added English (United Kingdom), French (France), Spanish (Spain), and Chinese (Simplified) to the list of included devices languages in addition to English (United States). We added this in the **UserInterface** section. For more information about other languages you can add to your mobile device, including how to change the default device language and regional format, see [Mobile device languages](#).
- Added the ContosoOptionalFeatures.xml feature manifest file that we created in the [Adding a package to an OEM manifest file](#) walkthrough and placed it under the **AdditionalFMs** section of the OEMInput file by adding a new **AdditionalFM** entry and specifying the location and name of the feature manifest file.
- Added the ECHO\_DRIVER and QUICK\_ACTIONS features by adding a new **OEM** subsection within the **Features** section of the OEMInput file. For each feature that we add within the **OEM** section, we add a new **Feature** entry and then specify the name of the feature that we defined in the feature manifest file.

**Note** If you're using a mobile reference device, make sure you update the values in the **SOC**, **SV**, and **Device** sections of the OEMInput.xml. You may also change the **ReleaseType** depending on what you're trying to do. Make sure you follow the guidance in [OEMInput file contents](#) when specifying values for these elements.

```
<?xml version="1.0" encoding="utf-8"?>
<OEMInput xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate">
    <Description>Windows mobile image test FFU generation</Description>
    <SOC>FAKE_SOC_Test</SOC>
    <SV>Microsoft</SV>
    <Device>FAKE</Device>
    <ReleaseType>Test</ReleaseType>
    <BuildType>fre</BuildType>
    <SupportedLanguages>
        <UserInterface>
            <Language>en-US</Language>
            <Language>en-GB</Language>
            <Language>fr-FR</Language>
            <Language>es-ES</Language>
            <Language>zh-CN</Language>
        </UserInterface>
        <Keyboard>
            <Language>en-US</Language>
        </Keyboard>
        <Speech>
            <Language>en-US</Language>
        </Speech>
    </SupportedLanguages>
    <BootUILanguage>en-US</BootUILanguage>
    <BootLocale>en-US</BootLocale>
    <Resolutions>
        <Resolution>480x800</Resolution>
    </Resolutions>
    <AdditionalFMs>
        <AdditionalFM>%WPDKROOT%\FMFiles\arm\ContosoOptionalFeatures.xml</AdditionalFM>
        <AdditionalFM>%WPDKROOT%\FMFiles\arm\MSOptionalFeatures.xml</AdditionalFM>
    </AdditionalFMs>
    <Features>
        <Microsoft>
            <Feature>IMGFAKEMODEM</Feature>
            <Feature>TEST_DISABLE_DISK_IDLE</Feature>
            <Feature>STANDARD_FEATURE_1</Feature>
            <Feature>CODEINTEGRITY_TEST</Feature>
            <Feature>SELFHOST_AUTOMATIC_DEVICE_CONFIGURATOR</Feature>
            <Feature>LOCATIONFRAMEWORKAPP</Feature>
            <Feature>FACEBOOK</Feature>
            <Feature>COMMSENHANCEMENTGLOBAL</Feature>
            <Feature>KDNETUSB_TEST_ON</Feature>
            <Feature>TESTINFRASTRUCTURE</Feature>
            <Feature>TEST</Feature>
            <Feature>STARTUPOVERRIDES</Feature>
            <Feature>GWPCERTTESTPROV</Feature>
            <Feature>MOBILECORE_TEST</Feature>
            <Feature>DRIVERS_WDTFINFRA</Feature>
            <Feature>SKYPE</Feature>
            <Feature>BOOTSEQUENCE_TEST</Feature>
            <Feature>SKIPOOBE</Feature>
            <Feature>CORTANADB_TEST_PROTECTED</Feature>
            <Feature>TEST_PROTECTED</Feature>
            <Feature>PSEUDOLOCALES</Feature>
        </Microsoft>
        <OEM>
            <Feature>ECHO_DRIVER</Feature>
            <Feature>QUICK_ACTIONS</Feature>
        </OEM>
    </Features>
    <Product>Windows Phone</Product>
</OEMInput>
```

2. Name and save your OEMInput.xml file. For our example, we named it ContosoTestOEMInput.xml and saved it in a %WPKCONTENTROOT%\ContosoOEMInput folder.

# Build a mobile image using ImgGen

1/25/2019 • 2 minutes to read

You can use two different tools to build a customized mobile image (FFU image) in Windows 10:

- Using ImgGen.cmd, which is a command file that runs the classic imaging tool, ImageApp.exe. This tool was also available earlier releases of Windows 10 Mobile including Windows Phone 8.1 and the various GDRs.
- Using the Windows Imaging and Configuration Designer (ICD) command-line interface, which is new for Windows 10.

In this walkthrough, we'll show how to use ImgGen.cmd to build the custom mobile image. In another walkthrough, we'll go through how to create another mobile image that includes the customizations we've done so far in addition to other customizations available only through Windows Provisioning by using the Windows ICD CLI.

## To build a customized image using ImgGen

1. Open a **Developer Command Prompt for VS2015** window as an administrator.
2. If you are running Windows 8.1, complete the following steps to set the USN journal registry size to 1 Mb on your development PC. Otherwise, skip to the next step.
  - a. Change the USN minimum size registry key by running the following command:

```
reg add HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem  
/v NtfsAllowUsnMinSize1Mb /t REG_DWORD /d 1
```
  - b. Reboot the PC before proceeding to the next step.
3. Run ImgGen.cmd by using the following command.

```
ImgGen TestFlash.ffuContosoTestOEMInput.xml "%WPDKCONTENTROOT%\MS Packages"  
MobileCustomizations.xml 10.0.0.1
```

This command will build an image that will be called TestFlash.ffu.

**Note** This command assumes you've gone through the rest of the walkthroughs in this section. For more information about the command-line syntax for ImgGen.cmd, see *Using ImgGen.cmd to generate the image* in [Build a mobile image using ImgGen.cmd](#).

Once the image is built, you'll need to sign it so it can be flashed to a mobile device.

# Sign a mobile image

1/25/2019 • 2 minutes to read

You need to sign a mobile image before you can deploy it to a device. For more information about signing images, see [Sign a full flash update \(FFU\) image](#).

Before you start, make sure you followed the steps in *Step 5: Install OEM test certs* in [Prepare for Windows mobile development](#). If you haven't done this yet, do this first before proceeding with the steps for signing an image.

In this walkthrough, we'll focus on test signing the image manually. In addition to ImageSigner, we'll also use Sign.cmd.

## To test sign an image

1. Open a developer prompt with administrator rights in the directory that contains the output from the image generation process.
2. Extract the catalog of the unsigned FFU file by running the following command:

**ImageSigner GETCATALOG TestFlash.ffu TestFlash.cat**

3. Sign the catalog using the /pk option. There are two parts to this step:

**Set SIGN\_OEM=1**

**Sign.cmd /pk TestFlash.cat**

4. Sign the FFU with the signed catalog file using ImageSigner.

**ImageSigner SIGN TestFlash.ffu TestFlash.cat**

Once the image is signed, you're ready to flash the image to your mobile device.

# Flash an image to a mobile device

1/25/2019 • 2 minutes to read

Flashing is the process of getting a mobile image into a mobile device. Each manufacturer has different techniques and tooling that they'll use to manufacture and service a Windows mobile device and this means that each OEM needs to determine which flashing and manufacturing process works best for them. For more information, see [Flashing tools](#).

In this walkthrough, we'll use ffutool.exe, which is installed as part of the Windows ADK, to flash the image to a mobile device. For more information about flashing and to learn more about what you need to get your device ready for flashing, see [Use the flashing tools provided by Microsoft](#).

## To flash an image

1. Boot the device into FFU flashing mode while it is connected to the host computer.

If you didn't include the LABIMAGE optional feature when generating your test image, you can force the device into FFU flashing mode manually by pressing and releasing the power button to boot the device and then immediately pressing and holding the volume up button. However, note that this option is only available if an FFU has been initially flashed to the device.

2. Open a command prompt with administrator rights.
3. Run ffutool.exe from the command line to flash the image.

**ffutool -flash TestFlash.ffu**

It will take a few minutes for the image to be fully flashed to the device. Once flashing is done, go through device setup and verify that your customizations appear as part of the image.

# Part 2: Mobile deployment using Windows Provisioning

1/25/2019 • 2 minutes to read

In this section, we'll go through the process of customizing and building a mobile image using the Windows Imaging and Configuration Designer (ICD). Windows ICD provides a hybrid method for customizing and building a mobile image because it lets you use both a Windows Provisioning answer file (WPAF) to configure the runtime settings, enterprise policies, and enrollment settings available in Windows Provisioning, and a Managed Centralized Settings Framework (MCSF) customization answer file (CAF) to fully customize the device hardware and connectivity settings, preload apps, and add assets such as ringtones and localized strings.

- [Use the Windows ICD CLI to customize and build a mobile image](#)

This method uses the Windows ICD CLI to build a customized mobile image.

- Legacy: [Use the Windows ICD UI to customize and build a mobile image](#)

Note, this method is not supported in Windows 10, version 1607.

## Requirements

Before proceeding with either of the walkthroughs, you must first complete the steps in [Prepare for Windows mobile development](#) and [Configure the Start layout](#). Windows Provisioning/Windows ICD understands MCSF CAF so if you want to learn how to create a CAF, see [Configure customization settings](#).

# Use the Windows ICD UI to customize and build a mobile image

1/25/2019 • 8 minutes to read

You can use the Windows Imaging and Configuration Designer (ICD) UI to create a new Windows 10 Mobile image and customize it by adding settings and some assets.

**Note:** This method is not supported in Windows 10, version 1607.

This imaging method requires a pre-installed OS kit so you must have all the necessary Microsoft OS packages and feature manifest files in your default install path. A configuration data file (BSP.config.xml), which contains information about the hardware component packages for your board support package (BSP), is also required. For the BSP.config.xml file, you can:

- Use the BSP.config.xml file you downloaded as part of the BSP kit, or,
- Generate your own BSP.config.xml by running the BSP kit configuration tools from the SoC vendor and selecting your component drivers.

## Build a mobile image using the Windows ICD UI

This walkthrough shows how to use the Windows ICD UI to customize, build, and flash a mobile image.

1. From the Windows ICD Start page, select **New Windows image customization**.

Or, you can also select **New Project...** from the **File** menu.

2. In the **Enter Project Details** window, specify a **Name** and **Location** for your project. Optionally, you can also enter a brief **Description** to describe your project.

3. Click **Next**.

4. If you created the project from the Start page, skip this step.

In the **Select project workflow** window, select **Imaging** from the list of available project workflows and then click **Next**.

5. In the **Select imaging source format** window, select **The Windows image is based on Microsoft packages**, and then click **Next**.

You will be prompted to specify a BSP.config.xml file.

6. In the **Select hardware component drivers** window, click **Browse** to launch File Explorer and search for the location of your BSP.config.xml file.

7. Click **Finish**.

This loads all the customizations that you can configure based on the Windows edition that you selected. Once all the available customizations are loaded, you can see the **Customizations Page**.

8. In the **Customizations Page**, select the settings you want to customize from the **Available Customizations** pane.

For this walkthrough, see [Configure customizations in the Windows ICD UI](#) for a list of the settings we're using as examples. Once you're done configuring the settings, proceed to the next step.

9. Although optional, export a provisioning package to encapsulate the settings you just configured and allow you to reuse all or most of the customizations for another project.

To do this, click the **Export** dropdown in the main menu, select **Provisioning Package**, add the required package information.

- **Name** - Name to use for the package, for example, *Contoso\_ppkg*.
- **ID** - Auto-generated package GUID.
- **Owner** - Package owner. Set to *OEM*.
- **Version** - Version info. This is pre-populated with the latest package version number or "1.0". Leave it set to the default, *1.0*, although you can set it to any version that you want.
- **Rank** - Package rank, which is a value between 0 and 99 (inclusive). Leave it set to the default, *0*.

Click **Next** until you get to the **Build the provisioning package** screen. Click **Build** to build the package and then click **Finish**.

10. Select **Create** from the main menu and then choose **FFU**.

11. In the **Select image type** screen, choose **Test** for the image type.

Although there are other image types, a Test image type is good when your image is not yet final and you are still testing various components in your image. The image is also unlocked and doesn't contain any security enforcements.

12. Click **Next**.

13. In the **Describe the image** screen, select the languages that you want to include as part of your image.

If you went through [Part 1: Classic mobile deployment](#), the settings in this step are similar to how you would set the device languages in [Configure the OEMInput file](#).

- **User Interface Languages** - These are the display language(s) to install on the device.

Select *en-gb*, *en-us*, *es-es*, *fr-fr*, and *zh-cn*.

- **Keyboard Languages** - These are additional keyboard languages to use for text correction and suggestions while typing on the device.

Select *en-us*.

- **Speech Languages** - These are the speech languages that you want to install on your device.

Select *en-us*.

14. Because we selected multiple languages in **User Interface Languages**, we need to select the default display language that the device will use when it is first turned on by the user. To do this, choose **Boot Language** and select the default device language that you want to set. For example, set this to *en-us*.

15. To set the country or region, choose **Boot Locale** and select the locale for the default region or country. Any locale can be used as the regional format, but only the country (GeOID) value is used. For example, set this to *en-us*.

16. Click **Next**.

17. To change the location where the files are saved, click **Browse...** under **Select where to save the files**.

You may change the location where the files are saved, but typically the default location is fine.

18. Optional. If you created a CAF file by following [Configure customization settings](#), you can include this by selecting **Browse...** under **Customization answer file (optional)** and specifying the location where you saved the CAF.

For example, C:\Contoso\Customizations\MobileCustomizations.xml.

19. To change the default location where you want to save the image, click **Browse...** to launch File Explorer and specify a new location.

To use the default location, click **Next**.

20. Click **Build** to start building the image. The project information is displayed in the build page and the progress bar indicates the build status.

If you need to cancel the build, click **Cancel**. This cancels the current build process, closes the wizard, and takes you back to the **Customizations Page**.

21. During the image build process, a lot of what's happening during the build process is shown in the build output window. This window shows:

- Warnings that might appear while the image is building.
- Verbose build messages to indicate the phases within the image build process.
- Error messages such as when the input files have schema errors or when the image fails to build.

If your build fails, an error message will be displayed. You can review the build log to identify the issue by clicking **View in Notepad**.

If your build is successful, the name of the image and its location will be displayed.

- If you choose, you can build the image again by picking a different image type, selecting different languages, and then starting another build. To do this, click **Back** to select what you want to change, and then click **Next** to start another build.
- Boot the device into image or FFU download mode. To force your phone into image or FFU download mode manually, press and release the power button to reboot the phone and then immediately press and hold the volume up button. Note that this option is available only after an initial FFU has been flashed to the phone.

If this doesn't work, check and follow the device flashing instructions provided by the SoC vendor.

- If you are ready to flash the built image to your device, click **Flash** and select the target device to flash the FFU. If you don't find the device listed in the list of available target device(s), click **Refresh**.

If you want to flash the image to the device later, follow the steps in [Deploy an image to a mobile device](#) when you are ready to flash the image to your device.

It will take a few minutes for the image to be fully flashed to the device. Once flashing is done, go through device setup and verify that your customizations appear as part of the image.

- If you are done, click **Finish** to close the wizard and go back to the **Customizations Page**.

## Configure customizations in the Windows ICD UI

**Note** When configuring customizations using Windows ICD, do not use the **Image time settings**, use the **Runtime settings** instead. If you configure the image-time settings, this will cause an error due to a settings collision if the setting is configured in both WPAF and MCSF CAF.

If you haven't done so already, you must first create the LayoutModification.xml files as shown in [Configure the Start layout](#) before proceeding with the steps in this section.

### To configure the Start layout

1. In the **Available customizations** pane in Windows ICD, expand **Runtime settings**, select **Start**, and then

select **StartLayout**.

2. In the middle pane, click **Browse** to open File Explorer.
3. In the File Explorer window, navigate to the location where you saved LayoutModification1.xml from step 1; for example, C:\Contoso\Customizations.
4. Select LayoutModification1.xml and then click **Open**.

This sets the value of **StartLayout** and the setting should appear in the **Selected customizations** page.

Enterprise policies and enrollment settings are some of the customizations available only through Windows Provisioning. Here, we'll configure a few of these policies to include as part of the image. For more information about the other policies that you can configure, see [Policies](#) (for the Windows Provisioning settings). Note that the Windows Provisioning settings topics do not provide a detailed description of each policy; instead, each topic links to the more detailed information in [Policy CSP](#).

#### To set policies

1. In the **Available customizations** pane, expand **Runtime settings**, and select **Policies**.
2. Find **Policies/DeviceLock** and set **MaxInactivityTimeDeviceLock** to "15".

This specifies that after the device has been idle for 15 minutes, the device will become PIN or password locked.

3. Find **Policies/DeviceLock** and set **ScreenTimeoutWhileLocked** to "15".

This specifies the duration, in seconds, for the screen timeout while on the lock screen. For this example, the duration is 15 seconds.

## Deploy an image to a mobile device

Follow these steps if you deferred flashing the image to the device after it was built.

1. Boot the device into image or FFU download mode. To force your phone into image or FFU download mode manually, press and release the power button to reboot the phone and then immediately press and hold the volume up button. Note that this option is available only after an initial FFU has been flashed to the phone.

If this doesn't work, check and follow the device flashing instructions provided by the SoC vendor.

2. Using a USB cable, connect your phone to the host computer.
3. Click **Deploy** from the main menu and choose **To USB connected device** to deploy the FFU image to the device.
4. In the **Select an FFU image** window, click **Browse...** to launch File Explorer and select the FFU that you want to flash to your target device, and then click **Next**.
5. Choose the target device or drive from the list. If your device or drive is not listed, click **Refresh**.
6. Click **Next**.
7. In the **Deploy to device** window, choose **Flash** to start flashing the image.
8. Click **Finish** to close the **Deployment** page.

# Use the Windows ICD CLI to customize and build a mobile image

1/25/2019 • 8 minutes to read

You can use the Windows Imaging and Configuration Designer (ICD) command-line interface (CLI) to generate a new Windows 10 Mobile image.

This imaging method requires a pre-installed OS kit so you must have all the necessary Microsoft OS packages and feature manifest files in your default install path. You also need either a `BSP.config.xml` file, which contains information about the hardware component packages for your board support package (BSP) or you can use an `OEMInput.xml` file.

If you're using a `BSP.config.xml` file, you can:

- Use the `BSP.config.xml` file you downloaded as part of the BSP kit, or,
- Generate your own `BSP.config.xml` by running the BSP kit configuration tools from the SoC vendor and selecting your component drivers.

## Create a WPAF with multivariant support

Multivariant provides a generic mechanism for creating a single image that can work for multiple markets. You can use it to dynamically configure language, branding, and network settings during runtime based on the mobile operator and locale/country.

Unlike the Windows ICD UI, you can use the Windows ICD CLI to create an image that has multivariant support. In order to do this, you must first edit a Windows Provisioning answer file (WPAF), `customizations.xml`, and add the **Targets** and **Variant** sections to the file. [Create a provisioning package with multivariant settings](#) provides a more information about multivariant support in Windows 10 and a list of the conditions that Windows supports along with their priorities. It also provides a step-by-step example on what you need to do.

In this section, we'll modify the `customizations.xml` file that was created from [Use the Windows ICD UI to customize and build a mobile image](#) and include **Targets** and **Variant** sections to support multivariant. If you are building a single variant image, you may skip this section.

1. Locate the provisioning package file, `customizations.xml`. If you didn't change the default project location, you can find the package in `<drive:>\Users\<user_name>\Documents\Windows Imaging and Configuration Designer (WICD)\<project_name>`.
2. Use an XML or text editor to open the `customizations.xml` file.

The following example shows the contents of the `customizations.xml` file created in [Use the Windows ICD UI to customize and build a mobile image](#).

```
<?xml version="1.0" encoding="utf-8"?>
<WindowsCustomizations>
  <PackageConfig xmlns="urn:schemas-Microsoft-com:Windows-ICD-Package-Config.v1.0">
    <ID>{239b9121-9f26-42db-8ae2-0d62989caa66}</ID>
    <Name>Contoso_ppkg</Name>
    <Version>1.0</Version>
    <OwnerType>OEM</OwnerType>
    <Rank>0</Rank>
  </PackageConfig>
  <Settings xmlns="urn:schemas-microsoft-com:windows-provisioning">
    <Customizations>
      <Common>
        <Policies>
          <DeviceLock>
            <MaxInactivityTimeDeviceLock>15</MaxInactivityTimeDeviceLock>
            <ScreenTimeoutWhileLocked>15</ScreenTimeoutWhileLocked>
          </DeviceLock>
        </Policies>
        <Start>
          <StartLayout>C:\Contoso\Customizations\LayoutModification1.xml</StartLayout>
        </Start>
      </Common>
    </Customizations>
  </Settings>
</WindowsCustomizations>
```

3. Edit the customizations.xml file and create a **Targets** section to describe the conditions that will handle your multivariant settings.

The following example shows the customizations.xml, which has been modified to include the conditions **MCC** and **MNC**. For parity, the example uses the same fictitious IDs and conditions that were used in *Create an MCSF customization answer file* section in [Configure customization settings](#).

```

<?xml version="1.0" encoding="utf-8"?>
<WindowsCustomizations>
    <PackageConfig xmlns="urn:schemas-Microsoft-com:Windows-ICD-Package-Config.v1.0">
        <ID>{239b9121-9f26-42db-8ae2-0d62989caa66}</ID>
        <Name>Contoso_ppkg</Name>
        <Version>1.0</Version>
        <OwnerType>OEM</OwnerType>
        <Rank>0</Rank>
    </PackageConfig>
    <Settings xmlns="urn:schemas-microsoft-com:windows-provisioning">
        <Customizations>
            <Common>
                <Policies>
                    <DeviceLock>
                        <MaxInactivityTimeDeviceLock>15</MaxInactivityTimeDeviceLock>
                        <ScreenTimeoutWhileLocked>15</ScreenTimeoutWhileLocked>
                    </DeviceLock>
                </Policies>
                <Start>
                    <StartLayout>C:\Contoso\Customizations\LayoutModification1.xml</StartLayout>
                </Start>
            </Common>
            <Targets>
                <Target Id="Id_PhoneCo">
                    <TargetState>
                        <Condition Name="MCC" Value="410" />
                        <Condition Name="MNC" Value="510" />
                    </TargetState>
                </Target>
                <Target Id="Id_Fabrikam">
                    <TargetState>
                        <Condition Name="MCC" Value="310" />
                        <Condition Name="MNC" Value="610" />
                    </TargetState>
                </Target>
            </Targets>
        </Customizations>
    </Settings>
</WindowsCustomizations>

```

4. In the customizations.xml file, create two **Variant** sections:

- Specify a **Name** for each variant.
- Add a child **TargetRefs** element.
- Within the **TargetRefs** element, add a **TargetRef** element.

The following example shows what the customizations.xml looks like after adding two variants, ThePhoneCompany and Fabrikam, and then using the two **Target Ids** we defined in the previous step to use as the **TargetRef Id** for each variant.

```

<?xml version="1.0" encoding="utf-8"?>
<WindowsCustomizations>
    <PackageConfig xmlns="urn:schemas-Microsoft-com:Windows-ICD-Package-Config.v1.0">
        <ID>{239b9121-9f26-42db-8ae2-0d62989caa66}</ID>
        <Name>Contoso_ppkg</Name>
        <Version>1.0</Version>
        <OwnerType>OEM</OwnerType>
        <Rank>0</Rank>
    </PackageConfig>
    <Settings xmlns="urn:schemas-microsoft-com:windows-provisioning">
        <Customizations>
            <Common>
                <Policies>
                    <DeviceLock>
                        <MaxInactivityTimeDeviceLock>15</MaxInactivityTimeDeviceLock>
                        <ScreenTimeoutWhileLocked>15</ScreenTimeoutWhileLocked>
                    </DeviceLock>
                </Policies>
                <Start>
                    <StartLayout>C:\Contoso\Customizations\LayoutModification1.xml</StartLayout>
                </Start>
            </Common>
            <Targets>
                <Target Id="Id_PhoneCo">
                    <TargetState>
                        <Condition Name="MCC" Value="410" />
                        <Condition Name="MNC" Value="510" />
                    </TargetState>
                </Target>
                <Target Id="Id_Fabrikam">
                    <TargetState>
                        <Condition Name="MCC" Value="310" />
                        <Condition Name="MNC" Value="610" />
                    </TargetState>
                </Target>
            </Targets>
            <Variant Name="ThePhoneCompany">
                <TargetRefs>
                    <TargetRef Id="Id_PhoneCo" />
                </TargetRefs>
            </Variant>
            <Variant Name="Fabrikam">
                <TargetRefs>
                    <TargetRef Id="Id_Fabrikam" />
                </TargetRefs>
            </Variant>
        </Customizations>
    </Settings>
</WindowsCustomizations>

```

- Move compliant settings from the **Common** section to the **Variant** section.

**Note** Settings that reside in the **Common** section are applied unconditionally on every triggering event.

In the following example, we used the `DeviceLock/MaxInactivity` policy under ThePhoneCompany variant while the `DeviceLock/ScreenTimeoutWhileLocked` policy was moved under the Fabrikam variant.

```
```XML
<?xml version="1.0" encoding="utf-8"?>
<WindowsCustomizations>
  <PackageConfig xmlns="urn:schemas-Microsoft-com:Windows-ICD-Package-Config.v1.0">
    <ID>{239b9121-9f26-42db-8ae2-0d62989caa66}</ID>
    <Name>Contoso_ppkg</Name>
    <Version>1.0</Version>
    <OwnerType>OEM</OwnerType>
    <Rank>0</Rank>
  </PackageConfig>
  <Settings xmlns="urn:schemas-microsoft-com:windows-provisioning">
    <Customizations>
      <Common>
        <Start>
          <StartLayout>C:\Contoso\Customizations\LayoutModification1.xml</StartLayout>
        </Start>
      </Common>
      <Targets>
        <Target Id="Id_PhoneCo">
          <TargetState>
            <Condition Name="MCC" Value="410" />
            <Condition Name="MNC" Value="510" />
          </TargetState>
        </Target>
        <Target Id="Id_Fabrikam">
          <TargetState>
            <Condition Name="MCC" Value="310" />
            <Condition Name="MNC" Value="610" />
          </TargetState>
        </Target>
      </Targets>
      <Variant Name="ThePhoneCompany">
        <TargetRefs>
          <TargetRef Id="Id_PhoneCo" />
        </TargetRefs>
        <Policies>
          <DeviceLock>
            <MaxInactivityTimeDeviceLock>15</MaxInactivityTimeDeviceLock>
          </DeviceLock>
        </Policies>
      </Variant>
      <Variant Name="Fabrikam">
        <TargetRefs>
          <TargetRef Id="Id_Fabrikam" />
        </TargetRefs>
        <Policies>
          <DeviceLock>
            <ScreenTimeoutWhileLocked>15</ScreenTimeoutWhileLocked>
          </DeviceLock>
        </Policies>
      </Variant>
    </Customizations>
  </Settings>
</WindowsCustomizations>
```

```

6. Save the updated customizations.xml file and note the path to this updated file. You will need the path as one of the values when you get ready to build the image.
7. Use the Windows ICD command-line interface (CLI) to create a provisioning package using the updated customizations.xml. For more information about how to build a provisioning package and a description of

the command switches and parameters, see **To build a provisioning package** in [Use the Windows ICD command-line interface](#).

For example:

```
icd.exe /Build-ProvisioningPackage /CustomizationXML:"C:\CustomProject\customizations.xml"
/PackagePath:"C:\CustomProject\output.ppkg" /StoreFile:C:\Program Files (x86)\Windows
Kits\10\Assessment and Deployment Kit\Imaging and Configuration Designer\x86\Microsoft-Common-
Provisioning.dat"
```

In this example, the **StoreFile** corresponds to the location of the settings store that will be used to create the package for the required Windows edition.

**Note** The provisioning package created during this step will contain the multivariant settings. You can use this package either as a standalone package that you can apply to a Windows device, use it as the base when starting another project, or use it as one of the inputs (**/ProvisioningPackage**) when building either a Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) image or Windows 10 Mobile image.

## Build a mobile image using the Windows ICD CLI

This walkthrough shows how to use the Windows ICD CLI to build a mobile image. For more information about the Windows ICD CLI, including usage examples and parameter descriptions, see [Use the Windows ICD command-line interface](#).

1. Open a command-line window with administrator rights.
2. From the command-line, navigate to the Windows ICD install directory:
  - On an x64 computer, go to: C:\Program Files (x86)\Windows Kits\10\Assessment and Deployment Kit\Imaging and Configuration Designer\x86
  - On an x86 computer, go to: C:\Program Files\Windows Kits\10\Assessment and Deployment Kit\Imaging and Configuration Designer\x86
3. Using the updated customizations.xml (with multivariant settings) and the MCSF CAF created in [Configure customization settings](#), use the Windows ICD CLI to build a mobile image.

To do this with the example files and using a bsp.config.xml, see the following command:

```
icd.exe /Build-ImageFromPackages /ImagePath:"C:\Contoso\Customizations\TestFlash2.ffd"
/BSPConfigFile:"C:\ContosoXDevice.bsp.config.xml" /ImageType:Test
/CustomizationXML:"C:\Contoso\Customizations\customizations.xml" /OEMCustomizationVer:1.0.0.0
/MCSFCustomizationXML:"C:\Contoso\Customizations\MobileCustomizations.xml"
```

Here are a few things to keep in mind:

- Replace all the placeholder values for each parameter with the values that match your assets and directory locations
- Specify /ImageType because we are using /BSPConfigFile
- Use /CustomizationXML to point to the customizations.xml
- Windows ICD requires /OEMCustomizationVer if ProvisioningPackage is defined
- Make sure the format for the /OEMCustomizationVer version number is in <Major>.<Minor>. <SubVersion>.<SubMinorVersion>, such as 1.0.0.0

To do this with the example files and using an OEMInput.xml file, see the following command:

```
icd.exe /Build-ImageFromPackages /ImagePath:"C:\Contoso\Customizations\TestFlash2.ffd"
```

**/OEMInputXML:**"C:\Contoso\ContosoTestOEMInput.xml"  
**/CustomizationXML:**"C:\Contoso\Customizations\customizations.xml" **/OEMCustomizationVer:**1.0.0.0  
**/MCSFCustomizationXML:**"C:\Contoso\Customizations\MobileCustomizations.xml"

Once the image (FFU) is built, you can flash it to your mobile device by using ffutool.exe or the **Deploy** option in the Windows ICD UI. See the following section for more information.

## Flash an image to a mobile device

There are two ways that you can use to flash an image to a mobile device:

- Using ffutool.exe, or,
- Using the built-in flashing functionality in Windows ICD

This section shows how to do both. Choose one of these methods to flash your image to your mobile device.

Follow these steps if you are flashing the image to the device using Windows ICD.

### To flash an image using Windows ICD

1. Boot the device into image or FFU download mode. To force your phone into image or FFU download mode manually, press and release the power button to reboot the phone and then immediately press and hold the volume up button. Note that this option is available only after an initial FFU has been flashed to the phone.

If this doesn't work, check and follow the device flashing instructions provided by the SoC vendor.

2. Using a USB cable, connect your phone to the host computer.
3. Launch Windows ICD.
4. Click **Deploy** from the main menu and choose **To USB connected device** to deploy the FFU image to the device.
5. In the **Select an FFU image** window, click **Browse...** to launch File Explorer and select the FFU that you want to flash to your target device, and then click **Next**.
6. Choose the target device or drive from the list. If your device or drive is not listed, click **Refresh**.
7. Click **Next**.
8. In the **Deploy to device** window, choose **Flash** to start flashing the image.
9. Click **Finish** to close the **Deployment** page.

Follow these steps if you are flashing the image to the device using ffutool.exe.

### To flash an image using ffutool.exe

1. Boot the device into FFU flashing mode while it is connected to the host computer.

If you didn't include the LABIMAGE optional feature when generating your test image, you can force the device into FFU flashing mode manually by pressing and releasing the power button to boot the device and then immediate pressing and holding the volume up button. However, note that this option is only available if an FFU has been initially flashed to the device.

2. Open a command prompt with administrator rights.
3. Run ffutool.exe from the command line to flash the image.

**ffutool -flash TestFlash.ffu**

No matter which method you used to flash the image to the device, it will take a few minutes for the image to be fully flashed. Once flashing is done, go through device setup and verify that your customizations appear as part of the image.

# Manufacturing Mode

1/25/2019 • 2 minutes to read

Introduced in Windows 10 Mobile, manufacturing mode is a mode of the full operating system that can be used for manufacturing-related tasks, such as component and support testing.

In Windows Phone 8.1, you had to flash the Microsoft Manufacturing Operating System (MMOS) to do manufacturing tests and processes, such as test hardware, blow fuses, and provision security keys. Once the tests were completed, you had to flash the full operating system. This added extra time on the manufacturing floor.

This new feature allows you to boot into a manufacturing mode of the full operating system (and do those manufacturing steps) without having to flash an MMOS image.

## Manufacturing profiles

A manufacturing profile defines settings that should be used when the operating system boots in manufacturing mode. The device can have more than one manufacturing profile. A profile named Default is included with Windows 10 Mobile. The default profile contains the settings for Microsoft components that let the device boot into a minimal environment for Manufacturing Mode.

Manufacturing profiles are stored in the registry on the device in the following location:

**HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control\ManufacturingMode** You must create a subkey for each manufacturing profile. Under the profile key, you can change the settings for some select operating system components for when the system is booting in manufacturing mode. For example, you can alter which services are started when this manufacturing profile is enabled. You can add your own services in the Services subkey, as shown below. If you want to set all services to the same start type, you can use an \* for the service name. If the \* wildcard is not used, all Win32 services that are not included in the manufacturing profile will use their default start type.

**Note** The \* wildcard only applies to Win32 services, excluding kernel-mode drivers.

The following example creates a manufacturing profile named CustomProfile, causes the service named OEMFactoryTestService to automatically start, and all other Win32 services to demand start:

```
[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\ManufacturingMode\CustomProfile]  
  
[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\ManufacturingMode\CustomProfile\Services]  
  
[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\ManufacturingMode\CustomProfile\Services\OEMFactoryTestService]  
"Start"=dword:00000002  
  
[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\ManufacturingMode\CustomProfile\Services\*]  
"Start"=dword:00000003
```

The manufacturing profile name must be less than 64 characters.

You cannot use **Current** for the name of your manufacturing profile. This name is reserved for the currently active manufacturing profile.

# Create a custom manufacturing profile package

1/25/2019 • 2 minutes to read

You can add manufacturing profiles to your device by using a package. For more info on creating packages, see [Creating packages](#). In some cases, you may need to create a custom profile. For example, perhaps you want to fine tune what Microsoft services are running for performance or functionality reasons or perhaps you want to have more than one manufacturing profile. When creating new custom profiles, you should start by copying the provided default profile and customizing it to suit your needs.

For example, if you wanted to create a new custom profile for factory testing that is a copy of the default profile but also starts your factory test service, it may look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
    Owner="Contoso"
    Component="MfgMode"
    SubComponent="ManufacturingModeServices"
    ReleaseType="Production"
    OwnerType="OEM">

    <Macros>
        <Macro Id="MfgMode" Value="$(hkLM.control)\ManufacturingMode" />
        <Macro Id="CustomProfileServices" Value="$(MfgMode)\CustomProfile\services" />
    </Macros>

    <Components>
        <OSComponent>
            <!-- Overrides copied from default profile -->
            <RegKeys>
                <RegKey KeyName="$(CustomProfileServices)\*"\>
                    <RegValue Name="Start" Type="REG_DWORD" Value="00000003" />
                </RegKey>
            </RegKeys>
            <RegKeys>
                <RegKey KeyName="$(CustomProfileServices)\AccountProvSvc"\>
                    <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
                </RegKey>
            </RegKeys>
            <RegKeys>
                <RegKey KeyName="$(CustomProfileServices)\adss"\>
                    <RegValue Name="Start" Type="REG_DWORD" Value="00000004" />
                </RegKey>
            </RegKeys>
            <RegKeys>
                <RegKey KeyName="$(CustomProfileServices)\AudioSrv"\>
                    <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
                </RegKey>
            </RegKeys>
            <RegKeys>
                <RegKey KeyName="$(CustomProfileServices)\BFE"\>
                    <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
                </RegKey>
            </RegKeys>
            <RegKeys>
                <RegKey KeyName="$(CustomProfileServices)\BrokerInfrastructure"\>
                    <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
                </RegKey>
            </RegKeys>
            <RegKeys>
                <RegKey KeyName="$(CustomProfileServices)\cellUserModeInterconnect"\>
                    <RegValue Name="Start" Type="REG_DWORD" Value="00000001" />
                </RegKey>
            </RegKeys>
        </OSComponent>
    </Components>

```

```
    <RegValue Name="Start" Type="REG_DWORD" Value="00000004" />
</RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\DcomLaunch">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\DHCP">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\Fusion">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000004" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\KeepWifiOnSvc">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000003" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\LaunchAppSvc">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000004" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\MPSSvc">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\NETACT">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000004" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\nsi">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\Power">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\ProfSvc">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\RpcEptMapper">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\RpcSs">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\SamSs">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
    </RegKey>
</RegKeys>
<RegKeys>
```

```

<RegKey KeyName="$(CustomProfileServices)\SecMgr">
    <RegValue Name="Start" Type="REG_DWORD" Value="00000004" />
</RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\SirepSvc">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000004" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\SystemEventsBroker">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
    </RegKey>
</RegKeys>
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\TestSirepSvc">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
    </RegKey>
</RegKeys>
<!-- Custom overrides for OEM services -->
<RegKeys>
    <RegKey KeyName="$(CustomProfileServices)\OEMFactoryTestService">
        <RegValue Name="Start" Type="REG_DWORD" Value="00000002" />
    </RegKey>
</RegKeys>
</OSComponent>
</Components>
</Package>

```

You can then create the package by using pkggen.exe (included with the Windows Driver Kit):

```
pkggen.exe example.pkg.xml /config:pkggen.cfg.xml
```

# Create a custom manufacturing profile package with USBFN settings

1/25/2019 • 2 minutes to read

You can also specify custom USBFN settings in a manufacturing profile that are only used when the device is in manufacturing mode.

When the device is not in Manufacturing Mode, USBFN settings will still be read from the normal location. When the device is in Manufacturing Mode and Manufacturing Mode-specific settings have been provided, the settings are from a different location. If no settings have been provided in the active manufacturing profile, the settings will be read from the normal location. This allows you to use different settings when the device is in manufacturing mode versus when it is not.

Here's an example of a manufacturing profile package that specifies USBFN settings:

```
<?xml version='1.0' encoding='utf-8'?>
<Package
    xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
    Owner="Microsoft"
    Component="UsbFn"
    SubComponent="Settings"
    ReleaseType="Production"
    OwnerType="Microsoft"
    >
<Macros>
    <Macro
        Id="MfgMode"
        Value="$(hk1m.control)\ManufacturingMode"
        />
    <Macro
        Id="MfgModeProfile"
        Value="Default"
        />
    <Macro
        Id="MfgModeUsbFn"
        Value="$(MfgMode)\$(MfgModeProfile)\USBFN"
        />
</Macros>
<Components>
    <OSComponent>
        <RegKeys>
            <RegKey
                KeyName="$(MfgModeUsbFn)\Default"
                >
                <RegValue
                    Name="bcdDevice"
                    Type="REG_DWORD"
                    Value="00000001"
                    />
            <RegValue
                Name="bDeviceClass"
                Type="REG_DWORD"
                Value="00000000"
                />
            <RegValue
                Name="bDeviceProtocol"
                Type="REG_DWORD"
                Value="00000000"
                />
        </RegKeys>
    </OSComponent>
</Components>
```



```
    />
<RegValue
    Name="MSOSExtendedPropertyDescriptor"
    Type="REG_BINARY"

    Value="74,01,00,00,00,01,05,00,05,00,84,00,00,01,00,00,00,28,00,44,00,65,00,76,00,69,00,63,00,65,00,49,00,6E
    ,00,74,00,65,00,72,00,66,00,61,00,63,00,65,00,47,00,55,00,49,00,44,00,00,00,4E,00,00,00,7B,00,32,00,36,00,66,00
    ,65,00,64,00,63,00,34,00,65,00,2D,00,36,00,61,00,63,00,33,00,2D,00,34,00,32,00,34,00,31,00,2D,00,39,00,65,00,34
    ,00,64,00,2D,00,65,00,33,00,64,00,34,00,62,00,32,00,63,00,35,00,63,00,35,00,33,00,34,00,7D,00,00,00,36,00,00,00
    ,04,00,00,00,24,00,44,00,65,00,76,00,69,00,63,00,65,00,49,00,64,00,6C,00,65,00,45,00,6E,00,61,00,62,00,6C,00,65
    ,00,64,00,00,00,04,00,00,01,00,00,00,34,00,00,00,04,00,00,00,22,00,44,00,65,00,66,00,61,00,75,00,6C,00,74,00
    ,49,00,64,00,6C,00,65,00,53,00,74,00,61,00,74,00,65,00,00,00,04,00,00,00,01,00,00,00,38,00,00,00,04,00,00,00,26
    ,00,44,00,65,00,66,00,61,00,75,00,6C,00,74,00,49,00,64,00,6C,00,65,00,54,00,69,00,6D,00,65,00,6F,00,75,00,74,00
    ,00,00,04,00,00,00,10,27,00,00,44,00,00,00,04,00,00,00,32,00,55,00,73,00,65,00,72,00,53,00,65,00,74,00,44,00,65
    ,00,76,00,69,00,63,00,65,00,49,00,64,00,6C,00,65,00,45,00,6E,00,61,00,62,00,6C,00,65,00,64,00,00,04,00,00,00
    ,01,00,00,00"
    />
</RegKey>
</RegKeys>
</OSComponent>
</Components>
</Package>
```

You can then create the package by using pkggen.exe (included with the Windows Driver Kit):

```
pkggen.exe exampleUSBFN.pkg.xml /config:pkggen.cfg.xml
```

# Define a service that only runs in Manufacturing Mode

1/25/2019 • 2 minutes to read

There may be cases where you want a service to be normally disabled but automatically run when the device is in Manufacturing Mode. For example, your factory test suite would probably run in this configuration. You would set your service start type to Disabled and then add a manufacturing profile override that makes your service auto-start when using the appropriate manufacturing mode profile.

Here's an example manufacturing profile:

```

<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
    Owner="Contoso"
    Component="MfgMode"
    SubComponent="FactoryTest"
    ReleaseType="Production"
    OwnerType="OEM">

    <Macros>
        <Macro Id="MfgMode" Value="$(hkLM.control)\ManufacturingMode" />
        <Macro Id="CustomProfileServices" Value="$(MfgMode)\CustomProfile\services" />
    </Macros>

    <Components>
        <!-- Definition for the service -->
        <Service
            Name="FactoryService"
            Start="Disabled"
            Type="Win32OwnProcess"
            ErrorControl="Ignore"
            DisplayName="OEMFactoryTestService"
            Description="A Sample OEM Factory Test Service">

            <Executable Source="$(BINARY_ROOT)\bin\factory\oem_factory_test_service.exe" />

            <!-- Failure actions should reset once per day, for security reasons -->
            <FailureActions ResetPeriod="86400">
                <Action Type="RestartService" Delay="1000"/>
                <Action Type="RestartService" Delay="1000"/>
                <Action Type="RestartService" Delay="1000"/>
                <!-- if it fails to start three times, services should just stop -->
                <Action Type="None" Delay="0"/>
            </FailureActions>

            <RequiredCapabilities>
                <!-- Needed to access and create RPC endpoints -->
                <RequiredCapability CapId="ID_CAP_INTEROPSERVICES" />
            </RequiredCapabilities>
        </Service>
    </Components>
</Package>

```

# Create a full operating system manufacturing profile

1/25/2019 • 2 minutes to read

When you boot into Manufacturing Mode, can you skip OOBE and preinstall apps to run your manufacturing tests. Skipping OOBE can speed up manufacturing time.

To set this up, you add the apps that should be installed with the manufacturing profile. For more info about manufacturing profiles, see [Manufacturing Mode](#).

## Add apps to a custom manufacturing profile

In a manufacturing mode profile, you must create a new registry key that lists the apps that should be installed with the manufacturing profile:

```
HKEY_LOCAL_MACHINE\System\ControlSet001\Control\ManufacturingMode\<Profile Name>\Apps\OobeInstall
```

Under the OobeInstall registry key, you must create registry key (with REG\_DWORD type) for each app. The name of the registry key must match the filename of the app package.

For example, to add the Settings app to the manufacturing profile, you would add a registry key named **systemsettings**.

Here are some things to consider when adding apps to a manufacturing profile:

- The manufacturing profile must not disable any Windows services.
- The value of the registry key is reserved. Only the registry key name is used.
- The app can be a first or second party app, but the app package must be a part of the image.
- The \* wildcard can be used in the name of the app.
- If you want the normal OOBE experience (with all apps installed), create a single registry key with the name of \\*.

### Find the name of the app

The name of the registry key must match the filename of the app package. You can get a list of the apps that are on the device by running the following command on the device's drive root:

```
dir /S MPAP_*.provxml
```

This command returns a list of files, similar to the following:

```
MPAP_systemsettings_001.provxml
```

The part of the filename after **MPAP\_** and before **\_0xx.provxml** is what you should use for the registry key name.

## Add the registry keys to your custom manufacturing profile package

You add the registry keys to your custom manufacturing profile package like you would with any other package. For more info about packaging, see [Creating Phone Packages](#).

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
    Owner="Contoso"
    Component="MfgMode"
    SubComponent="Manufacturing.FactoryTestSample"
    ReleaseType="Production"
    OwnerType="OEM">

    <Macros>
        <Macro Id="MfgMode" Value="$(hkLM.control)\ManufacturingMode" />
        <Macro Id="CustomProfile" Value="$(MfgMode)\CustomProfile" />
    </Macros>

    <Components>
        <OSComponent>
            <RegKeys>
                <RegKey KeyName="$(CustomProfile)\"/>
                <RegKey KeyName="$(CustomProfile)\Services"/>
                <RegKey KeyName="$(CustomProfile)\Apps"/>
                <RegKey KeyName="$(CustomProfile)\Apps\OobeInstall">
                    <RegValue Name="FactoryTestOEMSample" Value="0" Type="REG_DWORD"/>
                    <RegValue Name="systemsettings" Value="0" Type="REG_DWORD"/>
                </RegKey>
            </RegKeys>
        </OSComponent>
    </Components>
</Package>
```

# Detect Manufacturing Mode

1/25/2019 • 2 minutes to read

You can determine whether the device is in Manufacturing Mode or not by using either a kernel mode or user mode API.

In kernel mode, a new API has been defined in wdm.h:

```
_IRQL_requires_max_(APC_LEVEL)
NTKERNELAPI
BOOLEAN
ExIsManufacturingModeEnabled (
    VOID
);
```

Here's an example of how you might use it:

```
BOOLEAN ManufacturingModeEnabled = FALSE;

NTSTATUS
DriverEntry(
    PDRIVER_OBJECT DriverObject,
    PUNICODE_STRING RegistryPath
)
{
...
    ManufacturingModeEnabled = ExIsManufacturingModeEnabled();
...
}

NTSTATUS
DoManufacturingOperation(
    VOID
)
{
    if (ManufacturingModeEnabled == FALSE) {
        return STATUS_NOT_SUPPORTED;
    }
...
    return STATUS_SUCCESS;
}
```

In user mode, the API is defined in sysinfoapi.h:

```
WINBASEAPI
BOOL
WINAPI
GetOsManufacturingMode(
    _Out_ PBOOL pbEnabled
);
```

Here's an example of how you might use it:

```
DWORD
DoManufacturingOperation(
    VOID
)
{
    DWORD Error = ERROR_SUCCESS;
    BOOL ManufacturingModeEnabled = FALSE;

    if (!GetOsManufacturingMode(&ManufacturingModeEnabled)) {
        Error = GetLastError();
    }

    if ((Error != ERROR_SUCCESS) ||
        (ManufacturingModeEnabled == FALSE)) {
        return ERROR_NOT_SUPPORTED;
    }
    ...
    return ERROR_SUCCESS;
}
```

# Enable or Disable Manufacturing Mode

1/25/2019 • 2 minutes to read

If you want to test Manufacturing Mode, you can enable it by using ffutool.exe or by using a BCD setting.

**Note** The recommended way to support manufacturing mode on shipping devices is to have the firmware support the Boot mode management UEFI protocol. For more info on this protocol, see [Boot mode management UEFI protocol](#).

## Enable or disable Manufacturing Mode with ffutool.exe

If the device supports the boot mode UEFI protocol, you can enable or disable Manufacturing Mode with ffutool.exe by using the **setBootMode** parameter. The syntax is as follows:

```
ffutool.exe -setBootMode <boot mode> <profile name>
```

- *boot mode* -- an integer that corresponds to the boot mode documented in [EFI\\_BOOT\\_MODE\\_INFO enumeration](#).
- *profile name* -- the name of the manufacturing profile to enable. This is required when the boot mode is set to 1 and is ignored when the boot mode is set to 0.

The following example enables Manufacturing Mode and uses a manufacturing profile named CustomProfile:

```
ffutool.exe -setBootMode 1 CustomProfile
```

The following example disables Manufacturing Mode, allowing the operating system to boot normally:

```
ffutool.exe -setBootMode 0
```

## Enable Manufacturing Mode with a BCD setting

You can use the MfgMode BCD setting to test Manufacturing Mode with your custom manufacturing profiles. MfgMode is a string value that is set to the name of your custom manufacturing profile.

For example, you can start the device in Manufacturing Mode using the default manufacturing profile by running the following command on the device:

```
bcdedit.exe /set {default} mfgmode "default"
```

Or, you could start the device in Manufacturing Mode using a custom manufacturing profile named, CustomProfile, by doing the following:

```
bcdedit.exe /set {default} mfgmode "CustomProfile"
```

You can also enable it on an offline device that is plugged in and is in USB mass storage mode. For example:

```
bcdedit.exe /store "D:\EFI\ESP\efi\Microsoft\Boot\BCD" /set {default} mfgmode "default"
```

**Note** If you're using an older version of bcddedit.exe, you might have to use **custom:22000140** instead of **mfgmode** as the BCD setting name.

## Create a Manufacturing Mode BCD package

You can create a package that creates the MfgMode BCD entry and sets it to your custom manufacturing profile. To do this, you must first create an XML file that references the BCD entry:

```
<?xml version="1.0" encoding="utf-8"?>
<BootConfigurationDatabase xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.microsoft.com/phone/2011/10/BootConfiguration">
    <Objects>

        <!-- Global Settings Group -->
        <Object SaveKeyToRegistry="false">
            <FriendlyName>Windows Loader</FriendlyName>
            <Elements>

                <Element>
                    <DataType>
                        <WellKnownType>Manufacturing Mode</WellKnownType>
                    </DataType>
                    <ValueType>
                        <StringValue>Default</StringValue>
                    </ValueType>
                </Element>

            </Elements>
        </Object>

    </Objects>
</BootConfigurationDatabase>
```

After that is created, you can reference it in a package XML file:

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
    Owner="Contoso"
    Component="MfgMode"
    SubComponent="ManufacturingModeBcd"
    ReleaseType="Production"
    Partition="EFIESP"
    OwnerType="OEM">

    <Components>
        <BCDStore Source=".\\exampleBcd.bcd.xml"/>
    </Components>
</Package>
```

**Note** There is a Partition attribute defining that these BCD entries need to apply to the EFIESP partition. This should be updated to be the partition where the BCD store for your device resides. If this is different from the partition where the main operating system resides, other operations such as adding files and registry keys to the main operating system partition cannot be done from the same package.

To create the package, you can use pkggen.exe (included with the Windows Driver Kit):

```
pkggen.exe exampleBcd.pkg.xml /config:pkggen.cfg.xml
```

# Optional features for Manufacturing Mode

1/25/2019 • 2 minutes to read

The following features can be used with devices running in Manufacturing Mode.

**Note** All optional features included with Windows 10 Mobile can be used, too. For more info about the other optional features, see [Optional features for building images](#).

FEATURE	DESCRIPTION
BCDEDIT	Adds bcdedit.exe to the image for development support. This should not be included in final retail images.
MFGMODEBCD	Adds the BCD entry to enable Manufacturing Mode in the full OS Image. You should remove this before the device leaves the factory floor.
MFGSTARTUP	Adds startup.bsc file to the Manufacturing Mode image and is needed whenever the MOBILECOREBOOTSH feature is used.
MFGDEVTOOLS	Adds a collection of useful developer tools to the Manufacturing Mode image, such as tlist.exe, sc.exe, shutdown.exe. This should not be included in the final Manufacturing Mode image.
MFGTELNETFTP	Adds Telnet and TFTP servers to the Manufacturing Mode image. You can leave them in the final image for servicing since they only run when the device is in Manufacturing Mode.
MFGTSHELL	Adds TShell capability to the Manufacturing Mode image. You can leave it in the final image for servicing since it only runs when the device is in Manufacturing Mode.
MOBILECOREBOOTSH	Enables the bootsh service (bootshscv) so that features in startup.bsc, such as telnet and ftp, can be used. This is required when the MFGTELNETFTP feature is specified. You can leave it in the image for servicing since it only runs when the device is in Manufacturing Mode.

# Boot mode management UEFI protocol

1/25/2019 • 2 minutes to read

The boot mode management protocol is used to determine which boot mode the operating system should use when it starts.

## EFI\_BOOT\_MODE\_MGMT\_PROTOCOL

This section provides a detailed description of the **EFI\_BOOT\_MODE\_MGMT\_PROTOCOL**.

### GUID

```
#define EFI_BOOT_MODE_MGMT_PROTOCOL_GUID \
{ 0xBE464946, 0x9787, 0x4FEB, { 0xBD, 0x71, 0x14, 0xC1, 0xC5, 0x2D, 0x69, 0xD1 } }
```

### Revision number

```
#define EFI_BOOT_MODE_MGMT_PROTOCOL_REVISION 0x00010000
```

### Protocol interface structure

```
typedef struct _EFI_BOOT_MODE_MGMT_PROTOCOL {
    UINT32             Revision;
    EFI_SET_BOOT_MODE_INFO SetBootModeInfo;
    EFI_GET_BOOT_MODE_INFO GetBootModeInfo;
} EFI_BOOT_MODE_MGMT_PROTOCOL;
```

### Members

#### Revision

The revision to which the **EFI\_BOOT\_MODE\_MGMT\_PROTOCOL** adheres. All future revisions must be backward compatible. If a future version is not backward compatible, a different GUID must be used.

#### GetBootModeInfo

Determines the boot mode which the operating system should use when it starts. See

[EFI\\_BOOT\\_MODE\\_MGMT\\_PROTOCOL.GetBootModeInfo](#)

#### SetBootModeInfo

Specifies the boot mode the operating system should use when it starts, including an optional profile name. See

[EFI\\_BOOT\\_MODE\\_MGMT\\_PROTOCOL.SetBootModeInfo](#)

## Requirements

**Header:** User generated

## Related topics

[Manufacturing Mode](#)

# EFI\_BOOT\_MODE\_INFO enumeration

1/25/2019 • 2 minutes to read

Defines the possible boot modes that the operating system can use when it starts.

## Syntax

```
typedef enum _EFI_OS_BOOT_MODE {  
    EfiOsBootModeFullOs = 0,  
    EfiOsBootModeManufacturingOs = 1  
    EfiOsBootModeMax  
} EFI_OS_BOOT_MODE, *PEFI_OS_BOOT_MODE;
```

## Constants

### EfiOsBootModeFullOs

The device should boot normally into the operating system.

### EfiOsBootModeManufacturingOs

The device is in manufacturing mode.

## Related topics

[Boot mode management UEFI protocol](#)

# EFI\_BOOT\_MODE\_MGMT\_PROTOCOL.GetBootModelInfo

1/25/2019 • 2 minutes to read

This function is used to retrieve the boot mode and optional profile name from the UEFI firmware.

```
typedef EFI_STATUS  
(EFIAPI *EFI_GET_BOOT_MODE_INFO)(  
    IN  struct _EFI_BOOT_MODE_MGMT_PROTOCOL *This,  
    OUT EFI_OS_BOOT_MODE                 *BootMode,  
    IN OUT UINT32                        *ProfileNameElements,  
    OUT CHAR16                           *ProfileName  
>;
```

## Parameters

*This*

[in] A pointer to the **EFI\_BOOT\_MODE\_MGMT\_PROTOCOL** instance.

*BootMode*

[out] A pointer to the enumeration that holds the boot mode of the device.

*ProfileNameElements*

[in] [out] A pointer to a UINT32 value that receives the number of characters in the profile name.

*ProfileName*

[out] A pointer to the name of the current profile.

## Return values

Returns one of the following status codes:

RETURN CODE	DESCRIPTION
EFI_SUCCESS	Success
EFI_NOT_FOUND	The boot mode data was not found.
EFI_VOLUME_CORRUPTED	A required storage partition is not initialized or is corrupted.
EFI_INVALID_PARAM	An invalid parameter was passed to the function.
EFI_BUFFER_TOO_SMALL	Not enough space in the provided buffer.

## Requirements

**Header:** User generated

## Related topics

[Boot mode management UEFI protocol](#)

# EFI\_BOOT\_MODE\_MGMT\_PROTOCOL.SetBootModelInfo

1/25/2019 • 2 minutes to read

This function supplies a boot mode and optional profile name to the firmware to use on subsequent boots.

```
typedef EFI_STATUS  
(EFIAPI *EFI_SET_BOOT_MODE_INFO)(  
    IN struct _EFI_BOOT_MODE_MGMT_PROTOCOL *This,  
    IN EFI_BOOT_MODE *BootMode,  
    IN UINT32 *ProfileNameElements OPTIONAL,  
    IN CHAR16 *ProfileName OPTIONAL  
>;
```

## Parameters

*This*

[in] A pointer to the **EFI\_BOOT\_MODE\_MGMT\_PROTOCOL** instance.

*BootMode*

[in] A pointer to the enumeration that holds the boot mode of the device.

*ProfileNameElements*

[in] A pointer to a UINT32 value of the number of characters in the profile name to set.

*ProfileName*

[in] A pointer to the name of the boot mode profile to set.

## Return values

Returns one of the following status codes:

RETURN CODE	DESCRIPTION
EFI_SUCCESS	Success
EFI_NOT_FOUND	The boot mode data was not found.
EFI_VOLUME_CORRUPTED	A required storage partition is not initialized or is corrupted.
EFI_INVALID_PARAM	An invalid parameter was passed to the function.
EFI_BAD_BUFFER_SIZE	The ProfileName name string is too long.

## Requirements

**Header:** User generated

## Related topics

[Boot mode management UEFI protocol](#)

# Microsoft Manufacturing OS

1/25/2019 • 2 minutes to read

Microsoft Manufacturing OS (MMOS), an optimized configuration of the operating system, facilitates efficient manufacturing. MMOS is intended to provide the following capabilities:

- Fast boot time resulting from a smaller image that contains only the code necessary to support manufacturing test execution. The smaller OS image can be flashed quickly, accelerating the manufacturing process.
- The ability to create an OEM-customized MMOS image that includes unique drivers and test applications.
- Support for OEM-developed component-level testing, in addition to full-function quality and calibration testing.
- Access to low-level native APIs for direct testing of phone hardware components.
- The ability to create a mechanism to remotely control the test application and retrieve test logs over a USB communication channel.
- No dependency on display or touch hardware, to allow for headless operation.
- The ability to directly boot into MMOS in a manufacturing environment.
- Automated launch of OEM test applications when MMOS is booted.
- The ability to write to the DPP partition to store unique, per-phone data.
- Battery charging is supported in both UEFI and MMOS. It is set using a feature setting, so partners can turn it on and off as desired. For more info, see [MMOS image definition](#).

## Working with MMOS

This section describes how to build and flash an MMOS image.

To work with MMOS, complete the following tasks:

1. Create your test application or other manufacturing tools as packages. For more info, see [Creating packages](#).
2. Create a flashing protocol and add packages to support that functionality to MMOS. For more info, see [Flashing tools](#).
3. Create the MMOS image. For more info, see [MMOS image definition](#).
4. Flash the MMOS image to the device. For more info, see [Flash MMOS to the device](#).
5. Deploy manufacturing applications. For more info, see [Deploy and test a user-mode test application in MMOS](#).
6. Evaluate if you want to create a WIM to be able to run MMOS in RAM. For more info, see [Working with WIM MMOS images](#).

## Developing user mode test applications

To create a user mode test applications to test the device in manufacturing, complete the following tasks:

1. Develop the user mode test application using the MMOS library. For more info, see [Manufacturing test environment supported APIs](#) and [Develop MMOS test applications](#).
2. A service can be used to start a user mode test application. If desired, the test application can communicate with a test controller on a PC to control the testing process and to log results.

To configure the service to start automatically when the OS boots, set the **Start** attribute to "Auto" in the service package configuration file.

```
<Components>
  <Service
    Name="SampleMMOSSvc"
    Start="Auto"
    ...
```

This capability is only available in MMOS.

3. Some apps should only be allowed to run in MMOS. You can use an API to determine if MMOS is running or not. For more info, see [Determine if MMOS is running](#).
4. To cause the device to enter the connected standby power state, call the SetScreenOff function. For more info, see [Calling SetScreenOff to enter connected standby](#).

## Related topics

[Develop MMOS test applications](#)

# MMOS image definition

1/25/2019 • 7 minutes to read

This section provides instructions for creating Windows 10 Mobile MMOS images. This process is similar to the process for creating the primary OS image.

## Creating an MMOS image

The MMOS image created during this process requires packages from the SoC manufacturer and the OEM in addition to those provided by Microsoft. The OEM packages that are included in the sample MfgOEMInput.xml configuration file are for illustration purposes only. OEMs should remove those example packages and replace them with their own OEM packages. It is up to the OEM to determine which set to include in MMOS. OEMs can add additional OEM packages containing test applications and test controllers, etc.

### Important

All the imaging and packaging tools are located in %WPDKCONTENTROOT%\Tools\bin\i386. This path must be included in your environment Path for the tools to work. You must run these tools from a Visual Studio command-line window as an Administrator with access to MakeCat.exe in the environment path.

### Creating OEM packages

Adding content to an image is done by creating your own packages. OEM packages can be added to an image by modifying the MfgOEMInput.xml file before creating an image with Windows Imaging and Configuration Designer (ICD).

For instructions on creating packages, see [Creating packages](#).

### Specifying features in MMOS

You can use the **Feature** element in the MfgOEMInput.xml file to include optional packages provided by Microsoft. You can modify the input XML file to support different features in MMOS for development, manufacturing, and retail service needs.

### Manufacturing development and debugging environment

The following features are defined and supported in the manufacturing development and debugging environment. This environment can be used to create test applications for use in manufacturing.

### Important

The following features can only be used for test-signed packages and are not to be included in Customer Care WIM Images.

### General features

FEATURE	DESCRIPTION
ENABLE_BOOT_KEYS_TEST	Enables a boot menu that is launched by pressing and holding the power button. Use the Volume key to navigate and the Camera key to select. This feature is mutually exclusive with <b>ENABLE_BOOT_KEYS_RETAIL</b> .
MOBILECOREBOOTSH	Enables the bootsh service (bootshscv) so that features in startup.bsc, such as telnet and ftp, can be used.
LABIMAGE	This feature causes the device to enter the FFU download mode automatically when the device is booted. For more info, see <a href="#">Use the flashing tools provided by Microsoft</a> .
MFGTSHHELL	Enables TShell in MMOS and manufacturing mode. If you use this, you need to set the TestSirepServer service to auto in your manufacturing profile.

FEATURE	DESCRIPTION
OPTIMIZED_BOOT	Modifies the behavior of the OS boot process to start some system processes and services before all device drivers are started. Enabling this feature may decrease boot time, but it may also cause regressions in boot behavior in some scenarios.
BOOTKEYACTIONS_RETAIL	This feature enables a set of button actions for use in retail devices.
DISABLE_FFU_PLAT_ID_CHECK	Disables the device platform validation in the Microsoft flashing application. For more information about the platform check in flashing, see <a href="#">Use the flashing tools provided by Microsoft</a> .
	<p><b>Important</b> The device platform validation for flashing must not be disabled in retail images.</p>
PERF_TRACING_TOOLS	Contains tools for doing performance analysis, such as tools for stopping and merging ETL tracing.
ENABLE_BOOT_PERF_BASIC_TRACING	Enables boot performance tracing events to be generated.
ENABLE_BOOT_PERF_CPU_PROFILING_TRACING	Enables CPU profiling events on top of what ENABLE_BOOT_PERF_BASIC_TRACING enables.

## Debug features

Use the following settings to specify the transport that is used for debugging. If a debugging feature is not specified, debugging will not be enabled in MMOS.

FEATURE	DESCRIPTION
KDNETUSB_ON	Includes all kernel debugger transports and enables KDNET over USB.
KDUSB_ON	Includes all kernel debugger transports and enables KDUSB.
	<p><b>Note</b> Do not include either KDUSB_ON or KDNETUSB_ON if you need to enable MTP or IP over USB in the image. If the kernel debugger is enabled in the image and the debug transports are used to connect to the phone, the kernel debugger has exclusive use of the USB port and prevents MTP and IP over USB from working.</p>
KDSERIAL_ON	Includes all kernel debugger transports and enables KDSERIAL with the following settings: 115200 Baud, 8 bit, no parity.
KD	Includes all kernel debugger transports in the image, but does not enable the kernel debugger.
MFGCRASHDUMPSUPPORT	This feature enables offline crash dump functionality by using the wpcrdmp method for MMOS images. When the device crashes, it activates wpcrdmp and saves a dump of the memory and SOC subsystems to the disk for an offline investigation.

FEATURE	DESCRIPTION
MWDBGSRV	This feature adds support for the user mode debug server.

The previous DEBUGGERON feature has been deprecated.

### Other debug features

FEATURE	DESCRIPTION
	Dumpsize setting features - The following three features must only be used with the Test and Health image types. These features must not be used with retail images. Only one dumpsize setting can be selected at a time.
DUMPSIZE512MB	Specifies a pre-allocated crash dump file size of 592 MB. This is intended for a phone with 512 MB of memory.
DUMPSIZE1G	Specifies a pre-allocated crash dump file size of 1104 MB. This is intended for a phone with 1024 MB of memory.
DUMPSIZE2G	Specifies a pre-allocated crash dump file size of 2128 MB. This is intended for a phone with 2048 MB of memory.
	Dump data storage location - The next two settings control if crash dump data is stored on eMMC or if crash dump data is stored on an SD card. Only one of these settings can be selected at a time. These two features must only be used with the Test, Health and Selfhost image types. These features must not be used with retail images.
DEDICATEDDUMPONEMMC	Specifies that the DedicatedDumpFile location as c:\crashdump\dedicatedddump.sys.  <b>Note</b> This cannot be used with DEDICATEDDUMPONSDCARD.
DEDICATEDDUMPONSD	DEDICATEDDUMPONSD – Specifies that the DedicatedDumpFile location as d:\dedicatedddump.sys  When DEDICATEDDUMPONSD is used, crash dump will be disabled if the user removes the SD card or if the card is not present when the device booted. To re-enable crash dump:  1. Set this registry key <code>HKLM\System\CurrentControlSet\Control\CrashControl\CrashDumpEnabled</code> to the value of <code>HKLM\System\CurrentControlSet\Control\CrashControl\ExpectedCrashDuration</code>  2. Reboot the device.  <b>Note</b> This cannot be used with DEDICATEDDUMPONEEMC.

FEATURE	DESCRIPTION
DBGCHKDISABLE	<p>This feature disables debugger connection checking and the debugger connection status is ignored.</p> <p>The effect on the debugger behavior is different depending on the SoC that is being used.</p> <ul style="list-style-type: none"> <li>For QC8x26 and QC8974 devices, include DBGCHKDISABLE to ensure that offline dumps will be generated even if the device is connected to a debugger. Otherwise, an Offline Dump (Bug Check Code 0x14C dump) will not be created but a raw dump will still be created.</li> <li>For 8960 devices, include DBGCHKDISABLE to ensure that offline dumps will be generated even if the device is connected to a debugger. Otherwise, an Offline Dump (i.e. Bug Check Code 0x14C dump) will not be created.</li> </ul>
MSVCRT_DEBUG	<p>This feature adds support for explicit linking of debug c-runtime libs by including msycop120d.dll and msocr120d.dll in the image. For more information, see this topic on MSDN: <a href="#">CRT Library Features</a>.</p>
MWDBGSRV	<p>This feature adds support for the user mode debug server.</p>
NOLIVEDUMPS	<p>Disables non-fatal kernel error reports. These reports contain debugging information for OS and driver developers.</p>
TELEMETRYONSDCARD	<p>This feature enables temporary storage of debugging logs and files on the SD card. This feature is only appropriate for test/self-host images and only on devices with less than 8 GB of free space of primary storage.</p>

### Customer Care WIM Images

The following features are supported in MMOS in the manufacturing production environment.

FEATURE	DESCRIPTION
ENABLE_BOOT_KEYS_RETAIL	<p>Enables a set of buttons on the phone for use in retail device. This feature is mutually exclusive with <b>ENABLE_BOOT_KEYS_TEST</b> (see the previous table).</p>
ENABLE_IP_OVER_USB	<p>Enables IP Over USB.</p>
ENABLE_USB_COMPOSITE	<p>Enables the systems on a chip (SoC) provided composite USB stack in MMOS.</p>

### Dual use features

These MMOS features can be used with either test or retail customer care images.

ENABLE_MMOS_CHARGING	Enables battery charging when running in MMOS.
----------------------	--

UEFI charging must be either disabled or enabled for MMOS to work. Only one of these options can be set at a time.

ENABLE_UEFI_CHARGING	Enables battery charging when running in UEFI.
----------------------	--

DISABLE_UEFI_CHARGING	Disables battery charging when running in UEFI.
-----------------------	---

There are additional features that are defined in other image types of the operating system that are not supported in MMOS. This list is not exhaustive, but it provides examples of features that are not supported in the manufacturing or retail environments:

- TESTINFRASTRUCTURE
- IMGFAKELED
- LABIMAGE
- LOCATIONFRAMEWORKAPP

#### Configuring the MMOS MfgOEMInput.xml file

1. Open the sample MfgOEMInput.xml file using a text editor. By default, this file is installed to %WPDKCONTENTROOT%\OEMInputSamples\8960Fluid.
2. Add needed MMOS features as described previously.
3. Add any required OEM packages to the file.
4. Locate the **Product** element and confirm that it is set to "Manufacturing OS".

You can optionally update the description to record information about the OS image.

```
<Description>Manufacturing OS generation for ARM.fre</Description>
<Product>Manufacturing OS</Product>
```

5. Locate the **SOC** and **Device** elements and change them as necessary.
6. Add %WPDKCONTENTROOT%\Tools\bin\i386 to your environment **Path** variable.

# Flash MMOS to the device

1/25/2019 • 2 minutes to read

After the MMOS image definition is complete, the image can be flashed to the device

1. Flashing on the host side is accomplished through a connection established with WinUSB, the Microsoft generic USB device driver. The necessary drivers are included by default in Windows 8 and Windows 10. In Windows 7, the necessary drivers can be installed from Windows Update. To configure a Windows 7 computer to install the necessary drivers, click **Start**, type “Device Installation Settings”, select **Yes, do this automatically (recommended)**, and then click **Save Changes**.
2. Put the device in flashing mode by holding the volume up button while powering up the device. After the device is in flashing mode, connect the USB cable to your computer.
3. Verify that the device is detected in Device Manager as *WinUsb Device*.
4. To be able to use the ffutool, sign, and ImageSigner tools, add %WPDKCONTENTROOT%\Tools\bin\i386 to your environment **Path** variable.
5. Prior to flashing the FFU file to the device, you must sign the FFU file using the following syntax in the command prompt. The cat file is generated with the FFU, when using the ImgGen tool.

```
c:\>sign <path to cat file>
c:\>ImageSigner SIGN <path to ffu> <path to cat file>
```

For example:

```
c:\>sign MMOS.cat
c:\>ImageSigner SIGN MMOS.ffu MMOS.cat
```

6. At a command prompt, run the ffutool command, which uses the following syntax:

```
c:\>ffutool -flash <path to ffu image file>
```

For example:

```
c:\>ffutool -flash C:\MMOS\MMOS.ffu
```

7. You should see output similar to the following.

```
Logging to ETL file: C:\Users\Nancy\AppData\Local\Temp\ffutool18276.etl
Found device:
Name: Contoso.DCD6000 Phone.DCD6000
ID: 00000045-14ca-3016-8fbe-120000000000
Flashing: MMOS.ffu
[=====] 100.00%
Transferred 157.88 MB in 50.56 seconds, overall rate 3.12 MB/s.
```

8. The device will reboot into MMOS. The display on the device will show a small rotating graphic.

# Working with WIM MMOS images

1/25/2019 • 2 minutes to read

You can temporarily copy a WIM (Windows Imaging) image over to a device and then boot to that image running in volatile RAM memory. This capability can be used to service the device using MMOS. For more info about MMOS, see [Microsoft Manufacturing OS](#). The advantage of using this approach for servicing is that you will not need to reserve space on the retail OS for code that is only used in servicing. Minimizing the space that is consumed by the OS is an important consideration in low cost devices.

**Important** Only MMOS test images are currently supported. Retail signing is not currently supported.

## Creating a WIM Image

Before you create a WIM image, you must complete the steps described in the following topics:

### [MMOS image definition](#)

In addition, follow these guidelines to prepare an image so that it will operate properly when converted to a WIM image.

- The following XML shows an example of the Microsoft features that can be used in a test WIM Image.

```
<Microsoft>
  <Feature>MOBILECOREBOOTSH</Feature>
  <Feature>ENABLE_BOOT_KEYS_TEST</Feature>
  <Feature>ENABLE_IP_OVER_USB</Feature>
</Microsoft>
```

You may want to add additional features such as battery charging (ENABLE\_MMOS\_CHARGING) depending on your needs. For additional info about the MMOS features, see [MMOS image definition](#).

- Do not include packages that access other partitions on the device. Because the WIM is loaded and running in RAM, it is not able to access other partitions and the OS cannot contain packages that reference other partitions.

After you complete the steps in the preceding topics, use the **ImgToWIM** command to convert the signed FFU image to a WIM image. The ImgToWim executable is located in %WPDKCONTENTROOT%\Tools\bin\i386. The usage is summarized here.

```
ImgToWIM <FFUFileName> <WIMFileName>
```

When you enter the **ImgToWim** command, you should see output that is similar to the following.

```
C:\TestWIM>ImgToWim MMOS.ffu MMOSWim.wim
Reading the image file: MMOS.ffu
ETW Log Path: C:\Users\USER1\AppData\Local\Temp\storage_session_1210.etl
OS Version: Microsoft Windows NT 6.2.9200.0
OpenDiskInternal: Creating empty virtual disk.
No mounted WP disks found.
Storage Service: Created a new image in 0.7 seconds.
AddAccessPath: Mount point for volume MainOS: C:\Users\USER1\AppData\Local\Temp
\oji20cv1.mq5.mnt\.

Creating WIM at 'C:\TestWIM\MMOSWim.wim' ...

Capturing 'MainOS'...
WIM creation complete.
DismountFullFlashImage:[2.87] Cleaning up temporary paths.
CleanupTemporaryPaths: Cleaning up temporary path C:\Users\USER1\AppData\Local\
Temp\oji20cv1.mq5.mnt\.
Storage Service: Dismounting the image in 2.9 seconds.
```

## Booting from a WIM Image

Use the FFUTool **WIM** option to boot from a WIM image. The usage is summarized here.

```
ffutool -WIM <WIMFileName.wim>
```

To boot the device from a WIM image, complete the following steps.

1. Set up the PC side flashing tools.
2. Put the device in flashing mode by holding the volume up button while powering up the device. After the device is in flashing mode, connect the USB cable to your computer.
3. Use the **FFUTool** command with the **-WIM** option to boot a device from a WIM image. It is located in %WPKCONTENTROOT%\Tools\bin\i386. When you enter the **FFUTool -WIM** command, you should see output that is similar to the following.

```
C:\> ffutool -wim MMOSWim.wim
Found device:
Name: Contoso.MSM8960.JD301_ATT.3.2.1
ID: 00000011-f151-a509-0000-000000000000
Booting WIM: MMOSWim.wim
WIM transfer completed in 26.550073 seconds.
```

The ffutool sends a WIM opcode to the device, along with information about the size of the image. Next a RAM buffer is allocated that will hold image. The ffutool then transfers the WIM image to the device. Once it's fully transferred, the device boots into the WIM image in memory.

**Note** The current MMOS WIM images may not display the rotating disc graphic but MMOS is still functional.

# Creating a secure MMOS WIM image

1/25/2019 • 4 minutes to read

You can use the SecWimTool to create a secure WIM image that can be used to service retail devices.

Before you can create a secure WIM, you must first create the WIM image itself. For info about how to create a WIM image, see [Working with WIM MMOS images](#).

To use the MMOS WIM in retail, it will need to be signed using retail certificates. More info will be provided in a later release of this documentation about how to work with the retail images.

For the secure WIM to run on the device, the signing certificates must match the certificates in the Platform Key (PK) that is provisioned on the device.

## Installing the needed certificates on the phone

Before you run the tools, you must first provision the appropriate certificates on the device. To become familiar with SecWimTool, partners can use the secure boot *test* tool to provision test (non-retail) certificates on the device. Use the secure boot *retail* tool for retail device.

## Creating a secure MMOS WIM image

To create a secure wim image, complete the following steps.

1. Open an elevated developer prompt in the directory that contains the output from the ImgToWim image generation process. For info about how to create a WIM image, see [Working with WIM MMOS images](#).

**Note** The image generation executables are located in %WPDKCONTENTROOT%\Tools\bin\i386. You can use the `set` command to add that path to your environment.

2. The platform ID must be used to create the WIM for a specific platform. The platform ID is set in using a device platform XML file.

You can display the platform ID using the ffutool command with the **-list** option.

```
C:\> ffutool -list
Name: Contoso.DCD9X0X.BD301_ATT.3.2.1
ID: 00000011-f151-a509-0000-FF0000000000
```

Build the unsigned secure WIM targeted for a specific platform using the **-platform** command option as shown here. No output is returned if the command completes successfully.

```
C:\> SecWimTool -build MMOSwim.wim MMOS wim.secwim -platform Contoso.DCD9X0X.BD301_ATT.3.2.1
```

3. Extract the catalog from the secure WIM image. No output is returned if the command completes successfully.

```
C:\> SecWimTool -extractcat MMOSwim.secwim MMOSwim.cat
```

4. You can sign the WIM with test certificates to test the WIM process. For a retail secure WIM, the FFU must be signed by Microsoft.

**Important** Information about signing with the final retail certificates will be provided in a later release of the documentation.

To sign the catalog using the test image certificate, use this command.

```
C:\> sign /pk MMOSwim.cat
```

This command will generate output that is similar to the following.

```
signtool.exe sign /v /a /u 1.3.6.1.4.1.311.76.5.10 /r "Microsoft Testing Root Certificate Authority 2010" /fd SHA256 /s my /c "1.3.6.1.4.1.311.21.8.7587021.7518 74.11030412.6202749.3702260.207.4167089.4524209" "MMOSwim.cat"
The following certificate was selected:
    Issued to: User1
    Issued by: MSIT Test CodeSign CA 3
    Expires:   Sat Jun 21 15:05:01 2014
    SHA1 hash: F571C183F768638B424A59772A1AFB1168164AFC

Done Adding Additional Store
Successfully signed: MMOSwim.cat

Number of files successfully Signed: 1
Number of warnings: 0
Number of errors: 0
signed: "MMOSwim.cat"
Sign.Cmd RC=0
```

5. Replace the existing catalog with a signed catalog. No output is returned if the command completes successfully.

```
C:\> SecWimTool -replacecat flashwim.cat MMOSwim.secwim
```

6. Place the device in flashing mode.
7. Flash the WIM to the device to test it using the FFUTool.

```
C:\> ffutool -wim MMOSwim.secwim
```

The command will generate output that is similar to the following.

```
Found device:
Name: Contoso.MSM8X0X.BD301_ATT.3.2.1
ID: 00000011-f151-a509-0000-FF0000000000
Booting WIM: MMOSwim.secwim
WIM transfer completed in 26.550073 seconds.
```

## Working with serial numbers

The serial number option can be used when including tools that should only be used for a specific device.

**Serial Number-** The serial number is a unique ID for the phone that is generated automatically. You can display the serial number using the ffutool command with the *-serial* option.

To build a secure WIM targeted for a specific device, use the `-serial` command option as shown here.

# Troubleshooting

When you flash the image if the PK doesn't match, you will receive this message.

An FFU error occurred: Device returned WIM boot failure status code 0x80000004.

# SecWimTool command line reference

The following summarizes the command syntax and the four command-line options for SecWimTool.

**SecWimTool** <command> <arguments>

**-build** - Packs the specified WIM into a secwim that is suitable for signing and transferring to a retail device.

- *Platform* - Includes targeting information for a specific platform type that the WIM is intended for. This option must be used to target the WIM for a specific platform.
  - *Serial* - Includes the serial number of a specific device the WIM will be used on. This option can be used when including tools that should only be used for a specific device.

```
SecWimTool -build <WIM> <output file> [-platform <ID>] [-serial <serial number>]
```

**-extractcat** - Retrieves the catalog from the secure WIM file, and writes it either to the output file (if specified) or stdout.

```
secwim -extractcat <path to .secwim> [<output file>]
```

**-replacetcat** - Replaces the catalog in the specified .secwim with the contents of the new catalog.

```
secwim -replacecat <path to catalog> <path to .secwim>
```

**-?** - Displays command line help. Use `secwimtool -<command> -?` for command level help.

## Related topics

## Working with WIM MMOS images

# Develop MMOS test applications

1/25/2019 • 2 minutes to read

Developing user-mode tests in MMOS is very similar to developing user-mode applications.

## Test development in MMOS

Use the following steps to create, deploy, and test a MMOS test application.

1. Create an app.
2. Add code to test the desired component and display or send those results as desired. For example, this sample code can be used to display a console message in MMOS.

```
#include <stdio.h>
#include <Windows.h>

int main()
{
    int i = 0;
    for(;;)
    {
        wprintf(L"Test");
        Sleep(500);
        if (i == 12)
        {
            wprintf(L"Done");
            break;
        }
        i++;
    }
    return 0;
}
```

3. Build the test application in Visual Studio.
4. Locate the binaries generated when you built the app. Typically they are in a subdirectory of the projects folder, such as `\MyProject\arm\debug\`.
5. Sign the executable by using the `sign.cmd` script in the `%WPDKCONTENTROOT%\Tools\bin\i386` folder, as shown in the example below.

```
sign.cmd ApplicationForDrivers.exe
```

6. To test your app, copy it to the device in the `C:\Data\test` directory by using FTP and run it via Telnet. For more info, see [Deploy and test a user-mode test application in MMOS](#).

## Libraries in MMOS

Adding a lib in MMOS is similar to adding a lib in the production OS. Currently, this default lib location for the kit is configured in Visual Studio.

```
$(WPKInstallDir)lib\$(KitOs)\wp_um\$(Platform)  
$(WPKInstallDir)Tools\WPE\CRT\lib\$(Platform)
```

To add a lib in Visual Studio, select **Project > Properties > Configuration Properties > VC++ Directories > Include Directories**. To use the MMOS libraries, append the following directory to the path.

```
$(WPDKContentRoot)include\mmos
```

Add the arm directory. In Visual Studio, select **Project > Properties > Configuration Properties > Linker > General > Additional Library Directories**. To use the MMOS libraries, append the following directories to the path.

```
$(WPDKContentRoot)lib\win8\mmos\arm
```

## Security in MMOS

In development environments, user-mode test binaries can be test signed (not production signed). Use the process described in [Sign binaries and packages](#) to test sign the binaries.

If the test binaries are not signed and code integrity checking is active as it normally is, you will receive an error message similar to the following in the debug window.

```
* This is not a failure in CI, but a problem with the failing binary.  
* Please contact the binary owner for getting the binary correctly signed.
```

# Deploy and test a user-mode test application in MMOS

1/25/2019 • 3 minutes to read

To copy files to an MMOS image and run programs, you can use FTP and Telnet over a Virtual Ethernet connection.

**Important** The USB drivers and protocols described here must not be used in manufacturing or retail servicing. They are provided as a convenience for engineering bring-up.

## Preparing the device

Engineering builds of the OS typically include Ping, FTP, and Telnet servers. To confirm that these are active in the OS, examine the startup configuration or establish a debug connection to MMOS to view the files that are loaded.

To configure a device to support TCP/IP over USB, you must install the necessary tools and use the BCDEdit tool to modify the device's boot options that are stored in the BCD file.

## Running Virtual Ethernet and determining the IP address of the device

Virtual Ethernet creates a connection between the device's USB connection and the TCP/IP transport on the PC. connecting the device or powering it on, start Virtual Ethernet on the PC. VirtEth.exe is located in the "%ProgramFiles(x86)%\Microsoft Windows Phone 8 KDBG Connectivity\bin" folder.

A Virtual Ethernet console window will open. Leave this window open to maintain the connection to the phone; closing the window closes the connection.

- With Virtual Ethernet running, connect and power on the phone. When the device is connected, you will see lines of output that include the MAC address of the device. Virtual Ethernet reports the device's MAC address in output similar to the following:

```
NIC UP: 00-11-38-EA-88-7E : SUCCESS
```

Note the 12-digit device MAC address from the VirtEth console window and use it in the next step to determine the device's TCP/IP address.

- Open a new command-prompt window and type **arp -a**. Output similar to the following will be displayed.

```
C:\>arp -a

Interface: 10.178.1.40 --- 0xb
  Internet Address      Physical Address      Type
    10.178.0.1            cc-ef-48-a7-6f-3f    dynamic
    10.178.1.94           00-11-38-ea-88-7e    dynamic
    10.178.3.255          ff-ff-ff-ff-ff-ff    static
    ...
```

Use the IP address that is associated with the device MAC address to connect to the device via FTP and Telnet.

## Establishing an FTP connection

To browse and copy files via FTP:

- Open Windows Explorer and type: **FTP:\W.X.Y.Z** in the address bar, replacing *W.X.Y.Z* with the IP address of the device.

You should see the files on the device listed. Use Windows Explorer to copy files to or from the device, such as executable test programs or logs of test results.

To close the FTP connection, unmount the phone file system by selecting **Safely Remove Hardware** in the Windows notification area.

## Establishing a Telnet session

Make sure Telnet is enabled on the PC. To enable Telnet in the Windows operating system, select **Control Panel > Programs and Features > Turn Windows features on or off**. In the Windows Features list, select **Telnet Client**, and then click **OK**.

To establish a Telnet session with the device:

- Open a command-prompt window and type: **telnet W.X.Y.Z**, replacing *W.X.Y.Z* with the IP address of the device

The command prompt window that appears is cmd.exe running on the device. From that command prompt, you can run commands on the device, such as executing test applications that you included in the MMOS image.

## Troubleshooting: Confirming TCP/IP connectivity

You can use **ping** to test the TCP/IP connection with the device:

- Open a command-prompt window and type: **ping W.X.Y.Z**, replacing *W.X.Y.Z* with the IP address of the device

The command should indicate that the packets were returned. If this does not work, one common issue to investigate is the firewall configuration on the PC.

## Debugging in MMOS

To enable debugger support in MMOS, both communication and OS settings must be modified.

### Enabling debug support

To enable debugging support in MMOS, specify the following internal optional feature in the MfgOEMInput.xml file.

```
<Microsoft>
  ...
  <Feature>KDNETUSB_ON</Feature>
  ...
</Microsoft>
```

This adds the required packages to the MMOS image. For more info about working with the MfgOEMInput.xml file, see [MMOS image definition](#).

### Enabling communication settings

To enable the OS, communication settings must be changed after the image is flashed.

### Establishing the debug connection

To connect to MMOS for debugging, use WinDbg to specify the key and port that you configured earlier by using BCDEdit.

```
windbg.exe -k net:Port=50000,Key=1.2.3.4
```

# Determine if MMOS is running

1/25/2019 • 2 minutes to read

You can check to see if MMOS is running the same way that you check when running in Manufacturing Mode. For more info, see [Detect Manufacturing Mode](#).

**Note** You should not query the ManufacturingOS registry setting as this key is obsolete.

# Manufacturing test environment supported APIs

1/25/2019 • 2 minutes to read

The manufacturing test environment (MTE) supports APIs and techniques that are designed for use only in manufacturing. This section describes the specific libraries that are supported in MTE.

## MMOS.Lib

MMOS.Lib provides a mirror interface to the Windows 8 mincore.lib. For general information about mincore.lib, see [Windows API sets](#).

The primary user-mode native APIs for use in MTE are defined in the MMOS library (.lib) files installed by the Windows Driver Kit (WDK) as %WDFCONTENTROOT%\Lib\ folder.

## Supported APIs

The following sections provide a preview of the features that the provided MMOS APIs allow for testing. Additional information about the MMOS supported APIs will be provided in a later release of this documentation. The API header sub-folders and Windows SDK headers are located in the "%ProgramFiles(x86)%\Windows Kits\10\include\" folder. These APIs can be used in user-mode applications for MMOS. For more info on creating and running MMOS applications, see the [Develop MMOS test applications](#) topic.

### **Micophone, audio and audio routing**

The audio and audio routing APIs allow test apps to test the audio and audio routing capabilities. Some scenarios that test apps can test are playing frequencies on each of the different audio end-points. The microphone can be tested using the audio APIs. The following header files contain the audio and audio routing APIs.

- Mmos\audiotunerapi.h
- Mmos\audiotunerdef.h
- Mmos\audiotunerprop.h
- Km\ksmedia\_phone.h
- Um\initguid.h
- Um\avrt.h (Windows SDK)
- ctime.h (Windows SDK)
- Um\audioclient.h (Windows SDK)
- Um\mmdeviceapi.h (Windows SDK)
- Um\functiondiscoverykeys.h (Windows SDK)

### **Display**

You can use the Direct3D to display information. For general info, see this MSDN link [Direct3D 11 Graphics](#). The following header files contain the D3D APIs.

- Um\D3DWrapper.h
- Um\D3D11.h (Windows SDK)

## **Camera**

You can use the Media Foundation CaptureEngine APIs to work with the camera. For general info about working with Media Foundation, see [Media Foundation Programming Reference](#) on MSDN. For information about the IMFCaptureEngine interface, see this MSDN topic, [IMFCaptureEngine interface](#). The following header files contain interfaces that may be useful for camera testing.

- Um\mfapi.h
- Um\mfobjects.h
- Um\mfidl.h
- Um\mfreadwrite.h
- Um\mfcaptureengine.h
- Um\wincodec.h

## **Sensors**

For more information about sensors and APIs that are used for testing in MMOS, see the Sensors topic.

## **LED**

The LED APIs allow test apps to test the notification LED by calling different IOCTLs to cause the notification LED to turn on, turn off, or blink. The following header file contains the LED APIs.

- Mmos\hwn.h

## **SD Card**

The SD card APIs allow test apps to test the SD card driver by calling different IOCTLs to test cases such as reading and writing to the card. The following header file contains the SD card APIs.

- Um\sffdisk.h

## **Touch**

For more info about testing the Touch controller, see the [Access the touch interface in MMOS](#) topic. The following header files contain the touch APIs.

- Um\InputDriverRawSamples.h
- Um\WinPhoneInput.h

## **Vibrator**

The vibrator APIs allow test apps to test the vibrator on the device by calling different IOCTLs. The IOCTLs allow the apps to test various speeds and periods of the vibrator driver on the device. The following header file contains the vibrator APIs.

- Mmos\hwn.h

## **Hardware buttons**

For more information about hardware buttons, see the Hardware buttons topic. The following header file contains the hardware button APIs.

- UM\ntddkbd.h

## **FM radio**

The FM radio APIs allow test apps to test the FM radio tuner on the phone by calling FM IOCTLs. The IOCTLs allow apps to test various scenarios such as tuning to a frequency or seeking. The following header files contain the FM radio APIs.

- Mmos\audiotunerapi.h
- Mmos\audiotunerprop.h

## Wi-Fi

For more information about Wi-Fi testing APIs, see the [Wi-Fi manufacturing API](#) topic. The following header file contains the Wi-Fi APIs.

- Um\wifimte.h

## Related topics

[Calling SetScreenOff to enter connected standby](#)

# Manufacturing Mode Phone Call Testing APIs

1/25/2019 • 2 minutes to read

These APIs are used by phone manufacturers to test phone call functionality while the device is booted into Manufacturing Mode.

## In this section

TOPIC	DESCRIPTION
<a href="#">MfgPhoneDial</a>	Causes the phone to dial a call.
<a href="#">MfgPhoneEndCall</a>	Ends a phone call.
<a href="#">MfgPhoneGetSimLineCount</a>	Gets the number of currently detected SIM slots.
<a href="#">MfgPhoneGetSimLineDetail</a>	Retrieves a struct that contains the current details for a given SIM-based phone line.
<a href="#">MfgPhoneGetSpeaker</a>	Returns a boolean indicating whether the phone speaker is being used, as opposed to the handset earphone.
<a href="#">MfgPhoneInitialize</a>	Initializes the phone system and the internal state of the API implemented by DLL.
<a href="#">MfgPhoneSetSimLineEventNotifyCallback</a>	Callback-based notification mechanism for receiving events on SIM-based phone lines.
<a href="#">MfgPhoneSetSpeaker</a>	Sets a value indicating whether the phone speaker should be used, as opposed to the handset earphone.
<a href="#">MfgPhoneUninitialize</a>	Uninitializes the phone system and the internal state of the API implemented by DLL.
<a href="#">MFGPHONE_CALLSTATUS</a>	Provides information about the status of the call.
<a href="#">MFGPHONE_LINESYSTEMTYPE</a>	Provides information about the type of line system.
<a href="#">MFGPHONE_REGISTRATIONSTATE</a>	Provides information about the state of the phone line? call?

TOPIC	DESCRIPTION
<a href="#"><b>MFGPHONE_SIMLINEDETAIL</b></a>	Provides information about a particular SIM-based phone line.
<a href="#"><b>MFGPHONE_SIMSTATE</b></a>	Provides information about the state of the SIM.

# MfgPhoneDial function

1/25/2019 • 2 minutes to read

Causes the phone to dial a call.

**MfgPhoneDial** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
HRESULT APIENTRY MfgPhoneDial(
    _In_  UINT      SimSlot,
    _In_  PCWSTR   DialNumber
);
```

## Parameters

### *SimSlot* [in]

The SIM-based phone line to use.

### *DialNumber* [in]

The phone number to dial.

## Return value

S\_OK is returned upon success and an error code is returned otherwise.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
DLL	MFGPHONE.DLL

# MfgPhoneEndCall function

1/25/2019 • 2 minutes to read

Ends a phone call.

**MfgPhoneEndCall** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
HRESULT APIENTRY MfgPhoneEndCall(  
    _In_ UINT SimSlot  
) ;
```

## Parameters

*SimSlot* [in]

The SIM-based phone line whose call should be ended.

## Return value

S\_OK is returned upon success and an error code is returned otherwise.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
DLL	MFGPHONE.DLL

# MfgPhoneGetSimLineCount function

1/25/2019 • 2 minutes to read

Gets the number of currently detected SIM slots.

**MfgPhoneGetSimLineCount** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
HRESULT APIENTRY MfgPhoneGetSimLineCount(
    _Out_ PUINT SimLineCount
);
```

## Parameters

*SimLineCount* [out]

Pointer to a **UINT** that specifies the number of currently detected SIM slots. Both active and inactive SIM slots are included in the count.

## Return value

S\_OK is returned upon success and an error code is returned otherwise.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
DLL	MFGPHONE.DLL

# MfgPhoneGetSimLineDetail function

1/25/2019 • 2 minutes to read

Retrieves a struct that contains the current details for a given SIM-based phone line.

**MfgPhoneGetSimLineDetail** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
HRESULT APIENTRY MfgPhoneGetSimLineDetail(
    _In_     UINT             SimSlot,
    _Out_    PMFGPHONE_SIMLINEDETAIL SimLineDetail,
    _In_     ULONG            SimLineDetailSize,
    _Out_    PULONG           RequiredSize
);
```

## Parameters

*SimSlot* [in]

Specifies the SIM-based phone line.

*SimLineDetail* [out]

Pointer to a **MFGPHONE\_SIMLINEDETAIL** struct that contains the current details for the SIM-based phone line specified by *SimSlot*.

*SimLineDetailSize* [in]

Specifies the size of the **SimLineDetail** parameter.

*RequiredSize* [out]

Specifies the required size for the **SimLineDetail** parameter.

## Return value

S\_OK is returned upon success and an error code is returned otherwise.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
DLL	MFGPHONE.DLL

# MfgPhoneGetSpeaker function

1/25/2019 • 2 minutes to read

Returns a boolean indicating whether the phone speaker is being used, as opposed to the handset earphone.

**MfgPhoneGetSpeaker** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
HRESULT APIENTRY MfgPhoneGetSpeaker(  
    _Out_ PBOOL pbSpeakerOn  
) ;
```

## Parameters

*pbSpeakerOn* [out]

TRUE if the speaker is being used, otherwise FALSE.

## Return value

S\_OK is returned upon success and an error code is returned otherwise.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
DLL	MFGPHONE.DLL

# MfgPhoneInitialize function

1/25/2019 • 2 minutes to read

Initializes the phone system and the internal state of the API implemented by DLL.

**MfgPhoneInitialize** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
HRESULT APIENTRY MfgPhoneInitialize(void);
```

## Parameters

This function has no parameters.

## Return value

S\_OK is returned upon success and an error code is returned otherwise.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
DLL	MFGPHONE.DLL

# MfgPhoneSetSimLineEventNotifyCallback function

1/25/2019 • 2 minutes to read

Callback-based notification mechanism for receiving events on SIM-based phone lines.

**MfgPhoneSetSimLineEventNotifyCallback** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
HRESULT APIENTRY MfgPhoneSetSimLineEventNotifyCallback(
    _In_ MFGPHONE_SIMLINEEVENT_NOTIFY_CALLBACK  Callback,
    _In_ PVOID                               Context
);
```

## Parameters

*Callback* [in]

The callback function to call when the event occurs.

*Context* [in]

The context.

## Return value

S\_OK is returned upon success and an error code is returned otherwise.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
DLL	MFGPHONE.DLL

# MfgPhoneSetSpeaker function

1/25/2019 • 2 minutes to read

Sets a value indicating whether the phone speaker should be used, as opposed to the handset earphone.

**MfgPhoneSetSpeaker** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
HRESULT APIENTRY MfgPhoneSetSpeaker(
    _In_ BOOL bSpeakerOn
);
```

## Parameters

*bSpeakerOn* [in]

TRUE if the speaker should be used, otherwise false.

## Return value

S\_OK is returned upon success and an error code is returned otherwise.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
DLL	MFGPHONE.DLL

# MfgPhoneUninitialize function

1/25/2019 • 2 minutes to read

Uninitializes the phone system and the internal state of the API implemented by DLL.

**MfgPhoneUninitialize** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
HRESULT APIENTRY MfgPhoneUninitialize(void);
```

## Parameters

This function has no parameters.

## Return value

S\_OK is returned upon success and an error code is returned otherwise.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
DLL	MFGPHONE.DLL

# MFGPHONE\_CALLSTATUS enumeration

1/25/2019 • 2 minutes to read

Provides information about the status of the call.

**MFGPHONE\_CALLSTATUS** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
typedef enum _MFGPHONE_CALLSTATUS {  
    MFGPHONE_CALLSTATUS_UNKNOWN    = 0,  
    MFGPHONE_CALLSTATUS_IDLE      = 1,  
    MFGPHONE_CALLSTATUS_CALLING   = 2,  
    MFGPHONE_CALLSTATUS_INCOMING   = 3,  
    MFGPHONE_CALLSTATUS_ACTIVE     = 4  
} MFGPHONE_CALLSTATUS;
```

## Constants

### **MFGPHONE\_CALLSTATUS\_UNKNOWN**

The call status is unknown.

### **MFGPHONE\_CALLSTATUS\_IDLE**

The call status is idle.

### **MFGPHONE\_CALLSTATUS\_CALLING**

The call status is calling.

### **MFGPHONE\_CALLSTATUS\_INCOMING**

The call status is incoming.

### **MFGPHONE\_CALLSTATUS\_ACTIVE**

The call status is active.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
--------	---------------------------------

# MFGPHONE\_LINESYSTEMTYPE enumeration

1/25/2019 • 2 minutes to read

Provides information about the type of line system.

**MFGPHONE\_LINESYSTEMTYPE** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
typedef enum _MFGPHONE_LINESYSTEMTYPE {  
    MFGPHONE_LINESYSTEMTYPE_UNKNOWN    = 0,  
    MFGPHONE_LINESYSTEMTYPE_GSM        = 1,  
    MFGPHONE_LINESYSTEMTYPE_CDMA       = 2,  
    MFGPHONE_LINESYSTEMTYPE_IMS        = 3  
} MFGPHONE_LINESYSTEMTYPE;
```

## Constants

### **MFGPHONE\_LINESYSTEMTYPE\_UNKNOWN**

The line system type is unknown.

### **MFGPHONE\_LINESYSTEMTYPE\_GSM**

The type of line system is GSM.

### **MFGPHONE\_LINESYSTEMTYPE\_CDMA**

The type of line system is CDMA.

### **MFGPHONE\_LINESYSTEMTYPE\_IMS**

The type of line system is IMS.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
--------	---------------------------------

# MFGPHONE\_REGISTRATIONSTATE enumeration

1/25/2019 • 2 minutes to read

Provides information about the state of the phone line? call?

**MFGPHONE\_REGISTRATIONSTATE** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
typedef enum _MFGPHONE_REGISTRATIONSTATE {  
    MFGPHONE_REGISTRATIONSTATE_UNKNOWN      = 0,  
    MFGPHONE_REGISTRATIONSTATE_NOSIGNAL     = 1,  
    MFGPHONE_REGISTRATIONSTATE_UNREGISTERED = 2,  
    MFGPHONE_REGISTRATIONSTATE_REGISTERING  = 3,  
    MFGPHONE_REGISTRATIONSTATE_REGISTERED   = 4,  
    MFGPHONE_REGISTRATIONSTATE_DENIED       = 5  
} MFGPHONE_REGISTRATIONSTATE;
```

## Constants

### **MFGPHONE\_REGISTRATIONSTATE\_UNKNOWN**

The registration state is not known.

### **MFGPHONE\_REGISTRATIONSTATE\_NOSIGNAL**

There is no signal to detect the registration state.

### **MFGPHONE\_REGISTRATIONSTATE\_UNREGISTERED**

The registration state is unregistered.

### **MFGPHONE\_REGISTRATIONSTATE\_REGISTERING**

The registration state is registering.

### **MFGPHONE\_REGISTRATIONSTATE\_REGISTERED**

The registration state is registered.

### **MFGPHONE\_REGISTRATIONSTATE\_DENIED**

The registration state is denied.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
--------	---------------------------------

# MFGPHONE\_SIMLINEDETAIL structure

1/25/2019 • 2 minutes to read

Provides information about a particular SIM-based phone line. This **struct** is retrieved via the [MfgPhoneGetSimLineDetail](#) function.

**MFGPHONE\_SIMLINEDETAIL** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
typedef struct _MFGPHONE_SIMLINEDETAIL {  
    UINT             SimSlot;  
    MFGPHONE_SIMSTATE   SimState;  
    MFGPHONE_REGISTRATIONSTATE RegistrationState;  
    WCHAR [64]          NetworkName;  
    MFGPHONE_LINESYSTEMTYPE LineSystemType;  
    UINT              SignalStrength;  
    MFGPHONE_CALLSTATUS   CallStatus;  
} MFGPHONE_SIMLINEDETAIL, *PMFGPHONE_SIMLINEDETAIL;
```

## Members

### SimSlot

The SIM-based phone line to which the details in this struct pertain.

### SimState

An **enum** specifying the current state of the SIM-based phone line.

### RegistrationState

An **enum** specifying the current registration state of the phone line.

### NetworkName

WCHAR containing the name of the network.

### LineSystemType

An **enum** specifying the line system type of the phone line.

### SignalStrength

Unsigned Integer containing the signal strength of the phone line.

### CallStatus

An **enum** specifying the call status of the phone line.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
--------	---------------------------------

# MFGPHONE\_SIMSTATE enumeration

1/25/2019 • 2 minutes to read

Provides information about the state of the SIM.

**MFGPHONE\_SIMSTATE** is for phone manufacturers and can only be called in Manufacturing Mode.

## Syntax

```
typedef enum _MFGPHONE_SIMSTATE {  
    MFGPHONE_SIMSTATE_UNKNOWN = 0,  
    MFGPHONE_SIMSTATE_NONE = 1,  
    MFGPHONE_SIMSTATE_ACTIVE = 2,  
    MFGPHONE_SIMSTATE_INVALID = 3,  
    MFGPHONE_SIMSTATE_LOCKED = 4,  
    MFGPHONE_SIMSTATE_DISABLED = 5  
} MFGPHONE_SIMSTATE;
```

## Constants

### **MFGPHONE\_SIMSTATE\_UNKNOWN**

The SIM state is unknown.

### **MFGPHONE\_SIMSTATE\_NONE**

The SIM state is none. There is no SIM.

### **MFGPHONE\_SIMSTATE\_ACTIVE**

The SIM state is active.

### **MFGPHONE\_SIMSTATE\_INVALID**

The SIM state is invalid.

### **MFGPHONE\_SIMSTATE\_LOCKED**

The SIM state is locked.

### **MFGPHONE\_SIMSTATE\_DISABLED**

The SIM state is disabled.

## Requirements

Header	Mfgphone.h (include Mfgphone.h)
--------	---------------------------------

# Access the touch interface in MMOS

1/25/2019 • 3 minutes to read

The touch controller output can be captured and used in MMOS with proper touch driver support. The touch interface sample illustrates how you can gather touch coordinates in MMOS to implement manufacturing tests.

## Touch sample requirements

The sample code requires that the Microsoft.Input.TchHID.spkg package to be included in the image (either the Main OS or MMOS). The Microsoft.Input.TchHID.spkg package includes tchhid.sys, which implements the **CreateFile** and **Readfile** functions used by the sample code.

## Touch sample code

This sample user-mode application displays the touch coordinates received by the touch screen driver.

The sample app uses the **CreateFile** function to open an exclusive handle to the device to receive raw touch samples. The device name "TouchRaw0" is exposed by tchhid.sys, the HID touch class driver provided by Microsoft and included in Microsoft.Input.TchHID.spkg.

```
#define TOUCH_RAW_SAMPLES_DEVICE_NAME L"\\\\.\\\\TouchRaw0"
```

After the handle is successfully opened, the application loops until a touch contact point is received near the top-left corner of the screen. The application reads and displays the coordinates and the associated touch contact ID.

```
#include "sample.h"
#include <cfgmgr32.h>
#include <strsafe.h>

int main()
{
    HANDLE testDriver = INVALID_HANDLE_VALUE;
    BOOL exit = FALSE;
    INT32 i;
    TouchInfo touchInfo = {0};

    //
    // Open an exclusive handle to the device to get raw samples
    //
    testDriver = CreateFile(
        L"\\\\.\\\\TouchRaw0",
        GENERIC_READ,
        0,
        NULL,
        OPEN_EXISTING,
        0,
        NULL);

    if (INVALID_HANDLE_VALUE == testDriver)
    {
        goto exit;
    }

    //
    // Loop, printing touches to the debugger.
    //
```

```

// Release in upper-left corner ends the test.
//
while (!exit)
{
    DWORD touchInfoBytesRead = 0;

    //
    // Wait for data
    //
    if (!ReadFile(
        testDriver,
        &touchInfo,
        sizeof(TouchInfo),
        &touchInfoBytesRead,
        NULL))
    {
        goto exit;
    }

    if (touchInfoBytesRead == 0)
    {
        goto exit;
    }

    //
    // Print to debugger
    //
    for (i=0; i<touchInfo.ContactCount; i++)
    {
        WCHAR infoString[MAX_PATH] = {0};

        StringCchPrintf(
            infoString,
            MAX_PATH,
            L"%d: Contact ID %d, at (%d,%d) is %s\r\n",
            GetTickCount(),
            touchInfo.ContactArray[i].ContactID,
            touchInfo.ContactArray[i].ScreenX,
            touchInfo.ContactArray[i].ScreenY,
            FLAGS_TO_STRING(touchInfo.ContactArray[i].Flags));

        OutputDebugString(infoString);
    }

    //
    // Look for program exit
    //
    for (i=0; i<touchInfo.ContactCount; i++)
    {
        if ((touchInfo.ContactArray[i].ScreenX < 50) &&
            (touchInfo.ContactArray[i].ScreenY < 50) &&
            (touchInfo.ContactArray[i].Flags & InputEventFlag_Up))
        {
            OutputDebugString(L"Touch below (50,50), quitting!\r\n");
            exit = TRUE;
        }
    }
}

exit:
if (testDriver != INVALID_HANDLE_VALUE)
{
    CloseHandle(testDriver);
}

return 0;
}

```

The application uses the TouchInfo and TouchContact data structures, which are defined in %WPDKCONTENTROOT%\include\um\WinPhoneInput.h. The header code is shown here.

```
#pragma once

#include <windows.h>
#include <initguid.h>
#include <devguid.h>

// 
// This device name is used to access raw touch samples from user mode.
//

#define TOUCH_RAW_SAMPLES_DEVICE_NAME L"\\.\\"TouchRaw0"

enum InputEventFlag
{
    InputEventFlag_None      = 0x0000,
    InputEventFlag_Down      = 0x0001,
    InputEventFlag_Move      = 0x0002,
    InputEventFlag_Hold      = 0x0002,
    InputEventFlag_Up        = 0x0004,
    InputEventFlag_Empty     = 0x1000,
    InputEventFlag_Invalid   = 0x2000,
    InputEventFlag_TestSync  = 0x8000
};

typedef struct _TouchContact
{
    UINT16          ContactID;
    UINT16          Flags;           // See InputEventFlag_*
    INT16           ScreenX;        // Screen Space X-Position
    INT16           ScreenY;        // Screen Space Y-Position
    INT16           WindowX;        // Ignore
    INT16           WindowY;        // Ignore
} TouchContact;

typedef struct _TouchInfo
{
    UINT16          Size;           // Size, in bytes, of this structure (includes n contacts)
    UINT16          Flags;           // Ignore
    UINT32          TimeStamp;       // Driver timestamp
    HANDLE          Source;          // Ignore
    UINT32          SessionID;       // Ignore
    INT32           ContactCount;    // Count of touch contact data points
    TouchContact    ContactArray[10]; // Collection of contacts
} TouchInfo;

#define FLAGS_TO_STRING(x) \
    (x & InputEventFlag_Down) ? L"Down" : \
    (x & InputEventFlag_Move) ? L"Move" : \
    (x & InputEventFlag_Up) ? L"Up" : \
    L"Unknown"
```

## Building and deploying the sample application

To build the user-mode test application, complete the following steps.

1. Create a new user-mode application project in Visual Studio. For more info, see [Develop MMOS test applications](#)
2. Add the sample code and the header file to the project.

3. Build the solution, ensuring that it is generated successfully.
4. Disable code integrity in MMOS, or sign the test executable. For more info, see "Security in MMOS," in [Develop MMOS test applications](#).
5. Deploy the executable to the device and run the application. For more information, see [Deploy and test a user-mode test application in MMOS](#).

# Calling SetScreenOff to enter connected standby

1/25/2019 • 2 minutes to read

Calling the **SetScreenOff** function turns the screen and backlight off, which causes the phone to enter the connected standby power state. This lower power state can be helpful for testing power usage.

**Important** This function is for use only in the Microsoft Manufacturing OS.

## Syntax

```
HRESULT SetScreenOff();
```

## Parameters

None

## Return Value

HRESULT

## Remarks

There is not an equivalent function to return the device to a full power state.

## Example

To use SetScreenOff, include the header and call without any parameters.

```
#include <ManufacturingConnectedStandbyControl.h>
SetScreenOff();
```

## Requirements

**Header:** ManufacturingConnectedStandbyControl.h

**Library:** ManufacturingConnectedStandbyControl.lib

# Resetting a device during manufacturing

1/25/2019 • 2 minutes to read

The topic provides information about resetting a device during the manufacturing process.

## ResetPhoneEx

You can reset the device using the ResetPhoneEx API while preserving the following data.

- Map data.
- Runtime configuration data.
- Preinstalled apps.

# Wi-Fi manufacturing API

1/25/2019 • 2 minutes to read

As part of the manufacturing process, you must run tests to ensure that the components are integrated, functioning, and calibrated properly, and that they meet all regulatory requirements.

The API members documented in this section are interfaces defined for IHVs to use to write tests applications for the Wi-Fi chipset. This API set requires that the Wi-Fi driver conform to the driver OID specification.

This test API must only be used in manufacturing mode. For more info, see [Determine if MMOS is running](#).

## In this section

The following interfaces are defined for this API.

### [WlanMTEEnumAdapters](#)

Returns the list of the adapters that are recognized by the Wi-Fi stack.

### [WlanMTEOpenHandle](#)

Opens a handle to the driver based on the interface GUID specified and returns the handle to the caller.

### [WlanMTECloseHandle](#)

Closes a handle to the driver previously opened by [WlanMTEOpenHandle](#).

### [WlanMTERegisterCallbackHandler](#)

Registers a handler that will be called whenever the driver invokes a callback for a manufacturing functionality event.

### [WlanMTEDeRegisterCallbackHandler](#)

Unregisters a callback handler so that it will no longer be called when a manufacturing-related functionality event occurs.

### [WlanMTETGetVendorInfo](#)

Requests vendor-specific information, such as the vendor ID and vendor description.

### [WlanMTEResetAdapter](#)

Resets the Wi-Fi adapter.

### [WlanMTEQueryMacAddress](#)

Queries the MAC address of the Wi-Fi adapter.

### [WlanMTEQueryPhyTypes](#)

Queries the list of 802.11 PHY types configured on the adapter.

### [WlanMTEStartSelfTest](#)

Starts a preconfigured set of self-tests.

### [WlanMTEQuerySelfTestResult](#)

Queries the driver for the results of a previously requested self-test.

### [WlanMTERxSignal](#)

Queries information about the received signal at a specific band and channel.

### [WlanMTETxSignal](#)

Requests the driver to transmit a signal at the specified band and channel.

### [WlanMTEQueryADC](#)

Requests the value of the transmitted signal when performed over an open loop.

### [WlanMTESetData](#)

Requests that the driver write data to a specified location and offset from that location.

### [WlanMTEQueryData](#)

Queries the driver for data at a specific location and offset from that location.

### [WlanMTESleep](#)

Requests that the driver go to sleep either for a specified time interval or indefinitely until an awake request is sent.

### [WlanMTEAwake](#)

Requests that the driver wake up from its current sleep state.

## Related topics

[Adding Wi-Fi manufacturing test support to the OID interface](#)

# WlanMTEEnumAdapters

1/25/2019 • 2 minutes to read

Returns the list of adapters that are recognized by the Wi-Fi stack.

## Syntax

```
DWORD WlanMTEEnumAdapters(  
    __out_opt    WLAN_MTE_ADAPTER_LIST  **ppWlanAdapterList  
)
```

## Parameters

*ppWlanAdapterList*

[out] A list of detected Wi-Fi adapters.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS. Otherwise, the function returns a system error code.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTEOpenHandle

1/25/2019 • 2 minutes to read

Opens a handle on the driver based on the interface GUID specified and returns the handle to the caller.

## Syntax

```
DWORD WlanMTEOpenHandle(
    __in     GUID      *pAdapterGuid,
    __out    HANDLE    *phAdapter
);
```

## Parameters

### *pAdapterGuid*

[in] A pointer to the GUID identifying the Wi-Fi adapter on which the handle is to be opened.

### *phAdapter*

[out] A pointer to the Wi-Fi handle, if it was opened successfully.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails one of the system error codes is returned. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned if the <i>pAdapterGuid</i> or <i>phAdapter</i> parameters are NULL.
ERROR_INVALID_STATE	Returned if the current DOT11 operation mode cannot be retrieved.

## Remarks

The list of Wi-Fi interface GUIDs can be obtained by calling [WlanMTEEnumAdapters](#).

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTECloseHandle

1/25/2019 • 2 minutes to read

Closes a handle to the driver previously opened by [WlanMTEOpenHandle](#).

## Syntax

```
DWORD WlanMTECloseHandle(  
    __in     HANDLE  hAdapter  
)
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned if the library has not been initialized, or if the adapter handle specified by <i>hAdapter</i> is invalid.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTERegisterCallbackHandler

1/25/2019 • 2 minutes to read

Registers a handler that will be called whenever the driver invokes a callback for a manufacturing functionality event.

## Syntax

```
DWORD WlanMTERegisterCallbackHandler(
    __in     HANDLE                 hAdapter,
    __in     WLAN_MTE_NOTIFICATION_CALLBACK Callback
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

### *Callback*

[in] The handler function being registered by the application for manufacturing callbacks.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned if the adapter handle specified by the <i>hAdapter</i> parameter is invalid or NULL.

## Remarks

The callback function has the following prototype:

```
typedef VOID (WINAPI *WLAN_MTE_NOTIFICATION_CALLBACK)(
    __in     PDOT11_MANUFACTURING_CALLBACK_PARAMETERS     pMTECallback,
    __in     PVOID                                         pvReserved
);
```

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTEDeRegisterCallbackHandler

1/25/2019 • 2 minutes to read

Unregisters a callback handler so that it will no longer be called when a manufacturing-related functionality event occurs.

## Syntax

```
DWORD WlanMTEDeRegisterCallbackHandler(
    __in     HANDLE             hAdapter
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

## Remarks

The calling application must have previously registered a callback by calling [WlanMTERegisterCallbackHandler](#).

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned if the adapter handle specified by the <i>hAdapter</i> parameter is invalid or NULL.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTEGetVendorInfo

1/25/2019 • 2 minutes to read

Requests vendor-specific information, such as the vendor ID and vendor description.

## Syntax

```
DWORD WlanMTEGetVendorInfo(
    __in                  HANDLE  hAdapter,
    __out                 ULONG   *puVendorId,
    __in                  ULONG   uOutBufLen,
    __out_bcount(uOutBufLen) PUCHAR pucOutBuffer
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

### *puVendorId*

[out] The vendor ID.

### *uOutBufLen*

[in] The length of the buffer for retrieving the vendor description.

### *pucOutBuffer*

[out] The buffer that will contain the vendor description string.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned if the <i>puVendorID</i> , <i>uOutBufLen</i> , or <i>pucOutBuffer</i> parameter is NULL.
ERROR_INVALID_HANDLE	Returned if the adapter handle specified by the <i>hAdapter</i> parameter is invalid.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTEResetAdapter

1/25/2019 • 2 minutes to read

Resets the Wi-Fi adapter. The application can specify an optional callback and context handle to be invoked when the operation is complete.

## Syntax

```
DWORD WlanMTEResetAdapter(
    __in      HANDLE          hAdapter,
    __in      DOT11_RESET_REQUEST *pDot11ResetRequest,
    __in      WLAN_MTE_RESET_CALLBACK ResetCallback,
    __in      PVOID            pvContext
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

### *pDot11ResetRequest*

[in] Information about the reset request. If the application requires the reset in order to update the MAC address, it should specify either **dot11\_reset\_type\_mac** or **dot11\_reset\_type\_phy\_and\_mac** in order for the updated MAC address to be written to the DPP. Note that the MAC address change will only be valid when the device has booted in manufacturing mode.

### *ResetCallback*

[in, optional] The callback handler to be invoked on reset completion.

### *pvContext*

[in, optional] If the callback is specified, this context value is provided when the handler is called.

## Remarks

The callback function for Wi-Fi reset adapter notifications has the following prototype:

```
typedef VOID (WINAPI *WLAN_MTE_RESET_CALLBACK)(
    __in      DWORD      dwError,
    __in      PVOID      pvContext
);
```

## Return Value

If the function succeeds, the return value is **ERROR\_SUCCESS**.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned when the <i>pDot11ResetRequest</i> parameter is NULL or the <i>pDot11ResetRequest</i> type is invalid.
ERROR_INVALID_HANDLE	Returned if the <i>hAdapter</i> handle is invalid.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTEQueryMacAddress

1/25/2019 • 2 minutes to read

Returns the MAC address of the Wi-Fi adapter.

## Syntax

```
DWORD WlanMTEQueryMacAddress(
    __in     HANDLE          hAdapter,
    __out    DOT11_MAC_ADDRESS *pDot11MacAddress
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

### *pDot11MacAddress*

[out] The current MAC address of the Wi-Fi adapter.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned when the <i>pDot11MacAddress</i> parameter is NULL.
ERROR_INVALID_HANDLE	Returned if the <i>hAdapter</i> handle is invalid.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTEQueryPhyTypes

1/25/2019 • 2 minutes to read

Returns the list of 802.11 PHY types configured on the adapter.

## Syntax

```
DWORD WlanMTEQueryPhyTypes(
    __in     HANDLE          hAdapter,
    __out    PWLAN_MTE_PHY_LIST *ppPhyList
);
```

## Parameters

*hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

*ppPhyList*

[out] The list of available PHY types as described in [DOT11\\_PHY\\_TYPE](#) enumeration.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned if the <i>ppPhyList</i> parameter is NULL.
ERROR_INVALID_HANDLE	Returned if the <i>hAdapter</i> handle is invalid.
ERROR_OUTOFMEMORY	Returned when sufficient memory to perform the operation cannot be allocated.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTEStartSelfTest

1/25/2019 • 2 minutes to read

Starts a preconfigured set of self-tests.

## Syntax

```
DWORD WlanMTEStartSelfTest(
    __in                  HANDLE          hAdapter,
    __in                  DOT11_MANUFACTURING_SELF_TEST_TYPE eTestType,
    __in                  ULONG           uTestId,
    __in                  PVOID            pvContext,
    __in                  ULONG           uPinBitMask,
    __in                  ULONG           uInBufLen,
    __in_bcount_opt(uInBufLen) PUCHAR        pucInBuffer
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

### *eTestType*

[in] The type of self-test requested. The values of *eTestType* are defined by the DOT11\_MANUFACTURING\_SELF\_TEST\_TYPE enumeration, shown below:

```
typedef enum DOT11_MANUFACTURING_SELF_TEST_TYPE {
    DOT11_MANUFACTURING_SELF_TEST_TYPE_INTERFACE = 1,
    DOT11_MANUFACTURING_SELF_TEST_TYPE_RF_INTERFACE,
    DOT11_MANUFACTURING_SELF_TEST_TYPE_BT_COEXISTENCE
} DOT11_MANUFACTURING_SELF_TEST_TYPE, * PDOT11_MANUFACTURING_SELF_TEST_TYPE;
```

### *uTestId*

[in] The ID for the self-test requested.

### *pvContext*

[in] The context that uniquely identifies this request in the callback and in the subsequent results query.

### *uPinBitMask*

[in] The bit mask for adapter pins to be tested.

### *uInBufLen*

[in] The length of the buffer for passing in any additional information about the self-test.

### *pucInBuffer*

[in] The buffer that will contain additional information about the self-test.

## Remarks

On completion of the self-test, the application's callback handler is called, if one was registered, with the **dot11ManufacturingCallbackType** set to **dot11\_manufacturing\_callback\_self\_test\_complete**, and the result of the self-test is included.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned when the <i>uInBufLen</i> parameter is present but the <i>puInBuffer</i> parameter is NULL.
ERROR_INVALID_HANDLE	Returned if the adapter handle specified by the <i>hAdapter</i> parameter is invalid.
ERROR_OUTOFMEMORY	Returned when sufficient memory to perform the operation cannot be allocated.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTEQuerySelfTestResult

1/25/2019 • 2 minutes to read

Queries the driver for the results of a previously requested self-test.

## Syntax

```
DWORD WlanMTEQuerySelfTestResult(
    __in                      HANDLE          hAdapter,
    __in                      DOT11_MANUFACTURING_SELF_TEST_TYPE eTestType,
    __in                      ULONG           uTestId,
    __in                      PVOID            pvContext,
    __out                     BOOLEAN          *pbResult,
    __out                     ULONG           *puPinFailedBitMask,
    __out                     ULONG           *puBytesWrittenOut,
    __in                      ULONG           uOutBufLen,
    __out_bcount_opt(uOutBufLen) P UCHAR          pucOutBuffer
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

### *eTestType*

[in] The type of self-test requested. The values of *eTestType* are defined by the DOT11\_MANUFACTURING\_SELF\_TEST\_TYPE enumeration, shown below:

```
typedef enum DOT11_MANUFACTURING_SELF_TEST_TYPE {
    DOT11_MANUFACTURING_SELF_TEST_TYPE_INTERFACE = 1,
    DOT11_MANUFACTURING_SELF_TEST_TYPE_RF_INTERFACE,
    DOT11_MANUFACTURING_SELF_TEST_TYPE_BT_COEXISTENCE
} DOT11_MANUFACTURING_SELF_TEST_TYPE, * PDOT11_MANUFACTURING_SELF_TEST_TYPE;
```

### *uTestId*

[in] The ID for the self-test requested.

### *pvContext*

[in] The context that was specified in the original self-test request.

### *pbResult*

[out] The final result of the self-test. **True** if passed, **False** if failed.

### *puPinFailedBitMask*

[out] The bit mask for adapter pins that failed the test.

### *puBytesWrittenOut*

[out] The number of bytes of optional data returned from the self-test results.

### *uOutBufLen*

[in] The length of the buffer for returning any additional information about the self-test.

### *pucOutBuffer*

[out] The buffer of length *\*puBytesWrittenOut* that provides additional information about the self-test. The value of *\*puBytesWrittenOut* must be less than or equal to the value of *uOutBufLen*.

## Remarks

The application must have received a **dot11\_manufacturing\_callback\_self\_test\_complete** callback prior to calling this command. It should also provide the same context value that was used in the original self-test request in order to get the results for the appropriate self-test request.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned if the <i>pbResult</i> , <i>puPinFailedBitMask</i> , or <i>puBytesWrittenOut</i> parameter is NULL, or if the test type specified by the <i>eTestType</i> parameter is invalid.
ERROR_INVALID_HANDLE	Returned if the <i>hAdapter</i> handle is invalid.
ERROR_OUTOFMEMORY	Returned if sufficient memory to perform the operation could not be allocated.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTERxSignal

1/25/2019 • 2 minutes to read

Queries information about the received signal at a specific band and channel.

## Syntax

```
DWORD WlanMTERxSignal(
    __in     HANDLE      hAdapter,
    __out    BOOLEAN     *pbEnabled,
    __in     DOT11_BAND  Dot11Band,
    __in     ULONG       uChannel,
    __out    LONG        *pPowerLevel
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

### *pbEnabled*

[out] **True** if the driver detected a signal at the specified band and channel. **False** if no signal was detected.

### *Dot11Band*

[in] The band on which the signal is to be detected. The values of the *Dot11Band* parameter are defined by the DOT11\_BAND enum, shown below:

```
typedef enum DOT11_BAND {
    dot11_band_2p4g = 1,
    dot11_band_4p9g,
    dot11_band_5g
} DOT11_BAND, * PDOT11_BAND;
```

### *uChannel*

[in] The channel on which the signal is to be detected. The channel range depends on the band and on the supported PHY types.

### *pPowerLevel*

[out] The power level of the received signal detected at the antenna, returned as RSSI measured in dBm. This is valid only if *bEnabled* is **True**.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned if the <i>pbEnabled</i> , <i>Dot11Band</i> , <i>uChannel</i> , or <i>pPowerLevel</i> parameter is NULL.
ERROR_INVALID_HANDLE	Returned if the adapter handle specified by the <i>hAdapter</i> parameter is invalid.
ERROR_OUTOFMEMORY	Returned when sufficient memory to perform the operation cannot be allocated.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTETxSignal

1/25/2019 • 2 minutes to read

Requests the driver to transmit a signal at the specified band and channel.

## Syntax

```
DWORD WlanMTETxSignal(
    __in     HANDLE      hAdapter,
    __in     BOOLEAN     bEnable,
    __in     BOOLEAN     bOpenLoop,
    __in     DOT11_BAND  Dot11Band,
    __in     ULONG       uChannel,
    __in     LONG        SetPowerLevel,
    __out    LONG        *pADCPowerLevel
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

### *bEnable*

[in] If a value is set, the transmission is enabled. Otherwise, transmission at the specified band and channel is disabled.

### *bOpenLoop*

[in] When set to **True**, the driver must use an open loop power control and return the signal value in the *pADCPowerLevel* parameter. If this parameter is set and the hardware does not support open loop power control, an **ERROR\_NOT\_SUPPORTED** exception is returned.

### *Dot11Band*

[in] The band on which the signal is to be detected. The values of the *Dot11Band* parameter are defined by the DOT11\_BAND enum, shown below:

```
typedef enum DOT11_BAND {
    dot11_band_2p4g = 1,
    dot11_band_4p9g,
    dot11_band_5g
} DOT11_BAND, * PDOT11_BAND;
```

### *uChannel*

[in] The channel on which the signal is to be transmitted. The channel range depends on the band and supported PHY types.

### *SetPowerLevel*

[in] The power level of the transmitted signal, in dBm.

### *pADCPowerLevel*

[out, optional] The current signal level detected at the antenna, returned as a RAW value. The interpretation of this value is implemented by the IHV. This return parameter is valid if *bOpenLoop* is **True** and the hardware supports it.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists one of the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned when the <i>Dot11Band</i> or <i>uChannel</i> parameter is NULL, or if <i>bOpenLoop</i> is present but <i>pADCPowerLevel</i> is NULL.
ERROR_INVALID_HANDLE	Returned if the adapter handle specified by the <i>hAdapter</i> parameter is invalid.
ERROR_OUTOFMEMORY	Returned when sufficient memory to perform the operation cannot be allocated.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTEQueryADC

1/25/2019 • 2 minutes to read

Requests the value of the transmitted signal when performed over an open loop.

## Syntax

```
DWORD WlanMTEQueryADC(
    __in     HANDLE      hAdapter,
    __in     DOT11_BAND  Band,
    __in     ULONG       uChannel,
    __out    LONG        *pADCPowerLevel
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

### *Band*

[in] The band on which the signal is to be detected. The values of the *Dot11Band* parameter are defined by the DOT11\_BAND enum, shown below:

```
typedef enum DOT11_BAND {
    dot11_band_2p4g = 1,
    dot11_band_4p9g,
    dot11_band_5g
} DOT11_BAND, * PDOT11_BAND;
```

### *uChannel*

[in] The channel on which the signal is being transmitted. The channel range will depend on the band and supported PHY types.

### *pADCPowerLevel*

[out] The current signal level detected at the antenna returned as a RAW value. The interpretation of this value will be implemented by the IHV. This parameter is valid only if the device supports open loop power and is currently transmitting a signal on the open loop.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned if the <i>Dot11Band</i> , <i>uChannel</i> , or <i>pADCPowerLevel</i> parameter is NULL.

ERROR CODE	DESCRIPTION
ERROR_INVALID_HANDLE	Returned if the <i>hAdapter</i> handle is invalid.
ERROR_OUTOFMEMORY	Returned when sufficient memory to perform the operation cannot be allocated.

## Remarks

If open loop power is not supported, the driver will return **ERROR\_NOT\_SUPPORTED**.

## Return Value

If the function succeeds, the return value is **ERROR\_SUCCESS**.

If the function fails, the return value is one of the system error codes. The following table lists one of the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned when the <i>uInBufLen</i> parameter is present but the <i>puInBuffer</i> parameter is NULL.
ERROR_INVALID_HANDLE	The <i>hAdapter</i> handle is invalid.
ERROR_OUTOFMEMORY	There was insufficient memory to allocate to perform the function.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTESetData

1/25/2019 • 2 minutes to read

Requests that the driver write data to a specific location defined by a key and offset value.

## Syntax

```
DWORD WlanMTESetData(
    __in                  HANDLE  hAdapter,
    __in                  ULONG   uKey,
    __in                  ULONG   uOffset,
    __in                  ULONG   uInBufLen,
    __in_bcount(uInBufLen) P UCHAR pucInBuffer
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

### *uKey*

[in] The key for the write request.

### *uOffset*

[in] The offset for the write request.

### *uInBufLen*

[in] The length of the buffer containing the data to be written.

### *pucInBuffer*

[in] The buffer containing the data to be written.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned when the <i>uInBufLen</i> parameter is present but the <i>pucInBuffer</i> parameter is NULL.
ERROR_INVALID_HANDLE	Returned if the <i>hAdapter</i> handle is invalid.
ERROR_OUTOFMEMORY	Returned when sufficient memory to perform the operation cannot be allocated.

# Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTEQueryData

1/25/2019 • 2 minutes to read

Queries the driver for data stored at a specific location defined by a key and offset value.

## Syntax

```
DWORD WlanMTEQueryData(
    __in                  HANDLE  hAdapter,
    __in                  ULONG   uKey,
    __in                  ULONG   uOffset,
    __out                 ULONG   *puBytesWrittenOut,
    __in                  ULONG   uOutBufLen,
    __out_bcount(uOutBufLen) P UCHAR pucOutBuffer
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

### *uKey*

[in] The key for the query request.

### *uOffset*

[in] The offset for the query request.

### *puBytesWrittenOut*

[out] The number of bytes of data returned from the query request.

### *uOutBufLen*

[in] The length of the buffer for returning the information requested.

### *pucOutBuffer*

[out] The buffer that will contain the data returned per the query request.

## Return Value

If the function succeeds, the return value is ERROR\_SUCCESS.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
ERROR_INVALID_PARAMETER	Returned if the <i>puBytesWrittenOut</i> , <i>uOutBufLen</i> , or <i>pucOutBuffer</i> parameter is NULL.
ERROR_INVALID_HANDLE	Returned if the <i>hAdapter</i> handle is invalid.

ERROR CODE	DESCRIPTION
ERROR_OUTOFMEMORY	Returned when sufficient memory to perform the operation cannot be allocated.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# WlanMTESleep

1/25/2019 • 2 minutes to read

Requests that the driver to go to sleep either for a specified time interval, or indefinitely until an awake request is sent.

## Syntax

```
DWORD WlanMTESleep(
    __in                 HANDLE  hAdapter,
    __in                 ULONG   uSleepTime,
    __in                 PVOID   pvContext
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

### *uSleepTime*

[in] The time in milliseconds for the adapter to remain in sleep mode. If a value of –1 is specified, the adapter sleeps until a [WlanMTEAwake](#) request is sent.

### *pvContext*

[in] The context that uniquely identifies this request in the callback.

## Return Value

If the function succeeds, the return value is `ERROR_SUCCESS`.

If the function fails, the return value is one of the system error codes. The following table lists one of the error codes that may be returned.

ERROR CODE	DESCRIPTION
<code>ERROR_INVALID_PARAMETER</code>	Returned if the <i>uSleepTime</i> parameter is <code>NULL</code> .
<code>ERROR_INVALID_HANDLE</code>	Returned if the <i>hAdapter</i> handle is invalid.
<code>ERROR_OUTOFMEMORY</code>	Returned when sufficient memory to perform the operation cannot be allocated.

## Remarks

During sleep mode, all radios are turned off and the Wi-Fi chip is powered off. When the adapter reawakens, the application's callback handler, if one was registered with [WlanMTERegisterCallbackHandler](#), is called with the **dot11ManufacturingCallbackType** set to **dot11\_manufacturing\_callback\_sleep\_complete** and the result of the sleep operation is included.

# Requirements

**Header:** wifimte.w

## Related topics

[WlanMTEAwake](#)

[Wi-Fi manufacturing API](#)

# WlanMTEAwake

1/25/2019 • 2 minutes to read

Requests that the driver wake up from its current sleep state.

## Syntax

```
DWORD WlanMTEAwake(
    __in                  HANDLE  hAdapter
);
```

## Parameters

### *hAdapter*

[in] The handle to the Wi-Fi adapter, obtained by calling [WlanMTEOpenHandle](#).

## Return Value

If the function succeeds, the return value is `ERROR_SUCCESS`.

If the function fails, the return value is one of the system error codes. The following table lists the error codes that may be returned.

ERROR CODE	DESCRIPTION
<code>ERROR_INVALID_HANDLE</code>	Returned if the <i>hAdapter</i> handle is invalid.
<code>ERROR_OUTOFMEMORY</code>	Returned when sufficient memory to perform the operation cannot be allocated.

## Remarks

The driver must have been put into the sleep state using the [WlanMTESleep](#) function before this function is called. If the driver is not in a sleep state when this function is called, it returns `STATUS_INVALID_PARAMETER`.

## Requirements

**Header:** wifimte.w

## Related topics

[Wi-Fi manufacturing API](#)

# Adding Wi-Fi manufacturing test support to the OID interface

1/25/2019 • 2 minutes to read

To ensure that all device components are integrated, functioning correctly, calibrated properly, and meet all regulatory requirements, OEMs run a number of standard ad-hoc tests to ensure that any problems are found and corrected before the device goes to retail. These tests are also occasionally run at retail outlets to check for proper component operation. The implementation of these test interfaces and mechanisms is performed by hardware vendors (IHVs).

This section describes an extension to the existing [Wi-Fi OID documentation](#) so that IHVs can implement a standard set of interfaces that OEMs can use to create test applications.

## Assumptions

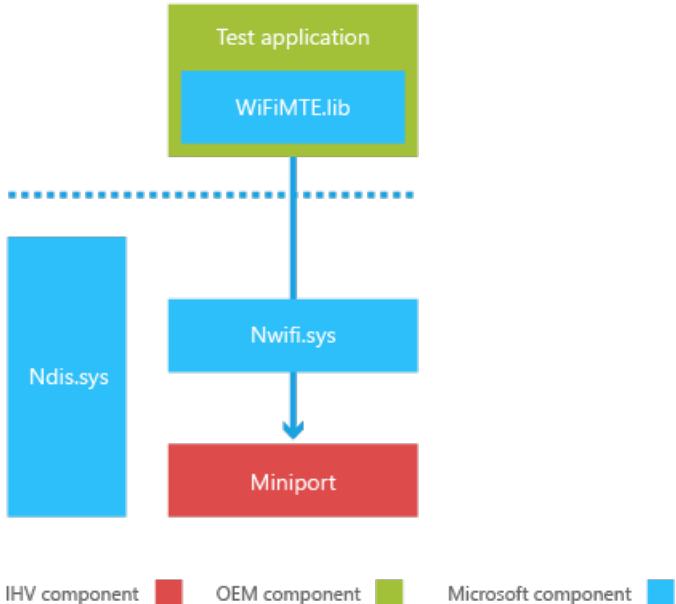
To perform these manufacturing tests, the device must be operating in a special operation mode called manufacturing mode. In manufacturing mode, only specific parts of the operating system are loaded in order to enable the proper execution of the component tests. Normal Wi-Fi operations, such as scanning and automatically connecting to networks, are disabled when the device is running in manufacturing mode.

Manufacturing mode can be entered in the manufacturing environment or during customer service. Writing to the Device Provisioning Partition (DPP) can only be performed in the manufacturing environment. If an OID that writes to the DPP is invoked in a non-manufacturing environment, the attempt to write to the DPP fails. Manufacturing operations should have only a transient effect on the system, and the state should not persist across reboots.

## Driver requirements

The Wi-Fi miniport driver must be able to operate in normal mode or manufacturing test mode, and it must be able to switch between modes at any time. The driver determines the appropriate mode during initialization by querying a specific registry key.

The following illustration shows the architecture of the manufacturing test environment.



## In this section

### [Reporting operating mode capabilities](#)

Describes the requirements and behavior for reporting changes with drivers operating in manufacturing test mode.

### [Supporting updated OID behavior in manufacturing mode](#)

Describes the updated OIDs that must be supported by the Wi-Fi miniport driver.

### [Supporting existing OID commands in manufacturing mode](#)

Describes the existing OIDs that must be supported by the Wi-Fi miniport driver.

### [Supporting new OID commands for manufacturing mode](#)

Describes the new OIDs that must be supported by the Wi-Fi miniport driver.

### [Supporting new callbacks for manufacturing mode](#)

Describes the new OID callback that must be supported by the Wi-Fi miniport driver.

# Reporting operating mode capabilities

1/25/2019 • 2 minutes to read

If a Wi-Fi driver supports running in manufacturing mode, it should add manufacturing mode to its list of supported capabilities. You can query the supported operation mode capabilities by using the **OID\_DOT11\_OPERATION\_MODE\_CAPABILITY** command, which will return information on the operation modes supported by the driver. For more info about **OID\_DOT11\_OPERATION\_MODE\_CAPABILITY**, see [Supporting updated OID behavior in manufacturing mode](#).

To switch the driver's operation mode to manufacturing mode, use the **OID\_DOT11\_CURRENT\_OPERATION\_MODE** command to ensure that manufacturing testing will not conflict with the driver's behavior in any of its other modes. For more info about **OID\_DOT11\_CURRENT\_OPERATION\_MODE**, see [Supporting updated OID behavior in manufacturing mode](#).

## Related topics

[Adding Wi-Fi manufacturing test support to the OID interface](#)

# Supporting updated OID behavior in manufacturing mode

1/25/2019 • 2 minutes to read

When running in manufacturing mode, Wi-Fi miniport drivers must add support for the following updated OIDs.

## OID\_DOT11\_OPERATION\_MODE\_CAPABILITY

The **OID\_DOT11\_OPERATION\_MODE\_CAPABILITY** command is called in query mode to return the list of operation modes supported by the driver. This command functions as previously documented, but drivers are now required to support a new operation mode, **DOT11\_OPERATION\_MODE\_MANUFACTURING**, which is the context in which manufacturing operations are performed. For complete documentation of this OID, see [OID\\_DOT11\\_OPERATION\\_MODE\\_CAPABILITY](#) on MSDN.

```
#define DOT11_OPERATION_MODE_UNKNOWN          0x00000000
#define DOT11_OPERATION_MODE_STATION           0x00000001
#define DOT11_OPERATION_MODE_AP               0x00000002
#define DOT11_OPERATION_MODE_EXTENSIBLE_STATION 0x00000004
#define DOT11_OPERATION_MODE_EXTENSIBLE_AP     0x00000008
#define DOT11_OPERATION_MODE_WFD_DEVICE       0x00000010
#define DOT11_OPERATION_MODE_WFD_GROUP_OWNER   0x00000020
#define DOT11_OPERATION_MODE_WFD_CLIENT        0x00000040
#define DOT11_OPERATION_MODE_MANUFACTURING    0x40000000
#define DOT11_OPERATION_MODE_NETWORK_MONITOR   0x80000000

typedef struct _DOT11_OPERATION_MODE_CAPABILITY {
    ULONG uReserved;
    ULONG uMajorVersion;
    ULONG uMinorVersion;
    ULONG uNumOfTXBuffers;
    ULONG uNumOfRXBuffers;
    ULONG uOpModeCapability;
} DOT11_OPERATION_MODE_CAPABILITY, * PDOT11_OPERATION_MODE_CAPABILITY;
```

## OID\_DOT11\_CURRENT\_OPERATION\_MODE

The **OID\_DOT11\_CURRENT\_OPERATION\_MODE** command can be called in either set or query mode to configure or return the driver's current operation mode.

This command functions as previously documented, but the driver is now required to support the **DOT11\_OPERATION\_MODE\_MANUFACTURING** operation mode. For complete documentation of this OID, see [OID\\_DOT11\\_CURRENT\\_OPERATION\\_MODE](#) on MSDN.

```
typedef struct _DOT11_CURRENT_OPERATION_MODE {
    ULONG uReserved;
    ULONG uCurrentOpMode;
} DOT11_CURRENT_OPERATION_MODE, * PDOT11_CURRENT_OPERATION_MODE;
```

### *uCurrentOpMode*

[in] Specifies the driver operation mode to be set. This parameter also functions as a placeholder for the driver to return the operation mode when called in query mode. If the driver does not support the requested operation mode, it should return **NDIS\_STATUS\_BAD\_VERSION**.

## Related topics

[Adding Wi-Fi manufacturing test support to the OID interface](#)

# Supporting existing OID commands in manufacturing mode

1/25/2019 • 2 minutes to read

When running in manufacturing mode, Wi-Fi miniport drivers must add support for the following existing OIDs.

## OID\_GEN\_SUPPORTED\_GUIDS

The **OID\_GEN\_SUPPORTED\_GUIDS** command is called in query mode to return the set of supported GUIDs. For complete documentation, see [OID\\_GEN\\_SUPPORTED\\_GUIDS](#) on MSDN.

**Note** This OID is typically called for compatibility purposes. The driver can choose to ignore it, if desired, and return **NDIS\_STATUS\_NOT\_SUPPORTED** instead.

## OID\_GEN\_VENDOR\_ID

The **OID\_GEN\_VENDOR\_ID** command is called in query mode to return the 3-byte IEEE-registered vendor code followed by a single byte assigned by the vendor that identifies a particular NIC. For complete documentation, see [OID\\_GEN\\_VENDOR\\_ID](#) on MSDN.

## OID\_GEN\_VENDOR\_DESCRIPTION

The **OID\_GEN\_VENDOR\_DESCRIPTION** command is called in query mode to return a NULL-terminated string that describes the NIC in ANSI format. For complete documentation, see [OID\\_GEN\\_VENDOR\\_DESCRIPTION](#) on MSDN.

## OID\_GEN\_CURRENT\_LOOKAHEAD

The **OID\_GEN\_CURRENT\_LOOKAHEAD** command is called in set mode to specify the number of bytes of received packet data to be sent to the protocol driver. For complete documentation, see [OID\\_GEN\\_CURRENT\\_LOOKAHEAD](#) on MSDN.

**Note** This OID is typically called for compatibility purposes. The driver can choose to ignore it, if desired, and return **NDIS\_STATUS\_NOT\_SUPPORTED** instead.

## OID\_PM\_ADD\_WOL\_PATTERN

The **OID\_PM\_ADD\_WOL\_PATTERN** command is called in set mode to specify the WOL pattern. For complete documentation, see [OID\\_PM\\_ADD\\_WOL\\_PATTERN](#) on MSDN.

**Note** This OID is typically called for compatibility purposes. The driver can choose to ignore it, if desired, and return **NDIS\_STATUS\_NOT\_SUPPORTED** instead.

## OID\_DOT11\_RESET\_REQUEST

The **OID\_DOT11\_RESET\_REQUEST** command is called in query mode to return the IEEE MAC address used by the driver. For complete documentation, see [OID\\_DOT11\\_RESET\\_REQUEST](#) on MSDN.

## OID\_DOT11\_CURRENT\_ADDRESS

The **OID\_DOT11\_CURRENT\_ADDRESS** command is called in query mode to return the IEEE MAC address used by the driver. For complete documentation, see [OID\\_DOT11\\_CURRENT\\_ADDRESS](#) on MSDN.

## OID\_DOT11\_SUPPORTED\_PHY\_TYPES

The **OID\_DOT11\_SUPPORTED\_PHY\_TYPES** command is called in query mode to request the list of PHY types supported by the 802.11 station. For complete documentation, see [OID\\_DOT11\\_SUPPORTED\\_PHY\\_TYPES](#) on MSDN.

## OID\_DOT11\_CURRENT\_PHY\_ID

The **OID\_DOT11\_CURRENT\_PHY\_ID** command is called in set mode to set the current PHY ID. For complete documentation, see [OID\\_DOT11\\_CURRENT\\_PHY\\_ID](#) on MSDN.

## OID\_DOT11\_HARDWARE\_PHY\_STATE

The **OID\_DOT11\_HARDWARE\_PHY\_STATE** command is called in query mode to return the PHY power state. For complete documentation, see [OID\\_DOT11\\_HARDWARE\\_PHY\\_STATE](#) on MSDN.

## OID\_DOT11\_NIC\_POWER\_STATE

The **OID\_DOT11\_NIC\_POWER\_STATE** command is called in query mode to return the NIC power state. For complete documentation, see [OID\\_DOT11\\_NIC\\_POWER\\_STATE](#) on MSDN.

## Related topics

[Adding Wi-Fi manufacturing test support to the OID interface](#)

# Supporting new OID commands for manufacturing mode

1/25/2019 • 7 minutes to read

When running in manufacturing mode, Wi-Fi miniport drivers must add support for the following new OID commands. The driver should ensure that the device is currently in manufacturing mode prior to calling any of these commands. For more info, see [Determine if MMOS is running](#). Some of the parameters specified in the API may be IHV-specific.

## OID\_DOT11\_MANUFACTURING\_TEST

OID\_DOT11\_MANUFACTURING\_TEST is called as a method request in the driver to perform a specific test. This OID should never be used during normal operation.

```
typedef struct _DOT11_MANUFACTURING_TEST {
    DOT11_MANUFACTURING_TEST_TYPE dot11ManufacturingTestType;
    ULONG uBufferLength;
    UCHAR ucBuffer[1];
} DOT11_MANUFACTURING_TEST, * PDOT11_MANUFACTURING_TEST;
```

### *dot11ManufacturingTestType*

[in] Specifies the manufacturing test to be run. The data type for this member is one of the values of the **DOT11\_MANUFACTURING\_TEST\_TYPE** enumeration.

The DOT11 manufacturing test type enumeration is defined as follows:

```
typedef enum _DOT11_MANUFACTURING_TEST_TYPE {
    dot11_manufacturing_test_unknown = 0,
    dot11_manufacturing_test_self_start = 1,
    dot11_manufacturing_test_self_query_result = 2,
    dot11_manufacturing_test_rx = 3,
    dot11_manufacturing_test_tx = 4,
    dot11_manufacturing_test_set_data = 5,
    dot11_manufacturing_test_query_data = 6,
    dot11_manufacturing_test_sleep = 7,
    dot11_manufacturing_test_awake = 8,
    dot11_manufacturing_test_IHV_start = 0x80000000,
    dot11_manufacturing_test_IHV_end = 0xffffffff
} DOT11_MANUFACTURING_TEST_TYPE, * PDOT11_MANUFACTURING_TEST_TYPE;
```

### *uBufferLength*

[in] The length, in bytes, of the **DOT11\_MANUFACTURING\_TEST** structure and any additional data specific to the test appended at the end.

### *ucBuffer*

[in] The buffer containing optional data as specified by the **dot11DiagnosticsTestType** member.

## dot11\_manufacturing\_test\_self\_start

The **dot11\_manufacturing\_test\_self\_start** command is called to request the driver to test WLAN IC connectivity, FEM IC connectivity, or the WLAN-BT coexistence interface.

**DOT11\_DIAGNOSTIC\_SELF\_TEST\_BT\_COEXISTENCE** is only applicable if the WLAN and Bluetooth chips are on separate ICs. If they are on the same module, this test is not supported and the miniport should return **NDIS\_STATUS\_NOT\_SUPPORTED**.

When called, the driver should run the requested tests as defined in the **DOT11\_MANUFACTURING\_SELF\_TEST\_SET\_PARAMS** structure and return success when the tests have been started. On completion, whether the tests have succeeded or failed, the driver should indicate the test status by using the **NDIS\_STATUS\_DOT11\_MANUFACTURING\_CALLBACK** callback handler, with the *dot11ManufacturingCallbackType* set to **dot11\_manufacturing\_callback\_self\_test\_complete** and the status describing the result of the test. The driver will then call the **OID\_DOT11\_MANUFACTURING\_TEST** oid with the **dot11\_manufacturing\_test\_self\_query\_result** command to query the detailed result of the test.

```
typedef struct _DOT11_MANUFACTURING_SELF_TEST_SET_PARAMS {
    DOT11_MANUFACTURING_SELF_TEST_TYPE SelfTestType;
    ULONG uTestID;
    ULONG uPinBitMask;
    PVOID pvContext;
    ULONG uBufferLength;
    UCHAR ucBufferIn[1];
} DOT11_MANUFACTURING_SELF_TEST_SET_PARAMS, *PDOT11_MANUFACTURING_SELF_TEST_SET_PARAMS;
```

#### *SelfTestType*

[in] Specifies the type of self-test to be run by the driver. The data type for this member is the **DOT11\_MANUFACTURING\_SELF\_TEST\_TYPE** enumeration with one of the following values:

#### **DOT11\_MANUFACTURING\_SELF\_TEST\_INTERFACE**

- Control and data interface to WLAN
- Clock request
- Sleep clock
- Interrupt and power supply lines
- All related connections

#### **DOT11\_MANUFACTURING\_SELF\_TEST\_RF\_INTERFACE**

- Control and RF interface to FEM IC
- FEM power supply
- Transmit signal on loopback path from TX interface to RX interface and validate.

#### **DOT11\_MANUFACTURING\_SELF\_TEST\_BT\_COEXISTENCE**

- Set line states from Bluetooth side and read line states from WLAN side
- Verify each pin's state

#### *uTestID*

[in] ID of the test to be run.

#### *uPinBitMask*

[in] Bit mask of pins to be tested.

#### *pvContext*

[in] The context value to be returned to the application by using **dot11\_manufacturing\_callback\_self\_test\_complete** callback when the driver has completed the tests.

#### *uBufferLength*

[in, optional] The length of the buffer containing additional input for the self-test.

#### *ucBufferIn*

[in, optional] The buffer that contains additional input for the self-test.

## dot11\_manufacturing\_test\_self\_query\_result

This command gets the results of a previously requested self-test. It should only be called when the driver has indicated that the self-test is complete by using the **NDIS\_STATUS\_DOT11\_MANUFACTURING\_CALLBACK** with the *dot11ManufacturingCallbackType* set to **dot11\_manufacturing\_callback\_self\_test\_complete** and the status describing the result of the test.

```
typedef struct _DOT11_MANUFACTURING_SELF_TEST_QUERY_RESULTS {
    DOT11_MANUFACTURING_SELF_TEST_TYPE SelfTestType;
    ULONG uTestID;
    BOOLEAN bResult;           // PASS/FAIL
    ULONG uPinFailedBitMask;   // Detected PIN faults
    PVOID pvContext;
    ULONG uBytesWrittenOut;
    UCHAR ucBufferOut[1];      // Additional output from self-test (optional)
} DOT11_MANUFACTURING_SELF_TEST_QUERY_RESULTS, *PDOT11_MANUFACTURING_SELF_TEST_QUERY_RESULTS;
```

#### *SelfTestType*

[in] Specifies the type of self-test whose result is being queried. The data type for this member is the **DOT11\_MANUFACTURING\_SELF\_TEST\_TYPE** enumeration.

#### *uTestID*

[in] ID of the test to be run.

#### *bResult*

[out] The result of the test. **True** if the test passed, **False** if it failed.

#### *uPinFailedBitMask*

[out] The bit mask of any detected PIN faults.

#### *pvContext*

[in] The context used when the driver indicated that the tests were complete.

#### *uBytesWrittenOut*

[out] The length of the buffer that contains any additional returned output from the self-test.

#### *ucBufferOut*

[in, out, optional] The buffer of length *uBytesWrittenOut* that contains additional output from the self-test.

## dot11\_manufacturing\_test\_rx

The **dot11\_manufacturing\_test\_rx** read-only command tests and verifies that there is connectivity between the antenna port and WLAN IC.

To test this connectivity, a signal generator generates a non-modulated carrier wave (CW) at a certain frequency and power that will be measured and returned by the device under test (DUT). If the band and/or channel setting are inconsistent, then the driver returns **STATUS\_INVALID\_PARAMETER**.

```

typedef struct _DOT11_MANUFACTURING_FUNCTIONAL_TEST_RX {
    BOOLEAN bEnabled;
    DOT11_BAND Dot11Band;
    ULONG uChannel;
    LONG PowerLevel;
} DOT11_MANUFACTURING_FUNCTIONAL_TEST_RX, * PDOT11_MANUFACTURING_FUNCTIONAL_TEST_RX;

```

*bEnabled*

[out] **True** if the driver detected a signal at the specified band and channel. **False** if no signal was detected.

*Dot11Band*

[in] The band on which the signal is to be detected.

*uChannel*

[in] The channel on which the signal is to be detected. The channel range depends on the band and supported PHYs.

*PowerLevel*

[out] The power level of the received signal detected at the antenna, returned as the RSSI measured in dBm. This is valid only if *bEnabled* is **True**.

## dot11\_manufacturing\_test\_tx

The **dot11\_manufacturing\_test\_tx** set-only command validates the connection from the chipset to the FEM output.

To perform this test, a signal analyzer is physically connected to the antenna port and the DUT is requested to transmit a CW with specific band, channel, and power level settings. The driver also measures its own ADC reading for the transmitted signal and returns it to the application.

```

typedef struct _DOT11_MANUFACTURING_FUNCTIONAL_TEST_TX {
    BOOLEAN bEnable;
    BOOLEAN bOpenLoop;
    DOT11_BAND Dot11Band;
    ULONG uChannel;
    ULONG uSetPowerLevel;
    LONG ADCPowerLevel;
} DOT11_MANUFACTURING_FUNCTIONAL_TEST_TX, * PDOT11_MANUFACTURING_FUNCTIONAL_TEST_TX;

```

*bEnable*

[in] If set, this command enables transmission. If not set, transmission at the specified band and channel are disabled.

*bOpenLoop*

[in] If set to **true**, this parameter indicates that the driver is requested to use an open loop power control and return the read signal value in *ADCPowerLevel*. If set to **false**, the driver will not use an open loop power control.

If this value is set and the hardware does not support open loop power control, the driver returns

**NDIS\_STATUS\_NOT\_SUPPORTED**.

*Dot11Band*

[in] The band on which the signal is to be transmitted.

*uChannel*

[in] The channel on which the signal is to be transmitted. The channel range depends on the band and supported PHYs.

*uSetPowerLevel*

[in] The power level of the transmitted signal. This is returned as a percentage of the maximum possible power level.

#### *ADCPowerLevel*

[out, optional] The current signal level detected at the antenna, returned as a RAW value. The interpretation of this value is specified by the IHV.

This must be set if *bOpenLoop* is **True** and the hardware supports it.

## dot11\_manufacturing\_test\_set\_data

The **dot11\_manufacturing\_test\_set\_data** set-only command enables the application to write data at a specific location.

```
typedef struct _DOT11_MANUFACTURING_TEST_SET_DATA {
    ULONG uKey;
    ULONG uOffset;
    ULONG uBufferLength;
    UCHAR ucBufferIn[1];
} DOT11_MANUFACTURING_TEST_SET_DATA, * PDOT11_MANUFACTURING_TEST_SET_DATA;
```

#### *uKey*

[in] The key is IHV specific and can be either a reference to a specific register or an entry from a named table.

#### *uOffset*

[in] The offset within the data.

#### *uBufferLength*

[in] The number of data bytes to be contained in the buffer of additional test data.

#### *ucBufferIn*

[in] The buffer containing the additional test data of length *uBufferLength*.

## dot11\_manufacturing\_test\_query\_data

The **dot11\_manufacturing\_test\_query\_data** command enables the application to read data at a specific location.

```
typedef struct _DOT11_MANUFACTURING_TEST_QUERY_DATA {
    ULONG uKey;
    ULONG uOffset;
    ULONG uBufferLength;
    ULONG uBytesRead;
    UCHAR ucBufferOut[1];
} DOT11_MANUFACTURING_TEST_QUERY_DATA, * PDOT11_MANUFACTURING_TEST_QUERY_DATA;
```

#### *uKey*

[in] The key is IHV specific and can be either a reference to a specific register or an entry from a named table.

#### *uOffset*

[in] The offset within the data.

#### *uBufferLength*

[in] The number of data bytes to be read in the buffer.

#### *uBytesRead*

[out] The actual number of data bytes read by the driver.

#### *ucBufferOut*

[out] Contains the data read by the driver.

## dot11\_manufacturing\_test\_sleep

The **dot11\_manufacturing\_test\_sleep** command instructs the Wi-Fi chipset to go into its lowest power state, for either a specified time or indefinitely.

For this test, all radios should be turned off and the Wi-Fi chipset should be powered off. The test verifies that Wi-Fi can enter the sleep state, that the current consumption is within the specified limits, and that there is no current drawn when Wi-Fi is switched off.

The driver can be awakened from the sleep state at any time by using the **dot11\_manufacturing\_test\_awake** command. If the sleep time-out is set to –1, the driver should sleep indefinitely unless asked to wake up by using **dot11\_manufacturing\_test\_awake**. When the driver wakes up, either due to the time-out expiring or as a result of the awake command, it should indicate its awake status by using the **NDIS\_STATUS\_DOT11\_MANUFACTURING\_CALLBACK** callback handler with the *dot11ManufacturingCallbackType* set to **dot11\_manufacturing\_callback\_sleep\_complete**.

```
typedef struct _DOT11_MANUFACTURING_TEST_SLEEP {
    ULONG uSleepTime;
    PVOID pvContext;
} DOT11_MANUFACTURING_TEST_SLEEP, * PDOT11_MANUFACTURING_TEST_SLEEP;
```

### *uSleepTime*

[in] The amount of time for the driver to sleep, in milliseconds. If set to –1, the driver enters sleep state until awakened by using the **dot11\_manufacturing\_test\_awake** command.

### *pvContext*

[in] The context used when the driver returns the test completion state to the application by using **dot11\_manufacturing\_callback\_sleep\_complete**.

## dot11\_manufacturing\_test\_awake

The **dot11\_manufacturing\_test\_awake** command causes the Wi-Fi chipset to wake up from its lowest-power sleep state. The driver returns **STATUS\_INVALID\_PARAMETER** if this command is sent when the chipset is already awake.

## Related topics

[Adding Wi-Fi manufacturing test support to the OID interface](#)

# Supporting new callbacks for manufacturing mode

1/25/2019 • 2 minutes to read

When running in manufacturing mode, Wi-Fi miniport drivers must add support for the following new callback.

## NDIS\_STATUS\_DOT11\_MANUFACTURING\_CALLBACK

The **NDIS\_STATUS\_DOT11\_MANUFACTURING\_CALLBACK** callback is used to indicate completion status for certain requests. The data structure used for this callback is defined here.

```
typedef enum _DOT11_MANUFACTURING_CALLBACK_TYPE {
    dot11_manufacturing_callback_unknown = 0,
    dot11_manufacturing_callback_self_test_complete = 1,
    dot11_manufacturing_callback_sleep_complete = 2,
    dot11_manufacturing_callback_IHV_start = 0x80000000,
    dot11_manufacturing_callback_IHV_end = 0xffffffff
} DOT11_MANUFACTURING_CALLBACK_TYPE, * PDOT11_MANUFACTURING_CALLBACK_TYPE;

typedef struct DOT11_MANUFACTURING_CALLBACK_PARAMETERS {
#define DOT11_MANUFACTURING_CALLBACK_REVISION_1 1
    NDIS_OBJECT_HEADER Header;
    DOT11_MANUFACTURING_CALLBACK_TYPE dot11ManufacturingCallbackType;
    ULONG uStatus;
    PVOID pvContext;
} DOT11_MANUFACTURING_CALLBACK_PARAMETERS, * PDOT11_MANUFACTURING_CALLBACK_PARAMETERS;
```

### **dot11\_manufacturing\_callback\_self\_test\_complete**

**dot11\_manufacturing\_callback\_self\_test\_complete** is called by the driver when a requested self-test is completed by the driver. This callback must return the context for self-test request as well as the self-test result. The driver then calls the **OID\_DOT11\_MANUFACTURING\_TEST** OID with the **dot11\_manufacturing\_test\_self\_query\_result** command to obtain the detailed result of the test.

### **dot11\_manufacturing\_callback\_sleep\_complete**

**dot11\_manufacturing\_callback\_sleep\_complete** is called when a requested sleep command is executed by the driver. This callback must return the context for the sleep request as well as the status, whether success or failure. This callback is also called by the driver when the application sends a request to wake the driver from a sleep state.

## Related topics

[Adding Wi-Fi manufacturing test support to the OID interface](#)

# Flashing tools

1/25/2019 • 3 minutes to read

Each manufacturer has different techniques and tooling that they will use to manufacture and service a Windows 10 Mobile device. The best technical expertise regarding manufacturing resides within those who built the OEM manufacturing line. This means that the OEM will need to determine which flashing and manufacturing process will work best for them. OEM service centers may have unique needs that will also influence the selection of flashing tools. The OEM will need to determine how to test and validate that the selected tools and processes meets their cost, quality, and other unique manufacturing objectives.

The OEM uses the Microsoft supplied imaging tools to create the FFU OS images that are flashed to the device.

## Flashing tools comparison

The OEM may need to develop a custom flashing tool to address the life cycle needs of the device. Other flashing options have limitations that the OEM should understand before deciding to use them.

The following table summarizes the flashing tool options.

SCENARIO	MICROSOFT FFU ENGINEERING TOOL	OEM CUSTOM FFU TOOL	SOC PROVIDED MANUFACTURING FLASHING TOOL	GANG PROGRAMMER
Engineering and Development	Yes	Yes	Yes	N/A
Manufacturing	No	Yes	Yes	Yes
Service Center	Yes	Yes	No	N/A

## OEM custom flashing tool

To create a flashing tool for manufacturing, the OEM must develop their own tools customized to their manufacturing environment and equipment.

Depending on the OEMs requirements, the flashing tools may also need to address a number of scenarios described in [Field service scenarios](#).

For more info, see [Developing custom OEM flashing tools](#).

## SoC provided manufacturing flashing tool

For information on the SoC provided manufacturing flashing tools, contact the SoC provider.

## Gang programmer

There are a number of options available to the OEM to flash binary images. The OEM can use their unique flashing tools as well as gang programmer technologies to manufacture the device if they find that those options are more suitable to their environment.

If the OEM uses a gang programmer they will need to develop a custom tool to convert the FFU image to a raw binary image. The conversion tool will need to:

1. Open a raw binary file in the format expected by the gang programmer.
2. Read in the FFU file and parse the file data as specified in [FFU image format](#).
3. Write out the data referenced in the FFU BLOCK\_DATA\_ENTRY elements to the raw file.
4. When there are no more entries, write out any metadata or padding needed for the raw format and then close the raw binary file.

## FFUTool support limitations

The Microsoft provided FFUTool full flash update (FFU) technology is designed for engineering development, and testing purposes; it is not supported for use in manufacturing. Each OEM must determine if the FFUTool is suitable for use in their service center environments.

## FFUTool known issues

Using the FFUTool has a number of significant limitations that are summarized here.

### **USB hub activity may cause flashing failures**

Some USB hubs have been known to cause reliability issues even when flashing devices in serial due to hardware interference to the streaming FFU data.

Multiple devices that share a single USB hub cannot be connected and disconnected while other connected device are flashing. This uncovers a known hardware issue with some USB controllers. For more info, see [KB908673](#). You should not unplug USB devices when device flashing is underway.

### **USB cable length is limited to 3 feet**

Flashing may be less reliable when using USB cables longer than 3 feet (.91 meters), or if your flashing setup contains consecutive cables that total to more than 3 feet. This is due to hardware limitations of data transfer in longer cables.

### **Flashing time per phone**

You will need to evaluate whether the flashing time per device meets their objectives for your manufacturing line.

# Developing custom OEM flashing tools

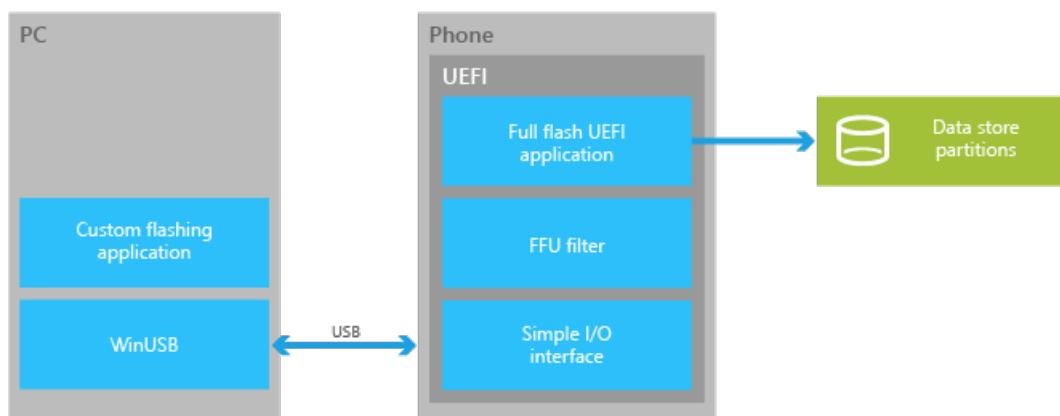
1/25/2019 • 2 minutes to read

OEMs can use the full flash update (FFU) image format and simple UEFI USB protocols to create custom flashing tools. An OEM custom flashing tool can integrate in with existing systems and support a range of scenarios discussed in [Flashing tools](#).

## UEFI flashing application

The OEM must flash the device from a UEFI application using a specific image layout that is discussed in [FFU image format](#).

This diagram summarizes the communication flow from the PC flashing tool to the device using the UEFI simple Windows Phone I/O protocol.

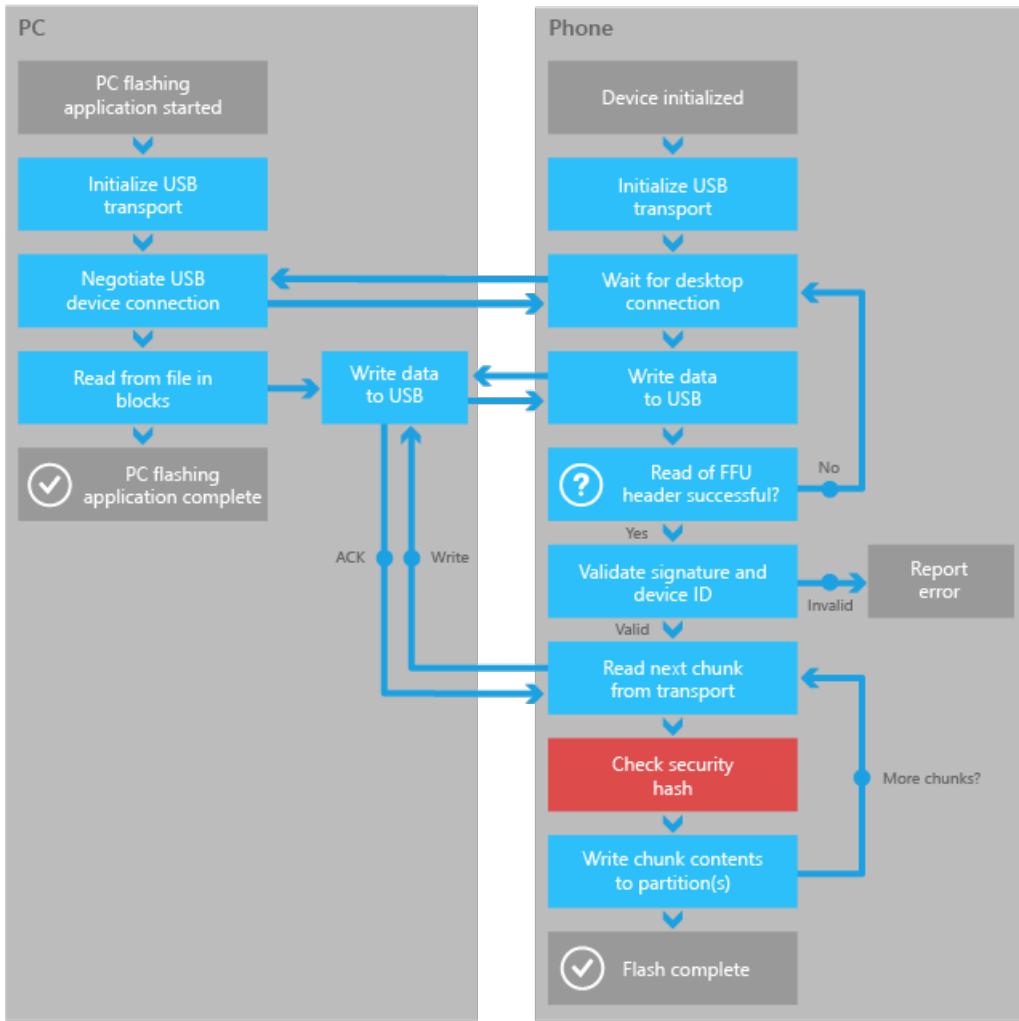


For more info on available USB APIs see, [UEFI flashing protocols](#).

## PC flashing application

The image is transferred to the device that is running the UEFI flashing application using a simple PC side client program. The PC application establishes a USB connection to the device and writes the data over that connection. The validation and verification of the image occurs in the UEFI flashing application running on the device.

The following diagram summarizes the overall flow of the OEM custom flashing PC application and the UEFI application.



**Note** This diagram illustrates one possible solution. The OEM is encouraged to modify this approach to create an optimal solution that best suits their needs.

## Checking SMBIOS values before flashing

To ensure that the correct image is flashed to the proper device, the OEM must check the SMBIOS system information structure values on the device. The check must confirm that the device platform ID values in the image, matches the SMBIOS system information structure values on the phone. Either the Manufacturer.Family.ProductName.Version or Manufacturer.Family.ProductName from SMBIOS must match the value in the image before flashing can proceed.

The device platform ID string is shown below.

### **Manufacturer.Family.ProductName.Version**

#### **Engineering devices and blank device IDs**

With a new engineering device, the OEM can use the SMBIOS values to determine if it is acceptable to flash an image that contains test signed certificates. The OEM may determine that test signed images may have blank system information structure values, where production signed images must have SMBIOS system information structure values that have been populated.

## Implementing signed image validation

FFU images contain elements such as hashes, signatures and catalogs, which must be used to validate the image. For more info, see [Implementing image integrity validation in custom flashing tools](#).

# UEFI flashing protocols

[UEFI USB function protocol](#)

Describes the **EFI\_USBFN\_IO\_PROTOCOL**.

[UEFI simple I/O protocol](#)

Describes the **EFI\_SIMPLE\_WINPHONE\_IO\_PROTOCOL**.

[UEFI check signature protocol](#)

Describes the **EFI\_CHECKSIG\_PROTOCOL**.

## Related topics

[Flashing tools](#)

[Manufacturing](#)

# Flashing security requirements

1/25/2019 • 2 minutes to read

Before flashing occurs, all OEM flashing mechanisms must validate cryptographic signatures in the image that chain to keys owned by the OEM. This validation of the cryptographic signature on the image must be done on the device (not on a desktop flashing tool) and must be done before the image is flashed to eMMC memory. This requirement is intended to protect devices from being compromised by users attempting to subvert the security protections in the system. Mechanisms that can flash or update the state of the device (debuggers, memory inspectors, etc.) and that are not properly secured could be used by an attacker to circumvent the device's security mechanisms.

The implemented solutions must be fully resilient even if the device is reverse engineered (that is, even if the user understands the operation of the security code in the firmware or SoC).

## Flashing security requirements summary

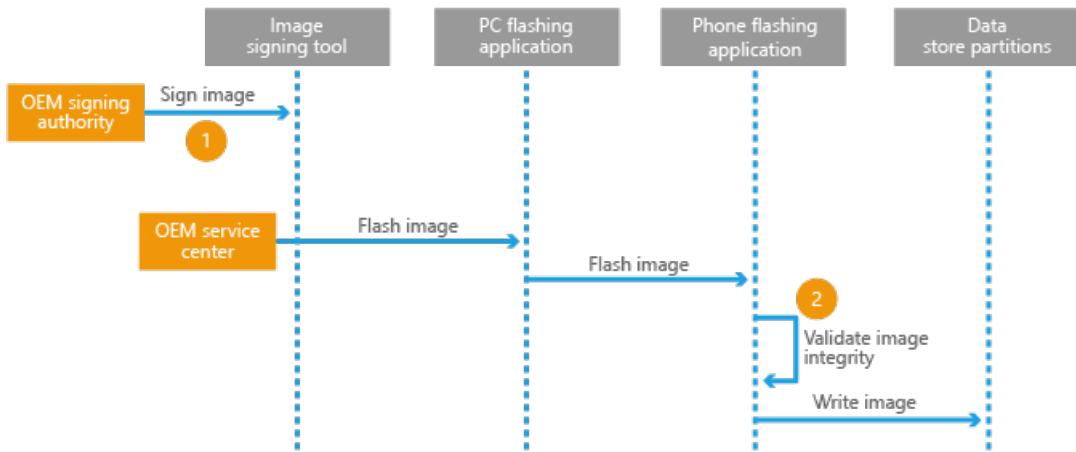
Security on retail devices must also meet the following requirements:

- Only secure flashing technologies can be included on retail devices.
- Images to be flashed must be cryptographically signed through a process that is controlled and verified by the OEM or Microsoft.
- Cryptographic signatures on images to be flashed must be verified by code on the device immediately before flashing.
- Code that verifies cryptographic signatures on the device must be tamper proof, and cryptographic key material on the device used for verification must be trustworthy at the point in time that it is used.
- Sufficiently large encryption algorithms must be used—at a minimum, RSA 2048 with SHA-256.
- The flashing tool must check the SMBIOS device identification values before flashing. For more info, see [Developing custom OEM flashing tools](#).
- The flashing tool must implement image integrity validation to protect against tampered images. For more info, see [Developing custom OEM flashing tools](#).
- Industry best practices for source code development and supply chain security must be followed.
- A threat model must be applied to identify priority risks, threats, and vulnerabilities.

## Recommended flashing solution

The following diagram illustrates a flashing solution that conforms to the flashing security requirements. Images to be flashed must be signed by the OEM signing authority using documented procedures, ensuring that there is a high level of control and security throughout the process. The component that initiates the flashing operation must perform a cryptographic verification of the signature on the image to be flashed using trustworthy key material immediately before flashing.

The cryptographic key should be stored so that it is tied either to secure boot key material (OEM\_PK\_HASH or UEFI Variable PK) or to a public key that is embedded in a binary that is cryptographically validated by secure boot. It is not acceptable to store the public or private key in the DPP, or in any other unsigned and unvalidated file store.



#### Required operations

- 1 Sign the image to be flashed
- 2 Validate image integrity:
  - Catalog signature validation
  - Hash of the table of hashes validation
  - Data chunk validation by using the hash table entries

## Related topics

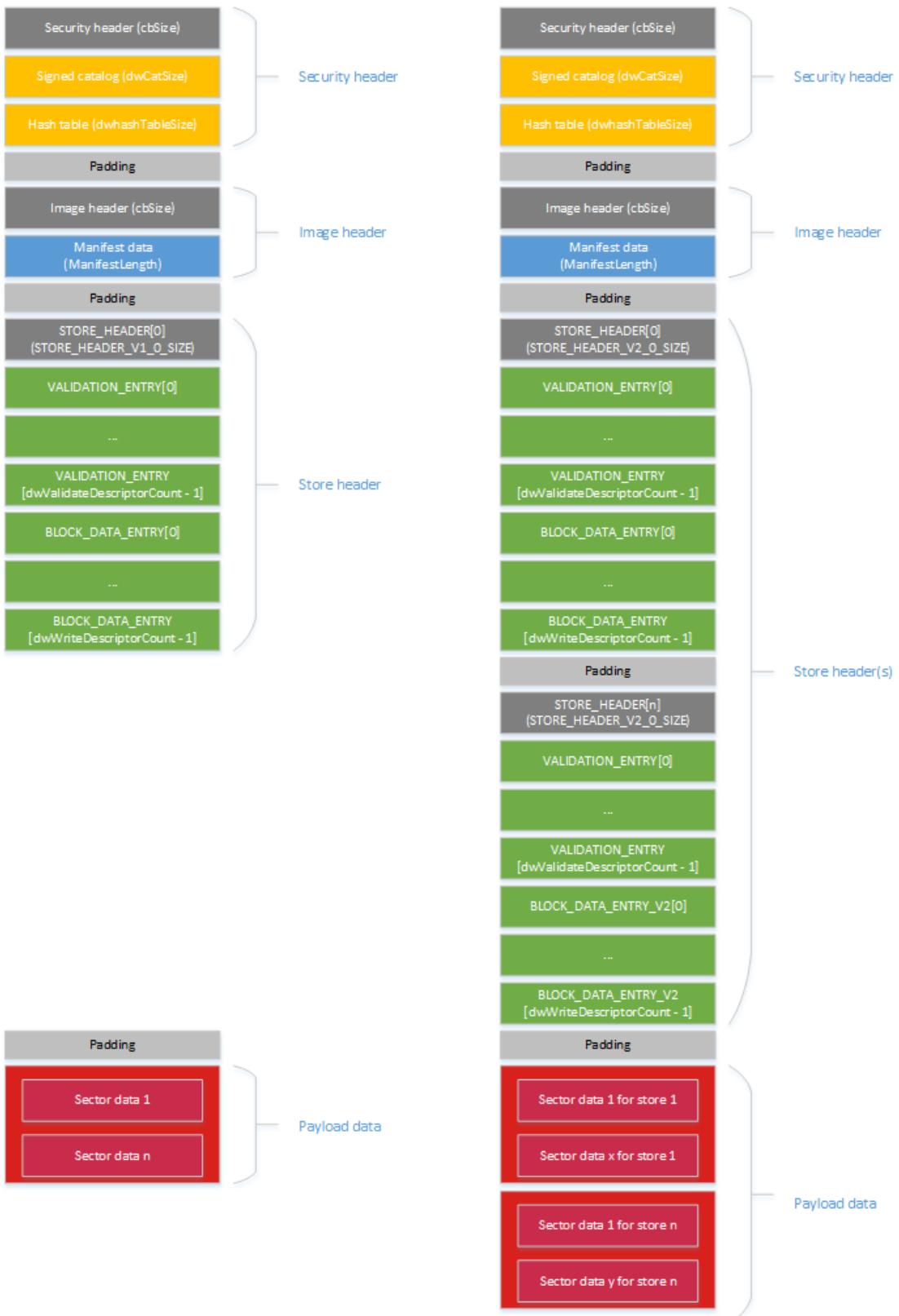
[Developing custom OEM flashing tools](#)

[Implementing image integrity validation in custom flashing tools](#)

# FFU image format

1/25/2019 • 7 minutes to read

The following diagram shows both V1 and V2 FFU format. A major change introduced in V2 FFU format is the support for multiple data stores – each store contains sector-based data targeting a unique physical partition.



## Security header region

### cbSize

The size of the SECURITY\_HEADER struct. Used in conjunction with the signature string to identify the FFU security header.

### Signature string

A hard-coded ASCII string of "SignedImage" that identifies this image as a secure FFU image.

### Chunk size in KB

The size of chunks used to generate the hash table. Used to break the image up into hashable chunks for validation against the hash table entries and ensure the image has not been tampered with since creation.

### Hash algorithm ID

Defines which hash algorithm was used to generate the hash table.

### Catalog size

The size in bytes of the catalog after the security header.

### Hash table size

The size in bytes of the hash table after the security header and catalog.

### **Security header**, byte count: cbSize

```
#define SECURITY_SIGNATURE "SignedImage"

typedef struct _SECURITY_HEADER
{
    UINT32 cbSize;           // size of struct, overall
    BYTE   signature[12];    // "SignedImage"
    UINT32 dwChunkSizeInKb; // size of a hashed chunk within the image
    UINT32 dwAlgId;         // algorithm used to hash
    UINT32 dwCatalogSize;   // size of catalog to validate
    UINT32 dwHashTableSize; // size of hash table
} SECURITY_HEADER;
```

### **Signed Catalog**, byte count: dwCatalogSize

A catalog file containing the hash of the hash table blob that will be signed and must match one of the certificates on the device. This approach allows checking for a signature up front without having the full image on the device before flashing. Streaming data is checked as it is received against the hash table entries.

### **Hash table data**, byte count: dwHashTableSize

The actual hashes for each chunk of the base image. Chunk validation begins at the image header and ends at the end of the FFU.

### **Padding** - next section starts on a chunk boundary, byte count: variable

After the hash table padding (blank space) is added to fill out to current chunk. This ensures that the full secure header, catalog, and hash table end at a chunk boundary and the actual image header and beyond are chunk aligned.

## Image header region

### cbSize

The size in bytes of the ImageHeader struc. Used in conjunction with the signature string to identify the FFU image header.

### Signature string

A hard-coded string of "ImageFlash" that identifies this image as an FFU image.

#### Manifest Length

The size in bytes of the manifest data immediately following the image header.

#### Chunk size

The size of chunks used to generate the hash table. Used to break the image up into hashable chunks for validation against the hash table entries and ensure the image has not been tampered with since creation. This should match the chunk size in the secure header. Used only during image validation.

#### Image header, byte count: cbSize

```
#define FFU_SIGNATURE "ImageFlash"

typedef struct _IMAGE_HEADER
{
    DWORD cbSize;           // sizeof(IMAGE_HEADER)
    BYTE Signature[12];     // "ImageFlash"
    DWORD ManifestLength;  // in bytes
    DWORD dwChunkSize;     // Used only during image generation.
} IMAGE_HEADER;
```

#### Manifest data, byte count: ManifestLength

The manifest contains the description of the device layout and the payload included in the FFU.

#### Padding byte count: variable

After the manifest padding (blank space) is added to fill out to current chunk. This ensures that the data that follows begins on a chunk boundary.

## Store header region

### Store header

#### Store header, byte count: STORE\_HEADER\_V1\_0\_SIZE (248 bytes)

The store header contains metadata that describes the payload. This includes update type, validation size, data size, and versioning. Some information is redundant, but is included for convenience.

The store header contains the DWORD count/length fields that describe the validation & write descriptor sections. This allows those sections to be copied out and processed later.

In V1 FFU format, you should see only one store header. In V2 FFU format, you should expect to see a number of store headers, depending on the value defined by the NumOfStores struct.

#### Validation descriptor region

The validation descriptor region is a collection of VALIDATION\_ENTRY structs. There are dwValidateDescriptorCount of them, and the overall byte count of the region is dwValidateDescriptorLength.

#### Write descriptor region

The write descriptor region is a collection of BLOCK\_DATA\_ENTRY structs. There are dwWriteDescriptorCount of them, and the overall size in bytes of the region is dwWriteDescriptorLength.

#### MajorVersion, MinorVersion

Major and minor versions of the store header.

#### FullFlashMajorVersion, FullFlashMinorVersion

Major and minor versions of the full flash update file format.

The following table shows the version values for V1 and V2 ffu image formats.

	V1	V2
MajorVersion	1	2
MinorVersion	0	0
FullFlashMajorVersion	2	2
FullFlashMinorVersion	0	0

## Note

- The OEM should not flash the image to the device unless the version of the image matches these values.

NumOfStores (V2 only)

Number of stores and their payloads in this FFU.

StoreIndex (V2 only)

Current store index, starting from 1.

StorePayloadSize (V2 only)

Size of the store payload in bytes, excluding padding.

DevicePathLength (V2 only)

Size of the device path that follows, in characters, without including the terminating null character.

DevicePath (V2 only)

Actual device path that the store is targeted for. This should be the same as device path retrieved from UEFI protocol: DEVICE\_PATH\_TO\_TEXT\_PROTOCOL.ConvertDevicePathToText()

```
typedef struct _STORE_HEADER
{
    UINT32 dwUpdateType; // indicates partial or full flash
    UINT16 MajorVersion, MinorVersion; // used to validate struct
    UINT16 FullFlashMajorVersion, FullFlashMinorVersion; // FFU version, i.e. the image format
    char szPlatformId[192]; // string which indicates what device this FFU is intended to be written to
    UINT32 dwBlockSizeInBytes; // size of an image block in bytes - the device's actual sector size may differ
    UINT32 dwWriteDescriptorCount; // number of write descriptors to iterate through
    UINT32 dwWriteDescriptorLength; // total size of all the write descriptors, in bytes (included so they can
be read out up front and interpreted later)
    UINT32 dwValidateDescriptorCount; // number of validation descriptors to check
    UINT32 dwValidateDescriptorLength; // total size of all the validation descriptors, in bytes
    UINT32 dwInitialTableIndex; // block index in the payload of the initial (invalid) GPT
    UINT32 dwInitialTableCount; // count of blocks for the initial GPT, i.e. the GPT spans blockArray[idx..
(idx + count -1)]
    UINT32 dwFlashOnlyTableIndex; // first block index in the payload of the flash-only GPT (included so safe
flashing can be accomplished)
    UINT32 dwFlashOnlyTableCount; // count of blocks in the flash-only GPT
    UINT32 dwFinalTableIndex; // index in the table of the real GPT
    UINT32 dwFinalTableCount; // number of blocks in the real GPT
    UINT16 NumOfStores; // Total number of stores (V2 only)
    UINT16 StoreIndex; // Current store index, 1-based (V2 only)
    UINT64 StorePayloadSize; // Payload data only, excludes padding (V2 only)
    UINT16 DevicePathLength; // Length of the device path (V2 only)
    CHAR16 DevicePath[1]; // Device path has no NUL at then end (V2 only)
} STORE_HEADER;
```

## Validation Entries

**Validation entries**, element count: dwValidateDescriptorCount, byte count: dwValidateDescriptorLength

The validation section is used only for partial updates. It contains a set of VALIDATION\_ENTRY structs. Each validation entry contains a byte array and a range on disk to compare. If the data in the validation entry matches the data on disk, that validation entry is confirmed. If and only if all validation entries are confirmed, the partial update is safe to apply to the device.

**Validation entry**, byte count: variable

Each VALIDATION\_ENTRY struct describes a location on disk that whose data should match the byte array in that entry.

```
typedef struct _VALIDATION_ENTRY
{
    UINT32 dwSectorIndex;
    UINT32 dwSectorOffset;
    UINT32 dwByteCount;
    BYTE rgCompareData[1]; // size is dwByteCount
} VALIDATION_ENTRY;
```

## Block data entries

**Block data entries**, element count: dwWriteDescriptorCount, byte count: dwWriteDescriptorLength

The block data entries describe how to write the data to disk. It is possible to write a single area of the disk more than once or to write the same data to multiple places on the disk, allowing for a compressed payload. The write descriptor region is composed of BLOCK\_DATA\_ENTRY structs. Each entry has a size and a byte array, and an array of locations to write to disk.

The fields **dwFlashOnlyTableIndex** and **dwFlashOnlyTableCount** are used to determine the last block entry that is necessary to lay out all of the partitions required to re-flash the device. If all of the blocks up to and including this block are successfully flashed to the device, the device can be re-flashed without requiring any silicon vendor or OEM code.

**Block data entry**, byte count: variable

Each block data entry describes a block of data in the store data section. Each entry describes the number of data blocks and the locations to which they should be written on disk. The accessMethod is used like the accessMethod in SetFilePointer, meaning that it gives meaning to the blockIndex.

```
enum DISK_ACCESS_METHOD
{
    DISK_BEGIN = 0,
    DISK_END   = 2
};

typedef struct _DISK_LOCATION
{
    UINT32 dwDiskAccessMethod;
    UINT32 dwBlockIndex;
} DISK_LOCATION;

typedef struct _BLOCK_DATA_ENTRY
{
    UINT32 dwLocationCount;
    UINT32 dwBlockCount;
    DISK_LOCATION rgDiskLocations[1];
} BLOCK_DATA_ENTRY;
```

## **Padding**

**Padding** – to allow the next section to start on a block boundary, byte count: variable

## Image payload region

The payload of block data to be written to disk, byte count: variable.

This is an array of data blocks. Each data block consists of **BytesPerBlock** bytes, where **BytesPerBlock** is defined in the store header.

In V1 FFU format, you should see only one image payload region. in V2 FFU format, you should expect to see a number of image payload regions depending on the value defined by the NumOfStores struct.

## Related topics

[Flashing tools](#)

# Implementing image integrity validation in custom flashing tools

1/25/2019 • 2 minutes to read

The FFU image contains a signed catalog file, a hash within the catalog, and a table of hashes corresponding to each chunk of the image. The hash table contents are generated using the SHA256 secure hash algorithm. Three checks must be performed before the image is flashed:

- **Catalog signature validation** - Validating the signature of the signed catalog file helps to verify that the image came from a trusted source.
- **Hash of the table of hashes validation** - Validating the hash of the table of hashes in the table helps to verify that the image has not been tampered with.
- **Data chunk validation using the hash table entries** - The FFU application must check each chunk against its corresponding chunk hash before flashing the image to the device.

## Checking the signature on the catalog and checking the hash of the table of hashes

The goal in signature validation is to make sure that the signature in the catalog matches the PK certificate on the phone. This approach allows checking for a signature up front without having the full image on the device before flashing. The signature check assumes that catalog contains a SHA1 hash.

Microsoft provides a UEFI protocol which exposes a function for this purpose, `EFI_CHECK_SIG_AND_HASH`. For more information, see [UEFI check signature protocol](#). This function also validates the hash of the table of hashes.

### Example code flow

1. Establish pointers to catalog and hash table data.
2. Determine the size of the catalog and hash table data in bytes.
3. Use the [UEFI check signature protocol](#) to call `EFI_CHECK_SIG_AND_HASH`, passing the pointers and data sizes.
4. Based on the EFI return code either continue to process the image, or post a security message such as `EFI_SECURITY_VIOLATION`.

**Note** If secure boot is not enabled on the device, a signature check is not performed.

## Checking the data against the hash table entries

The OEM flashing tool must check the data against the hash table entries. For info about the flashing tool, [Developing custom OEM flashing tools](#).

### Example code flow

A number of valid approaches can be used; an example is provided here to serve as a common point of reference.

1. Get the new hash data from the hash table in the image header.
2. Set up a loop to process chunks of data in the image.

3. Get a pointer to the hash of the current chunk of data.
4. Compare the hash of the current chunk of data against the hash table data using a function such as **memcmp**.
5. If the two hashes match, increment the pointer and get ready to check the next chunk of data.
6. If the two hashes do not match, stop all processing of the image and post a security message such as **EFI\_SECURITY\_VIOLATION**.
7. Continue processing until there is no more data in the image to process.

For info on the FFU elements discussed here, see [FFU image format](#).

## Error handling

Standard error handling code techniques should be used. A few common situations to handle are listed here:

- Missing catalog data
- Insufficient resources
- Empty image

## Clean up and exit

Follow standard practice and clean up any created arrays or other objects before exiting the flashing code. The exit process should return the final **EFI\_STATUS** value. For example, if the image is valid, you can return a value of **EFI\_SUCCESS**.

## Encryption library

Locate and include an appropriate encryption library in the image to support hash validation, such as **EFI\_HASH\_PROTOCOL**.

## Related topics

[Developing custom OEM flashing tools](#)

# Field service scenarios

1/25/2019 • 3 minutes to read

Scenarios can help to identify security vulnerabilities in field service processes. Each scenario should be reviewed to verify that a secure solution has been implemented by the OEM.

Device refurbishing	<p>Mobile operators and OEMs can refurbish devices by using a variety of approaches. The first refurbishment scenario occurs when customers return phones to the mobile operator for whatever reason. In this case, the devices are typically shipped to regional OEM service centers where they are reflashed with the current OEM image.</p> <p>The second refurbishment scenario occurs when customers are experiencing problems with the operation of the device. The customer brings the device back to the mobile operator store, and the store associate runs basic diagnostics. The associate can attempt to reset the phone to factory settings or reflash the device OS by using flashing tools provided by Microsoft (FFU) or the OEM. If this fails, the store associate can decide to send the device to an OEM service center. Before the device leaves the control of the mobile operator and is returned to the OEM, some mechanism is used to remove all the customer data.</p> <p>The renewal of the device at the OEM service center is the third refurbishment scenario. The device can be reflashed using whatever tools the OEM service center is equipped with. Some OEMs will use OS-level flashing technology, meaning they cannot reflash the device if the modem-level boot loader or software is broken. Others will be able to reflash the modem and OS partitions.</p>
Phone troubleshooting	<p>Field test and diagnostics apps can be used by the mobile operator store, mobile operator service centers, and OEM service centers to perform specific tests on a phone in two different scenarios.</p> <p>The first troubleshooting scenario is when the customer is experiencing difficulties with the device and a diagnostic test is run to gather information on the reported issue.</p> <p>The second scenario is when the mobile operator or OEM service centers seek to establish quality status of a device. This scenario is the more general form of the previous troubleshooting scenario. This is because there could be more reasons than customer difficulties that would trigger the need for diagnostics and testing of retail phones. For example, to perform field quality measures, the test could be initiated to perform random-sample testing.</p>

Engineering flashing	The engineering staffs of silicon vendors and OEMs need access to flashing technologies during development of hardware and software for the phones. This scenario is typically supported by low-level technologies such as JTAG, UEFI-based flashing, or OEM-specific flashing technologies. Which technology is selected depends on issues being worked on, development processes, and the supporting tools. The granularity of flashing options varies; lower-level tools have more flexibility in choosing which partitions can be flashed.
Engineering diagnostics	The engineering staffs of silicon vendors and OEMs need access to diagnostics technologies during development of hardware and software for the phones. These scenarios are typically supported by technologies provided by the SV or OEM that will be disabled on retail devices. Some of the SV technologies provide a broad set of capabilities—including features that support reading and writing to flash memory—that must be disabled before the device is shipped.
Mobile operator trials	There may be a need to flash OS images to the device to support mobile operator trials.
Production manufacturing	The process of flashing phones during manufacturing varies per OEM. Some OEMs use gang programmers that can flash a number of units at a time; others use the flashing technology described in this documentation. For more info on the flashing process, see <a href="#">Flashing tools</a> .
Rework manufacturing	Some OEMs have a process in place to rework phones that fail manufacturing quality control. Because the device is completely assembled, it may not be possible to use a gang programmer; this rework can be accomplished by using tethered flashing technologies similar to the customer return refurbishment scenario.
Remanufacturing	There can be a need to repurpose existing phone inventory for a different region or mobile operator by replacing the existing OS with one customized for the new market.

## Related topics

[Manufacturing](#)

# Using a host PC to reboot a device to flashing mode and get version information

1/25/2019 • 6 minutes to read

When a Windows 10 Mobile device is connected to a host PC via a USB cable, you can perform the following tasks in an application that is running on the host PC. These tasks are useful in certain manufacturing or customer care scenarios.

- Reboot the device into flashing mode.
- Retrieve version information from the device.

The host app uses the Windows Portable Devices API to accomplish these tasks.

## Understanding the Windows Portable Devices API

The Windows Portable Devices (WPD) API is a COM-based API that enables computers to communicate with attached devices. To learn more about this API, refer to the following resources:

- [Windows Portable Devices](#): This section of the MSDN library provides architecture guidance and reference documentation for the Windows Portable Devices API.
- [Portable Devices COM API Sample](#): This sample demonstrates how to use the Windows Portable Devices API to perform a variety of tasks, including enumerating connected devices, reading properties of content on a connected device, and sending MTP commands to a device.
- [Portable Devices Services COM API Sample](#): This sample demonstrates how to use the Windows Portable Devices API to perform a variety of operations on device services, including enumerating services and service content.

## Discovering Windows 10 Mobile devices that are connected to the host computer

Before the host app can reboot a device to flashing mode or retrieve version information from the device, the host apps must discover all devices that are connected to the computer.

1. Create an [IPortableDeviceManager](#) object and use the [IPortableDeviceManager::GetDevices](#) function to get the collection of device IDs for all connected devices.
2. Enumerate through the collection of device IDs to determine which connected device is a Windows 10 Mobile. For each device ID, create an [IPortableDevice](#) object and then use the [IPortableDevice::Content](#) and [IPortableDeviceContent::Properties](#) functions to retrieve the value of the [WPD\\_DEVICE\\_MODEL\\_UNIQUE\\_ID](#) device property for the device. If the device is a Windows 10 Mobile device, the value of the [WPD\\_DEVICE\\_MODEL\\_UNIQUE\\_ID](#) property is a GUID with the following value.

```
0x59f12ea9, 0x53ce, 0x452d, 0x97, 0x11, 0xca, 0x4e, 0xea, 0xf1, 0x80, 0x89
```

## Examples and additional resources

The following code examples in the [Portable Devices COM API Sample](#) demonstrate the following related tasks:

- Creating an [IPortableDeviceManager](#) object: see the `chooseDevice` function in `DeviceEnumeration.cpp`.

- Getting the device IDs of all connected devices: see the `EnumerateAllDevices` function in `DeviceEnumeration.cpp`.
- Retrieving device properties: see the `ReadContentProperties` function in `ContentProperties.cpp`.

For more info about enumerating devices and retrieving device properties, see the following articles:

- [Enumerating Devices](#)
- [Device Properties](#)
- [Retrieving Properties for a Single Object](#)

## Rebooting a Windows 10 Mobile device into flashing mode

After you have an `IPortableDevice` object that represents a Windows 10 Mobile device, you can send an MTP command to reboot the phone into flashing mode.

1. Create an `IPortableDeviceValues` object and configure it to set up the MTP command parameters:
  - a. Call `IPortableDeviceValues::SetGuidValue`. Pass `WPD_PROPERTY_COMMON_COMMAND_CATEGORY` to the `key` parameter and pass `WPD_COMMAND_MTP_EXT_EXECUTE_COMMAND_WITHOUT_DATA_PHASE.fmtid` to the `Value` parameter.
  - b. Call `IPortableDeviceValues::SetUnsignedIntegerValue`. Pass `WPD_PROPERTY_COMMON_COMMAND_ID` to the `key` parameter and pass `WPD_COMMAND_MTP_EXT_EXECUTE_COMMAND_WITHOUT_DATA_PHASE.pid` to the `Value` parameter.
  - c. Call `IPortableDeviceValues::SetUnsignedIntegerValue` again. Pass `WPD_PROPERTY_MTP_EXT_OPERATION_CODE` to the `key` parameter and pass the value **0x9401** to the `Value` parameter. This value represents the MTP command that reboots the phone into flashing mode.
  - d. Create an `IPortableDevicePropVariantCollection` object.
  - e. Call `IPortableDeviceValues::SetIPortableDevicePropVariantCollectionValue`. Pass `WPD_PROPERTY_MTP_EXT_OPERATION_PARAMS` to the `key` parameter and pass the `IPortableDevicePropVariantCollection` object to the `pValue` parameter.
2. Call `IPortableDevice::SendCommand` to send the MTP command. Pass the initialized `IPortableDeviceValues` object to the `pParameters` parameter. This operation sends the MTP command to reboot the phone into flashing mode.

### Examples and additional resources

The following code examples demonstrate how to set up MTP command parameters and send an MTP command:

- [Issuing the GetNumObjects Command](#): This topic in the MSDN library demonstrates how to send the standard GetNumObjects MTP command.
- The `SendhintsCommand` function in `ContentEnumeration.cpp` in the [Portable Devices COM API Sample](#).

For more information about sending MTP commands using the Windows Portable Devices API, see [Supporting MTP Extensions](#).

## Retrieving version information from a Windows 10 Mobile device

After you determine the device ID for a Windows 10 Mobile device that is connected to the host computer, you can

use a Windows 10 Mobile-specific device service named MtpDuDeviceService to retrieve version information from the device.

**Note** If the device is protected with a PIN, the MtpDuDeviceService is only available if the PIN has been entered and the device is unlocked.

## Open the MtpDuDeviceService

First, enumerate through the device services to open an [IPortableDeviceService](#) object for the MtpDuDeviceService.

1. Get an array of IDs for all device services supported by the device. To do this, cast the existing [IPortableDeviceManager](#) object to an [IPortableDeviceServiceManager](#) and call the [IPortableDeviceServiceManager::GetDeviceServices](#) method. Pass the device ID for the Windows 10 Mobile device to the *pszPnPDeviceID* parameter and the value **GUID\_DEVINTERFACE\_WPD\_SERVICE** to the *guidServiceCategory* parameter. The array of service IDs is returned in the *pServices* parameter.
2. Iterate through the array of service IDs, and perform the following tasks for each service ID:
  - a. Create an [IPortableDeviceService](#) object and call the [IPortableDeviceService::Open](#) function to open the service. Pass the current service ID to the *pszPnPServiceID* parameter.
  - b. Get the service object ID by calling the [IPortableDeviceService::GetServiceObjectID](#) function. You need the service object ID to access properties of the service.
  - c. Use the [IPortableDeviceService::Content](#) and [IPortableDeviceContent::Properties](#) functions to retrieve the collection of properties for the service (an [IPortableDeviceProperties](#) object).
  - d. Create an [IPortableDeviceKeyCollection](#) object and add the WPD-defined **WPD\_OBJECT\_NAME** property key to this collection. This property key indicates that you are retrieving the display name for the service.
  - e. Call the [IPortableDeviceProperties::GetValues](#) function to retrieve an [IPortableDeviceValues](#) object that contains the property values. Pass the service object ID to the *pszObjectID* parameter and the initialized [IPortableDeviceKeyCollection](#) object to the *pKeys* parameter.
  - f. Call the [IPortableDeviceValues::GetStringValue](#) function, and pass the WPD-defined **WPD\_OBJECT\_NAME** property key to the *key* parameter.
  - g. If the [IPortableDeviceValues::GetStringValue](#) function returns the string **MtpDuDeviceService**, you have found the service object you need to retrieve version information from the phone. Exit the loop and proceed to the next section.

If the name of the service is not **MtpDuDeviceService**, call the [IPortableDeviceService::Close](#) function, iterate to the next service ID returned by [IPortableDeviceServiceManager::GetDeviceServices](#), and return to step 2.

## Retrieve version information from the device

After you have opened an [IPortableDeviceService](#) object for the MtpDuDeviceService, you can use this object to retrieve version information from the device

1. Use the [IPortableDeviceService::Content](#) and [IPortableDeviceContent::Properties](#) functions to retrieve the collection of properties for the service (an [IPortableDeviceProperties](#) object).
2. Create an [IPortableDeviceKeyCollection](#) object and add a **PROPERTYKEY** value to this collection that specifies the version data you want to retrieve from the device. The **PROPERTYKEY** value must have the following structure:
  - The *fmtid* field must have the following GUID value:

0x9BFC64C1, 0x19C9, 0x4F3D, 0xA1, 0x4D, 0xC8, 0xDB, 0xE0, 0x47, 0x5D, 0x13

- The *pid* field must have one of the values shown in the table below

DEVICE DATA TO RETRIEVE	PID VALUE
The build string for the image on the device	Use the data returned by the following pid values to construct the build string: <ul style="list-style-type: none"><li>4 (the name of the internal Microsoft branch the OS was built from)</li><li>6 (the Windows build number)</li><li>7 (the Windows 10 Mobile build number)</li><li>8 (the time stamp for the build)</li></ul> An example build string is WPMAIN.9600.12186.20130906-1624.
The OS version	12
The OEM device name	15
The firmware version	16
The SoC version	17
The radio software version	18
The radio hardware version	19
The bootloader version	20
The platform ID (from SMBIOS).	29

- Call the [IPortableDeviceProperties::GetValues](#) function to retrieve an [IPortableDeviceValues](#) object that contains the requested version information. Pass the service object ID to the *pszObjectID* parameter and the initialized [IPortableDeviceKeyCollection](#) object to the *pKeys* parameter.
- Call the [IPortableDeviceValues::GetStringValue](#) function. For the *key* parameter, pass the same [PROPERTYKEY](#) value that you used earlier. This function returns the requested version information in the *pValue* parameter.

## Examples and additional resources

The following code examples in the [Portable Devices Services COM API Sample](#) demonstrate the following related tasks:

- Enumerating through device services: see the [EnumerateContactsServices](#) function in [ServiceEnumeration.cpp](#).

- Reading properties of a service: see the `ReadContentProperties` function in ContentProperties.cpp.

For more info about working with device services, see [Opening a service](#), [Accessing service object properties](#) and [Retrieving Object Properties](#).

# Disabling the initial setup process

1/25/2019 • 2 minutes to read

To disable the device's initial setup process (also sometimes called the *out-of-box experience* or *OOBE*) in a Test, Health, or Production image that is used during manufacturing, include the SKIPOOBE imaging feature in the OEMInput.xml file that is used to build the image.

The SKIPOOBE feature sets the **OobeHeadless** registry value (a REG\_DWORD value under the HKEY\_LOCAL\_MACHINE\Software\Microsoft\Shell\OOBE entry) to 1. Alternatively, you can configure this registry value directly in one of your own packages. The following example demonstrates a package XML file that sets this registry value.

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  Owner="Contoso"
  Component="Shell"
  SubComponent="DisableOOBE"
  OwnerType="OEM"
  ReleaseType="Production" xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00">

  <Macros>
    <Macro Id="hk1m.shell" Value="$(hk1m.microsoft)\Shell"/>
  </Macros>

  <Components>
    <OSComponent>
      <RegKeys>
        <RegKey KeyName="$(hk1m.shell)\OOBE">
          <RegValue Name="OobeHeadless" Type="REG_DWORD" Value="00000001" />
        </RegKey>
      </RegKeys>
    </OSComponent>
  </Components>
</Package>
```

## Related topics

[Specifying files and registry entries in a package project file](#)

# Reset protection

1/25/2019 • 9 minutes to read

Reset Protection helps you secure a device in case it is stolen. It must be enabled on the device during manufacturing time.

Reset Protection consists of the following parts:

- **Reset and reactivation protection** – The stolen device cannot be reused by resetting or flashing the device. When a user performs a factory reset on the device, they will be asked to enter the Microsoft Account credentials that are associated with that device. Additionally, if the device is flashed with a new image and Reset Protection is turned on, the Microsoft Account credentials that were associated with that device is required to finish OOBE and use the device.
- **Anti-rollback protection** – If Reset Protection is enabled, the stolen device cannot be flashed to an earlier version of the operating system that did not support Reset Protection.

To turn on Reset Protection, you must configure two secure UEFI variables:

- **ANTI\_THEFT\_ENABLED** –This variable needs to be set with a value that will be provided by Microsoft and indicates that the device can support Reset Protection. The operating system enables Reset Protection on the device based on this setting. This variable is in the 1A597235-6378-4910-9F8B-720FEE9357A3 namespace.
- **DBX** - This variable must contain the image hashes of the builds to which the device cannot be rolled back. These image hashes are provided by Microsoft. This variable is in the EFI\_IMAGE\_SECURITY\_DATABASE\_GUID namespace.

## Turn on Reset Protection in your images

There are two ways to turn on reset protection in your images:

### Option 1: Enable it by using oeminput.xml

On retail devices, you enable Reset Protection by adding the RESET\_PROTECTION feature to the OEMInput.xml file. When you include this feature, the device's UEFI secure boot keys for Reset Protection are provisioned as a scheduled task that will run once at first boot into main operating system and will not run again. For more info on the optional features that are available, see [Optional features for building images](#).

**Note** If you're building a test image, use RESET\_PROTECTION\_INTERNAL instead.

### Option 2: Enable when provisioning secure boot keys

Reset Protection is enabled on a device by provisioning UEFI secure boot keys and is a two-step process:

1. **Anti-Rollback provisioning** -- The DBX variable must be updated to contain the hashes for the builds that did not support Reset Protection. Specifically, the sample scripts that create PK, KEK, DB and DBX variables will be modified to add in the DBX variable the list of hashes provided by Microsoft. The list of hashes will be supplied by Microsoft in a file called **OEM\_RollbackHashes.bin**. The DBX variable must be signed with the OEM certificate.

The following excerpt includes the changes to the script that creates the DBX variable:

```
write-progress -activity "Making secure boot variables" -status "Creating DBX"
# add SHA256 hashes from the supplied file to the DBX variable
$hashes = Get-Content .\OEM_RollbackHashes.bin
format-sb-hashes "dbx" $ownerGuid $hashes
```

2. **Reset and Reactivation Protection provisioning** -- After setting the DBX variable, you must also set the ANTI\_THEFT\_ENABLED authenticated variable. The content of this variable will be provided by Microsoft in the **OEM\_ResetProtection\_Enable\_Resource.bin** file. The name of the variable is ANTI\_THEFT\_ENABLED and the namespace GUID is 1A597235-6378-4910-9F8B-720FEE9357A3. You can set this in the same way as the secure boot keys.

## How do I update a retail image with Reset Protection?

When you submit an update, Reset Protection should not be included as part of the update. However, when you build image, Reset Protection should be included in the oeminput.xml file. We recommend the following steps to update a retail image with Reset Protection enabled:

1. When you're developing an image, the RESET\_PROTECTION optional feature should be included in the oeminput.xml file.
2. Before you submit the packages for signing, you should remove the RESET\_PROTECTION optional feature from the oeminput.xml file.
3. After you receive the packages signed by Microsoft, you must add the RESET\_PROTECTION optional feature back to your oeminput.xml file.

## Bootable WIM files and MMOS

When using Reset Protection on a device, the MMOS or bootable WIM files that support this device must be built on a version of the tools that support Reset Protection. We recommend that the version of the device image and the version of MMOS match.

## Reverse logistics

With reverse logistics, you can get information about the status of Reset Protection on a device, such as the device IMEI, or check if Reset Protection is currently enabled on the device. You can also use this to remove Reset Protection if you have the appropriate recovery key for that device. Reverse logistics can help you in refurbishment scenarios where Reset Protection is turned on, but you don't have the Microsoft Account credentials that are required to turn it off. We've provided sample code on how to use reverse logistics in the [Portable Devices COM API Sample](#).

### How to get started using reverse logistics

In order to use Microsoft's automated reverse logistics program, organizations need to sign up for an account with the [Microsoft Dashboard](#) and perform the following tasks:

- Purchase an Authenticode signing certificate.
- Install the certificate on all machines that will be used to submit requests.
- Assign an administrator(s) to manage the program.
- For each user in your company who will contribute submissions to the Dashboard, add the Microsoft account for the user and grant each user the **Reverse Logistics** permission. To grant permissions, click **Your Profile** and then click **Permissions**.

### Register your company

Your company may already have an account with the Microsoft Dashboard. In that case, you will need to find the administrator of your account with the Dashboard. To find the administrator, click **Administration** and then click

**My Administrators.** We recommend adding a reverse logistics manager as an additional administrator so it's easier to approve users' reverse logistics requests. The administrators responsibilities include approving requests to join the company, approving request for permissions, and removing users after they leave the company. For more information, see [Manage users and permissions](#).

If your company does not yet have an account with the Dashboard, here is how to get started:

- [Get a code signing certificate](#)
  - In order to use reverse logistics, you **must** purchase a standard class 3 certificate, NOT an EV certificate.
  - Make sure you establish your company with the same name that you used to purchase the certificate. This is the name that will be exposed to users.
- [Establish a company](#)

Make sure you save this certificate and that it is accessible. You will need to install it on multiple computers later in this section. We recommend that you save a copy of the certificate on a thumb drive, or something easily accessible.

### Add users for your company

After you register your company, add other users who need reverse logistics permission:

- The first person to register a company becomes an administrator for that company account.
- Subsequent users need to register by using a Microsoft account. On the top right-hand corner of the [Dashboard](#), click **Register** to add yourself to your company and request the **Reverse Logistics** permission under **Additional Permissions Request**.
- The administrator receives notification and approves the request.

For more information about signing in to the Dashboard, see [Before you sign in](#).

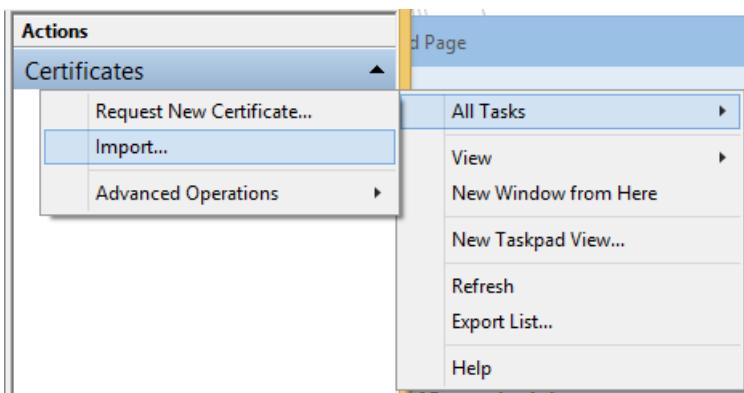
### Set up your workstation for reverse logistics

#### Prerequisites

- You need a workstation that runs Windows 7 or later and has browser access to the internet.
- Each reverse logistics submitter must have his or her own Microsoft account; account credentials should not be shared amongst multiple people.
- Only computers that have the certificate installed locally will be able to perform reverse logistics.

#### Process

1. Plug in the thumb drive that contains the certificate you purchased.
2. On each computer where you plan to submit reverse logistics requests, sign in as a local Administrator and install the code signing certificate:
  - a. Open a command prompt.
  - b. Type `mmc` and press ENTER.
  - c. On the **File** menu, click **Add/Remove Snap-in**.
  - d. Click **Add**.
  - e. In the **Add Standalone Snap-in** dialog box, select **Certificates**.
  - f. Click **Add**.
  - g. In the **Certificates Snap-in** dialog box, select **Computer account** and click **Next**.
  - h. In the **Select Computer** dialog box, click **Finish**.
  - i. On the **Add/Remove Snap-in** dialog box, click **OK**.
  - j. In the **Console Root** window, click **Certificates (Local Computer)** to view the certificate stores for the computer.
  - k. In the **Actions** pane, under Certificates, select **More Actions**, then **All Tasks**, and then **Import**:

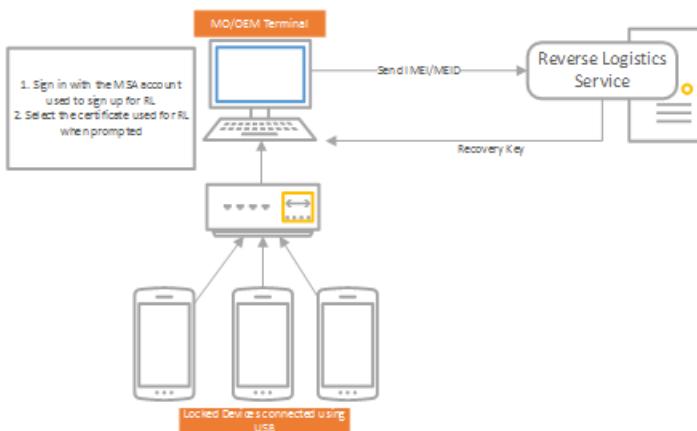


- I. Click **Browse** and find the certificate you purchased.
- m. Click **OK**. The certificate should be installed in your **Personal** certificate store.

## Authentication and Use

The next step is to create a client tool on a provisioned workstation to submit reverse logistics requests. You will need to create a third-party app with Microsoft account. The app will use a browser to allow a user to enter credentials using a Microsoft account website. That will grant access to your tool to get the appropriate token to call the Reverse Logistics API. For more information about how to build the app, see [Mobile and Windows desktop apps](#), and use "dds.reverse\_logistics" scope (instead of "wl.basic") to get the appropriate token.

After your tool has the token, it can call the Reverse Logistics API with that token, your client certificate, and the target IMEI in order to retrieve the recovery key for the target device.



## API specification

### Request

Reverse Logistics API endpoint:

POST

<https://cs.dds.microsoft.com/Command/ExternalClientCert/AdministrativeUnprotect/%7BPartnerName%7D/%7BDeviceId%7D>

{PartnerName} should be replaced with an end-user readable string that will be included in an email to the user whose Microsoft account is protecting the phone.

{DeviceId} should be replaced with a string in one of the following formats (leaving the square brackets and replacing the text inside and including the curly braces):

- ImeiOrMeid[{IMEI or MEID of the device}]
- Duid[{DUID of the device}]

Include the Microsoft account user token in the "Authorization" header of the request.

The certificate provisioned with the Dashboard for your organization must be used as the client certificate for

mutual HTTPS.

## Response

```
{  
    "UnprotectResult": "{UnprotectResult}"  
    "RecoveryKey": "{RecoveryKey}"  
}  
  
UnprotectResult will be a string value of the enum specified below:  
  
/// <summary>  
/// Result of the unprotect operation  
/// </summary>  
public enum UnprotectResult  
{  
    /// <summary>  
    /// Device was not found in DDS  
    /// </summary>  
    DeviceNotFound,  
  
    /// <summary>  
    /// Device was already unprotected  
    /// </summary>  
    DeviceAlreadyUnprotected,  
  
    /// <summary>  
    /// Device has been unprotected  
    /// </summary>  
    DeviceUnprotected,  
  
    /// <summary>  
    /// IF we find more than 1 device, we don't currently have a way to resolve the conflict. So, we  
don't unprotect.  
    /// </summary>  
    MultipleDevicesFound,  
}
```

Response codes:

- 200: Success.
- 400: The request is malformed.
- 401: The request is unauthorized. Your organization may not be provisioned properly, the user may not be provisioned with the organization, or there may be a problem or mismatch with the client certificate or the Microsoft account user token. The response may include text giving a reason for the authorization problem.
- 404: The API path or device specified was not found.
- 500: An unexpected error. If this persists, contact Microsoft for resolution of the issue.
- 503: Storage error. If this persists, contact Microsoft for resolution of the issue.

# Building and flashing mobile images

1/25/2019 • 2 minutes to read

Here's information to help you build and flash images to your Windows 10 Mobile device.

## 1. Define, customize, and build an image

To build a customized image that can be flashed to mobile devices, you can use either the new Windows Imaging and Configuration Designer (ICD) or classic ImgGen.cmd, or a hybrid of these methods using the Windows ICD command-line interface. See [Customizations for Windows 10 Mobile](#) to help you decide which method you need to use to meet your device customization needs.

- **Using Windows ICD:** This tool walks you through the process of creating images for Windows 10 Mobile using the Windows provisioning framework. Use this tool if you are building a device that is based on a reference design from a SoC vendor. To learn more about the tool, see [Windows Imaging and Configuration Designer](#). To start building a mobile image, see [Build a mobile image using Windows ICD](#).
- **Using ImgGen.cmd:** If you're building a device based on your own hardware design or using MCSF customization answer files and other assets that were generated using the legacy tools that shipped in Windows Phone 8.1, you can generate a customized mobile image using this tool. To get started, see [Building a mobile image using ImgGen.cmd](#).
- **Using a hybrid method:** If you want to use an OEMInput.xml file to fully define the contents of your image, take advantage of all the new runtime settings, enterprise policies, and enrollment settings available in Windows provisioning and also be able to fully customize the device hardware and connectivity settings, preload apps, and add assets such as ringtones and localized strings through MCSF, you can use a hybrid approach using the Windows ICD CLI to build your image. To learn more, see [Build a mobile image using a hybrid method](#).

## 2. Sign an image

No matter which method you used to customize and build your image, you'll need to cryptographically sign your image before you can deploy it to a device.

- [Sign a full flash update \(FFU\) image](#)

## 3. Flash an image to a device

Use the information in the following topics to learn about flashing and update tools:

- [Use the flashing tools provided by Microsoft](#)
- [Update packages on a device and get package update logs](#)
- [Update packages in an .FFU image file](#)

# Build a mobile image using Windows ICD

1/25/2019 • 5 minutes to read

Windows Imaging and Configuration Designer (ICD) is a Windows provisioning tool that lets you streamline customizing and provisioning a Windows image. It offers both a GUI or command-line interface that you can use to:

- Configure settings and policies for a mobile image, including new customizations primarily focused on enterprise and education-specific scenarios such as bulk enrollment and enterprise policies, and then build the customized image.
- Create a multivariant image by creating a provisioning package that defines targets and adds conditions that specify when settings for a variant will be applied, and then using the provisioning package as input to building a mobile image.

Use Windows ICD if you are building a device that is based on a reference design from a SoC vendor or if you are looking for a simple and streamlined process for creating a mobile image.

## To build a mobile image using the Windows ICD UI

The Windows ICD user interface (UI) provides an easy-to-use interface to build a customized mobile image. The UI shows all the settings that you can configure for a single variant mobile image and it then guides you through a step-by-step process to build, and even flash, the customized image.

However, be aware that there are some settings that are not available through the Windows provisioning framework and so these are not available to configure using Windows ICD. This includes [MCSF settings not supported in Windows Provisioning](#) as well as the support for data assets (such as ringtones, localized strings, and so on).

If you don't need to configure these settings or assets for your image, you can use the Windows ICD UI to build your customized, single variant mobile image by following the instructions in [Build and deploy an image for Windows 10 Mobile](#).

## To build a mobile image using the Windows ICD CLI

If you want to configure the available mobile settings in the Windows provisioning framework but want more flexibility in building your image, you can use the Windows ICD command-line interface (CLI). Using the Windows ICD CLI, you can:

- Choose how you define the packages and features contained in your image—either by using an OEMInput.xml file or a BSP.config.xml file as one of the inputs.
- Build a single variant mobile image.
- Create a provisioning package with multivariant settings, which you can use as one of the inputs for creating a multivariant mobile image.

Whether you're building a multivariant or single variant image, you must choose how you want to define the contents of the image. This determines the packages or features that will be part of the image and can include hardware support, languages, market-specific apps or functionality, and so on.

### Define the contents of the image

- Using a BSP.config.xml file.** You can download these as part of the BSP kit or you can generate your own BSP.config.xml file by running the BSP kit configuration tools from the SoC vendor and selecting your

component drivers.

The BSP.config.xml file is required if you tried or want to use the Windows ICD UI to build your image. If you are building a mobile image for a hardware reference design, you should already have the BSP.config.xml file for your hardware reference design.

2. **Using an OEMInput.xml file.** An OEMInput.xml file describes the required and optional elements that are used to define the mobile image. If you are creating a mobile image for a hardware reference design or for your own hardware design, and you also want to add your own features as part of the image, the OEMInput.xml file allows you to do this.

The mobile kit includes OEMInput.xml samples that you can use as a starting point. You can find these in the Windows install folder, C:\Program Files (x86)\Windows Kits\10\OEMInputSamples on x64 host computers or C:\Program Files\Windows Kits\10\OEMInputSamples on x86 host computers.

To learn more about OEMInput contents and other features you can add to your image, see [OEMInput file contents](#) and [Optional features for building images](#). If you are already familiar with OEMInput.xml and want to know how to add your own feature into the image and other functionality available to you, see the other topics under [Define the image using OEMInput and feature manifest files](#).

To build a mobile image, you must also have a provisioning package or Windows provisioning answer file to use as inputs. These files specify the customization settings and assets that you want to include in your image.

- You can use the Windows ICD UI to generate a provisioning package by configuring the settings and then exporting a provisioning package.
- You can use the Windows ICD UI to quickly generate a Windows provisioning answer file. Whenever you start a new project using the UI, Windows ICD always creates a customizations.xml in your project folder. This is typically found in your Documents\Windows Imaging and Configuration Designer (WICD)\Project\_Name folder. If you want to create one from scratch, see [Windows provisioning answer file](#) to understand the schema and then see [Windows Provisioning settings reference](#) to learn about the settings available for you to configure.

**Note** We recommend using the Windows ICD UI to easily generate either the provisioning package or the Windows provisioning answer file.

Follow these instructions to build a single variant mobile image:

### **Build a single variant mobile image**

1. See [Getting started with Windows ICD](#) and follow the instructions on launching the Windows ICD CLI.
2. [Build an image for Windows 10 Mobile](#) using the Windows ICD CLI.

Windows supports a mechanism that allows you to create a single image that can work for multiple markets. You can dynamically configure languages, branding, apps, and settings based on conditions. Building a mobile image that contains multivariant settings is only possible through the Windows ICD CLI.

The process for building a multivariant mobile image is similar to building a single variant mobile image except that you must create a provisioning package with multivariant settings first and then you must use this package as one of the inputs when you are ready to build the image.

Follow these instructions to build multivariant mobile image:

### **Build a multivariant mobile image**

1. Create the provisioning package. For more information, see [Create a provisioning package with multivariant settings](#).

You will use this package as one of the inputs for the next step.

2. Build an image for Windows 10 Mobile using the Windows ICD CLI.

## Related topics

[Building and flashing mobile images](#)

[Build a mobile image using a hybrid method](#)

# Build a mobile image using ImgGen.cmd

1/25/2019 • 17 minutes to read

You can use ImgGen.cmd to generate an image for a Windows 10 Mobile device that is based on your own hardware design or if you're using MCSF customization answer files and other assets that were generated using the legacy tools that shipped in Windows Phone 8.1.

Here's the high-level view of the steps you'll take to build an image using ImgGen.cmd:

1. Identify the packages to include in the image. For more info, see [Getting packages for the image](#).
2. Reference the packages by adding them to one or more feature manifest files, and save those files in the root directory for Microsoft packages, for example, %WPDKCONTENTROOT%\MS Packages. For more info, see [Specifying packages to include in images by using feature manifest files](#).
3. For each device platform, do the following:
  - a. Create a *device platform package*, and save it in the root directory for Microsoft packages. For more info, see [Set device platform information](#).
  - b. Reference the device platform package in a feature manifest file by using the **OEMDevicePlatformPackages** element.

You can include several device platforms in a feature manifest file. The OEMInput file will specify which device to use by its DeviceName.

Image creation will fail unless a valid device platform package is specified for the image.

4. Create an *OEMInput file* that specifies the device platform, the feature manifest files, and other attributes used to define the image. For more info, see [Creating an OEMInput file to define the image](#).
5. Create an MCSF customization answer file. At minimum, specify the required device platform information and the information required for the mobile operator network. For more info, see [Customization answer file](#) and [Phone metadata in DeviceTargetingInfo](#).

**Note** If you want to support multivariant settings in the answer file, the same set of conditions are supported in MCSF as in Windows provisioning. See the section *Target*, *TargetState*, *Condition* and *priorities* in [Create a provisioning package with multivariant settings](#) for a list of supported conditions but be sure to follow the schema for a MCSF customization answer file when you specify your **Targets** in the answer file.

6. Run the ImgGen.cmd to build the image. For more info, see [Using ImgGen.cmd to generate the image](#) below.
7. Sign the image so that it can be flashed to a device. For more info, see [Sign a full flash update \(FFU\) image](#).

## Getting packages for the image

Before building an image, you must first identify all the packages you need for the image. Generally, there are three categories of packages that are used to build an image:

- **Microsoft packages.** These include all OS and Microsoft-implemented driver packages required for any image, as well as additional platform-specific driver and firmware packages (for example, packages for components specific to a particular device resolution or SoC). These packages are included with the

MobileOS and must be installed under %WPDKCONTENTROOT%\MS Packages. You can use the following command to display the current value of WPDKCONTENTROOT.

```
ECHO %WPDKCONTENTROOT%
```

Feature manifest files abstract the location and groupings of packages that Microsoft provides. This allows for the specification of a group of related packages by specifying a single feature name. For example, specifying the TESTINFRASTRUCTURE feature includes multiple packages that support test execution. For more info, see [Optional features for building images](#).

- **SoC vendor packages.** These include packages for drivers and firmware components implemented by the SoC vendor. For more info about these packages, refer to documentation provided by the SoC vendor.

#### Note

Several UEFI packages are required from the SoC vendor to create bootable images. These packages vary depending on the layout of the device and the SoC vendor. Most of the packages populate binary partitions on the device. For more info about creating binary partition packages to populate these partitions, see [Specifying components in a package project file](#). The EFI system partition (ESP) is populated using Microsoft content and OEM content.

- **OEM packages.** These are packages created by OEMs for content such as drivers and applications. For info about creating packages, see [Creating packages](#).

## Creating an OEMInput file to define the image

To define the image, you must create an *OEMInput* file. This is an XML file that specifies the following:

- The type of image to generate. For example, you specify whether the image contains only Microsoft production packages or a mixture of production and test packages in the **ReleaseType** element, and you specify what screen resolution the image will support in the **Resolution** element.

#### Note

OEMs have some control over what Microsoft packages are included in the image by choosing different values for the **ReleaseType** element or by referencing Microsoft-defined features under the **Features** element. For more info, see [Specifying packages to include in images by using feature manifest files](#) later in this topic.

- The OEM packages to include in the image. To specify which OEM packages are included in the image, create a *feature manifest* file and reference this in the OEMInput file. For more info, see [Specifying packages to include in images by using feature manifest files](#) later in this topic.

For a full list of the supported elements in the OEMInput file, see [OEMInput file contents](#).

OEMs should use sample OEMInput files included with the MobileOS as the starting point for their own images. These sample files provide the starting configuration for a set of standard image types, including retail, production, test, and manufacturing images. OEMs should extend each file with the packages that contain the drivers, applications, and other components needed for their specific device. These sample files are available under %WPDKCONTENTROOT%\OEMInputSamples. For guidance about how maintain an OEMInput file so that the latest changes to the Microsoft sample files can be integrated into it, see [Configuring the OEMInput file to integrate feature changes from the Microsoft samples](#) later in this topic.

OEMs should contact the SoC vendor for the feature manifest files that are used for a specific device and include these in the OEMInput file.

#### Note

The OEMInput XML file supports explicit paths and environment variables. If you use environment variables in

paths to packages and other files, the environment variables will be expanded when the OEMInput XML file is processed by the imaging tool. The sample files included with the MobileOS use the %WPKCONTENTROOT% environment variable in some of the paths.

## Image types

The following table lists the types of images OEMs can build and the OEMInput sample to use as the starting configuration for each image type.

IMAGE TYPE	DESCRIPTION	OEMINPUT SAMPLE
Retail	<p>Retail images are the images that are flashed to final retail phones. Retail images must use Microsoft-signed packages that are returned to OEMs after submitting production images to Microsoft by using the OEM submission tool. For more info, see <a href="#">Submit binaries to be retail signed</a>.</p> <p>Retail images include the following:</p> <ul style="list-style-type: none"> <li>● Production version of core Windows components included in Windows 10 Mobile</li> <li>● Production Windows 10 Mobile components.</li> </ul>	RetailOEMInput.xml
Production	<p>Production images are similar to final retail images, but they have test signing enabled to run OEM-signed components as well as production-signed components, and they may contain test-related packages as well as production packages. Production images can be used for engineering work as well as mobile operator trials and other certification processes. Production images are submitted to Microsoft by using the OEM submission tool to be production signed by Microsoft before generating the final retail image. For more info, see <a href="#">Submit binaries to be retail signed</a>.</p> <p>Production images include the following:</p> <ul style="list-style-type: none"> <li>● Production version of core Windows components included in Windows 10 Mobile.</li> <li>● Production Windows 10 Mobile components.</li> <li>● Test signing enabled.</li> </ul>	ProductionOEMInput.xml

IMAGE TYPE	DESCRIPTION	OEMINPUT SAMPLE
Test	<p>Test images can be run in offsite test labs to test the functionality of the OS and drivers on a device. Test images include the following:</p> <ul style="list-style-type: none"> <li>• Test version of core Windows components included in Windows 10 Mobile.</li> <li>• Production Windows 10 Mobile components.</li> <li>• Test signing enabled.</li> <li>• Test applications, drivers, and other components to use for testing the OS in different conditions.</li> </ul> <div style="border: 1px solid black; padding: 5px;"> <p><b>Note</b></p> <p>To generate an image that includes OS tools such as ipconfig.exe, kill.exe, ping.exe, minshutdown.exe, reg.exe, tracelog.exe, sc.exe, and tlist.exe, build a test image.</p> </div>	TestOEMInput.xml
Health	Health images to be run in offsite test labs to test the power and performance capabilities of the device. Health images are similar to production images, with the addition of components for running tests related to power and performance.	HealthOEMInput.xml
Manufacturing	Manufacturing images to be used in the manufacturing environment. For more info, see <a href="#">MMOS image definition</a> .	MfgOEMInput.xml
Customer care	Customer care images include MMOS for retail customer care scenarios. For more info, see <a href="#">MMOS image definition</a> .	CustomerCareOEMInput.xml

### OEMInput file example

The following example shows the contents of a sample ProductionOEMInput.xml file.

```

<?xml version="1.0" encoding="utf-8" ?>
<OEMInput xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate">
  <Description>Test FFU generation for {SOC TYPE} with build number XXXXX</Description>
  <SOC>{PROCESSOR_NAME}</SOC>
  <SV>{SV_NAME}</SV>
  <Device>{DEVICE_NAME}</Device>
  <ReleaseType>Test</ReleaseType>
  <BuildType>fre</BuildType>
  <SupportedLanguages>
    <UserInterface>
      <Language>en-US</Language>
    </UserInterface>
    <Keyboard>
      <Language>en-US</Language>
    </Keyboard>
    <Speech>
      <Language>en-US</Language>
    </Speech>
  </SupportedLanguages>
  <BootUILanguage>en-US</BootUILanguage>
  <BootLocale>en-US</BootLocale>
  <Resolutions>
    <Resolution>480x800</Resolution>
  </Resolutions>
  <AdditionalFMs>
    <AdditionalFM>%WPDKCONTENTROOT%\FMFiles\MSOptionalFeatures.xml</AdditionalFM>
    <!-- Add OEM FM files here -->
  </AdditionalFMs>
  <Features>
    <Microsoft>
      <Feature>CODEINTEGRITY_TEST</Feature>
      <Feature>PRODUCTION_CORE</Feature>
      <Feature>BOOTKEYACTIONS_RETAIL</Feature>
    </Microsoft>
    <!-- Insert OEM\SOC features here
    <OEM>
      <Feature>xxx</Feature>
    </OEM>
    -->
  </Features>
</OEMInput>

```

## Specifying packages to include in images by using feature manifest files

Feature manifest files specify a set of features, packages, and apps that are available when building images. The OEMInput.xml file later selects the desired features from within these feature manifests to be included in the image.

In addition to a core set of Microsoft packages that are added to every image, there are additional packages that are added only for certain types of images. For example, a test image includes a different set of Microsoft packages than a production image.

The set of packages that are included in certain types of images is controlled by using feature manifest files. Feature manifests provide an extensible infrastructure for adding sets of optional packages to different types of image builds. The WDK includes a feature manifest named MSOptionalFeaturesFM.xml, and OEMs can also create their own. Feature manifests are referenced in the **AdditionalFMs** element of the OEMInput XML file. For more info about the contents of a feature manifest, see [Feature manifest file contents](#).

When the OEMInput XML file references a feature manifest, there are several ways that additional optional packages are included in the image:

- Any packages listed under the **BasePackages** element in the feature manifest file are automatically

included in the image.

- Other packages listed in the feature manifest file under elements such as **ReleasePackages** and **DeviceSpecificPackages** are included if the image definition matches the parameters specified by the elements in the feature manifest file. For example, if the feature manifest file lists a package under the **ReleasePackages** element where the release type is **Test**, the package is included only if the OEMInput file is configured to generate a test image.
- Any optional features defined in the feature manifest can be referenced in the OEM or Microsoft **Features** elements in the OEMInput file to add additional optional packages. A feature is a string that identifies one or more optional packages that can be included in an image. Usually, a feature identifies a set of packages that are associated with a specific component or feature area. For more info about optional features provided by Microsoft for use in OEMInput files, see [Optional features for building images](#).
- Depending on who will be working with the image, it may be appropriate to exclude prerelease features using the **ExcludePrereleaseFeatures** element. For more info, see [OEMInput file contents](#).

### Configuring the OEMInput file to integrate feature changes from the Microsoft samples

In any MobileOS release, the OEMInput samples provided by Microsoft may change from the previous release. For example, Microsoft-defined features may be added to a sample to incorporate new OS components that are available, or Microsoft-defined features may be removed from a sample if they are no longer supported for a certain image type. To ensure that OEMs are building the correct images for every MobileOS release, Microsoft recommends that OEMs adhere to the following process.

1. Structure the **AdditionalFMs** and feature-related elements in the OEMInput file as shown in the following example. In each section, include the feature manifests and features specified by the Microsoft sample first, and then include the OEM-specified feature manifests and features. Use comments to clearly separate each set of feature manifests and features, and to clearly explain why each OEM-specific feature is included in the file.

If the contents of the **AdditionalFMs** and **Features** elements in the OEMInput file deviate from the Microsoft samples, include detailed comments that explain why the changes were made. If any changes are introduced as a temporary workaround for an issue, the temporary workaround should be explained so that engineers using the file in the future can determine whether to keep the change or revert to the Microsoft sample when the OEM integrates a newer BSP or kit.

```
<!-- From Microsoft sample OEMInput.xml file -->
<AdditionalFMs>
    <AdditionalFM>%WDKCONTENTROOT%\FMFiles\MSOptionalFeatures.xml</AdditionalFM>
        <!-- Add OEM FM files here -->
</AdditionalFMs>

<Features>
    <Microsoft>
        <!-- Features from Microsoft OEMInput sample -->
        <!-- Additional OEM-selected features defined by Microsoft
            Include detailed comments about why each feature was chosen for this image -->
    </Microsoft>
    <OEM>
        <!-- Additional OEM-selected features defined by the OEM
            Include detailed comments about why each feature was chosen for this image -->
    </OEM>
</Features>
```

2. When the OEM integrates a new WDK and MobileOS release, compare the **Features** elements in the OEMInput sample from the latest release against the same elements in the OEMInput file created by the OEM.

3. Identify any changes to the Microsoft-specified features in the latest sample, and port the changes into the **Features** element in the OEMInput file created by the OEM.

## Using ImgGen.cmd to generate the image

ImgGen.cmd is a command file that runs the imaging tool (ImageApp.exe) with the appropriate parameters to create an FFU image. ImgGen.cmd also runs a utility application, DeviceNodeCleanup, after every run of ImageApp.exe. Running DeviceNodeCleanup helps ensure that the registry on the development computer remains clean and boot time does not increase. ImgGen.cmd and other related components are located in %WSDKCONTENTROOT%\Tools\bin\i386.

To use ImgGen.cmd to generate an image:

1. Configure your development computer as follows:

- Open a **Developer Command Prompt for VS2013** window (if you have installed Visual Studio 2013) or a **Command Prompt** window (if you have not installed Visual Studio 2013) as an administrator.
- Confirm that the TEMP environment variable refers to a directory that is not compressed or encrypted using the Encrypting File System (EFS) functionality. If the directory that the TEMP environment variable refers to does not meet these requirements and you do not want to modify the variable or the directory properties, you can alternatively create a BINARY\_ROOT environment variable and set it to an existing directory that also meets these requirements. If neither of these locations exist, or if they exist but are compressed or encrypted, the image cannot be generated and the ImageApp.exe tool will return an error.
- If you are running Windows 8.1, complete the additional steps to set the USN journal registry size to 1 Mb on the build PC.
  - a. Change the USN minimum size registry key by running the following command from an administrator command prompt:

```
reg add HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem /v NtfsAllowUsnMinSize1Mb /t REG_DWORD /d 1
```

- b. Reboot the PC before you build an image.

2. In the command prompt window, run ImgGen.cmd by using the following syntax. See the following section for more information about the parameters.

```
ImgGen.cmd outputFile OEMInputXML MSPackageRoot OEMCustomizationXML OEMCustomizationVer
```

### Command-line syntax for ImgGen.cmd

The following table describes the command line parameters for ImgGen.cmd.

ARGUMENT	DESCRIPTION
<i>OutputFile</i>	The name of the FFU file to be created. If you do not include a path, the FFU file will be created in the current directory. If you include a path, the specified path must already exist. FFU files use the ffu file extension.

ARGUMENT	DESCRIPTION
<i>OEMInputXML</i>	The name of the OEMInput xml file that defines the image to be created. If this file is not in the current directory, you must include the path to the file.
<i>MSPackageRoot</i>	The path to the root directory that contains the Microsoft packages. By default, this directory is %ProgramFiles(x86)%\Windows Kits\10\MS Packages (or the corresponding path under %ProgramFiles% on computers running a 32-bit version of Windows).
<i>OEMCustomizationXML</i>	The path to the OEM customization XML file. For more info about customization answer files, see <a href="#">Customization answer file</a>
<i>OEMCustomizationVer</i>	The version number that will be used for the generated OEM customization package.

## Usage samples

The following sample shows the basic use of ImgGen.cmd.

```
ImgGen Flash.ffu OEMInput.xml "%WPDKCONTENTROOT%\MSPackages" OEMCustomization.XML 8.1.0.1
```

## Output files

After an image is successfully generated in the folder specified by the *FFU\_file* command-line argument, the imaging tools copy the following files to the folder of the output FFU file:

- <FFUName>.ImageApp.log: This file contains all the logging information about creating the image. Any failures are reported in this file as well as the Console window.
- <FFUName>.UpdateOutput.xml: This file describes each of the packages applied to the image.
- <FFUName>.UpdateHistory.xml: This file contains a list of all the update and imaging events that have occurred on the image.
- <FFUName>.UpdateInput.xml: This file lists all of the packages that are included in the image.
- <FFUName>.PackageList.xml: This is a list of the package files contained in the image.
- <FFUName>.cat: This is a code signing catalog that can be used to sign the image. For more info see, [Sign a full flash update \(FFU\) image](#).

If a customization answer file is provided, this file is generated:

- <Owner>.<DeviceName>.Customizations.<Partition>.spkg

If any of the customizations include static applications, the following file will be generated.

- <Owner>.<DeviceName>.CustomizationsApps.spkg

## Generating customization packages without creating an image

To process a customization answer file without creating an image, use **CustomizationGen.cmd**.

CustomizationGen.cmd applies all of the customization rules and builds the customization packages. It skips the final step of building the ffu image. The syntax is similar to ImgGen.cmd.

For example, to process an answer file and just build customization packages, use this command.

```
CustomizationGen.cmd C:\OEMCustomization OEMInput.xml "%WPDKCONTENTROOT%\MSPackages" OEMCustomization.XML  
8.0.0.1
```

### Command-line syntax for CustomizationGen.cmd

The following table describes the command line parameters for CustomizationGen.cmd. All of the arguments are required.

ARGUMENT	DESCRIPTION
<i>OutputDirectory</i>	The path to the output directory for the OEM customization packages that will be created.
<i>OEMInputXML</i>	The name of the OEMInput xml file that defines the image to be created. If this file is not in the current directory, you must include the path to the file.
<i>MSPackageRoot</i>	The path to the root directory that contains the Microsoft packages. By default, this directory is %ProgramFiles(x86)%\Windows Kits\10\MSPackages (or the corresponding path under %ProgramFiles% on computers running a 32-bit version of Windows).
<i>OEMCustomizationXML</i>	The path to the OEM customization XML file.
<i>OEMCustomizationVer</i>	This is the version number that will be used for the generated OEM customization package.

## Large scale image generation recommendations

When images are repeatedly generated on a workstation, it is possible that virtual hard disk (VHD) manipulation errors can occur from time to time. If these errors occur, you can rerun the image generation process. All intermediate files should be automatically cleaned up by ImageApp. This section provides guidance for the configuration of PCs that will generate images in a large scale automated image generation environment to minimize the number of errors that may occur.

To generate a large volume of images, consider the following recommendations:

- Remove or disable the antivirus software on the image generation PC. The presence of antivirus software, and in particular file system filters often used to monitor activity, can have a major impact on the image generation process. At a minimum, virus scanning should be disabled on input and output directories and on all processes involved in the build, though this will typically not disable the file system filter of interest. Some vendors of antivirus software may offer settings to allow scanning activities to be delayed or scheduled to possibly lessen the impact on the image generation process. If additional safeguards are in place to isolate the system from viruses and other software risks, remove or disable the virus software on the image generation PC. Temporarily removing the virus software can help isolate the impact on the image generation process to determine if further investigation is warranted.

- Windows Server 2012 is used in the Microsoft labs and is recommended. The server should be configured for maximum file I/O performance.
- The image creation process should not be run on virtual machines (VMs), but on physical machines instead.
- All other services (such as print or HTTP services) that are unrelated to the file system should be disabled on the server.
- System activity traces may be able to help identify processes on build machines that are interacting with mounted VHDs. Any interactions that are not initiated by imaging tools should be avoided if possible, potentially by disabling services that take action on disk mount.
- The output drives on the PC should have write cache buffer flushing disabled. There is some risk in using this setting on a file server, but with image generation, this setting can be acceptable, as the imaging data can be re-created if a power loss occurs. If other data cannot be re-created on the server, use this setting with caution, because data can become corrupted.

## Related topics

[Building and flashing mobile images](#)

[OEMInput file contents](#)

[Sign a full flash update \(FFU\) image](#)

# Build a mobile image using a hybrid method

1/25/2019 • 2 minutes to read

You can take advantage of the benefits offered by both the Windows provisioning framework and MCSF by using a hybrid method to build your customized mobile image. This means that:

- You can use a MCSF [customization answer file](#) to fully customize the device hardware and connectivity settings, preload apps, add assets such as ringtones and localized strings, and configure any other [MCSF settings not supported in Windows provisioning](#).
- You can use a [Windows provisioning answer file](#) to define the new runtime settings, enterprise policies, enrollment settings, and configure any other [mobile settings supported only in Windows Provisioning](#).
- You can use the Windows Imaging and Configuration Designer (ICD) CLI to build your image.

**Note** Only the Windows ICD CLI allows you to use both a MCSF customization answer file and a Windows provisioning answer file to create a customized mobile image.

## To build a customized mobile image using a hybrid method

Here's the high-level steps you need to take to build a customized mobile image using the Windows ICD CLI:

1. Choose how you define the packages and features contained in your image.
  - You can use BSP.config.xml file - If you select this method, you should already have this as part of your BSP kit or you can generate your own using the configuration tools from the SoC vendor.
  - You can use an OEMInput.xml file and OEMDevicePlatform.xml to define your platform. To do this, follow steps 1-4 in the high-level list of steps in [Build a mobile image using ImgGen.cmd](#).
2. Create your answer files to define the settings that you want to configure for your image.
  - Create a MCSF [customization answer file](#) to customize any of the available customizations in the MCSF framework. For more information, see the *Customizations for <feature>* sections in [Customize using the mobile MCSF framework](#).
  - Create a [Windows provisioning answer file](#) to define any of the available settings in the Windows provisioning framework. For more information, see [Windows Provisioning settings reference](#).

If you are adding multivariant settings in both answer files, verify whether the multivariant rules in both answer files are consistent. See the section *Target*, *TargetState*, *Condition and priorities* in [Create a provisioning package with multivariant settings](#) for a list of supported conditions but be sure to follow the schema for the answer file you are creating when you specify your **Targets** within the answer file.

Also, make sure there are no duplicated settings in both answer files. You can use the [MCSF to Windows Provisioning settings map](#) to help you identify the settings that correspond to each framework.

3. Run the Windows ICD CLI to build the image. For more information, see [Build an image for Windows 10 Mobile](#).
4. Sign the image so that it can be flashed to a device. For more information, see [Sign a full flash update \(FFU\) image](#).

## Related topics

## Building and flashing mobile images

# Define the image using OEMInput and feature manifest files

1/25/2019 • 2 minutes to read

Learn how to create an OEMInput and feature manifest files to fully define the contents of your mobile image.

## In this section

TOPIC	DESCRIPTION
<a href="#">OEMInput file contents</a>	An OEMInput.xml file contains the required and optional elements used to define a mobile image. The OS uses this file to determine the applications processor, build type, UI languages, default region format, resolution, and other properties to include in the image that will be generated.  This topic provides a full listing of the XML schema for the file.
<a href="#">Optional features for building mobile images</a>	You can add optional features to images by including them under the <b>Features</b> element in the OEMInput XML file.
<a href="#">Feature manifest file contents</a>	<i>Feature manifest (FM) files</i> are used to define specific types of image builds that contain different sets of optional packages. This topic describes the required and optional elements in a FM file.
<a href="#">Create a feature and include it in an image</a>	This topic shows you how to create a feature and add it to an image.
<a href="#">Adding a driver to a test image</a>	This topic shows you how to create a feature and add it to a test image.
<a href="#">Feature groupings and constraints</a>	Feature groups and feature constraints allow additional logic to be added to the build system to support intelligent processing of the OEMInput XML.
<a href="#">Set device platform information</a>	Learn about the prerequisites for building an image that can be flashed to a mobile device, including additional device platform information such as partner names, version numbers, and device names, before the image is finalized for retail devices.

## Related topics

[Building and flashing mobile images](#)

# OEMInput file contents

1/25/2019 • 10 minutes to read

An OEMInput.xml file contains the required and optional elements used to define a mobile image. The OS uses this file to determine the applications processor, build type, UI languages, default region format, resolution, and other properties to include in the image that will be generated.

This topic provides a full listing of the XML schema for the file.

## Required elements in the OEMInput file

The following table describes the required elements in the OEMInput file.

ELEMENT	DESCRIPTION						
<b>OEMInput</b>	The root element for the OEMInput file.						
<b>Description</b>	A string that describes the image. OEMs should add a description that is particular to the device the image is built for.						
<b>SOC</b>	A string that identifies the SoC used on the device. The following values are currently supported: <table border="1"><thead><tr><th>APPLICATIONS PROCESSOR</th><th>SUPPORTED SOC VALUES</th></tr></thead><tbody><tr><td>QC8974</td><td><ul style="list-style-type: none"><li><b>QC8974:</b> Creates an image without a crash dump partition. Use this value for Production images.</li><li><b>QC8974_Test:</b> Creates an image with a crash dump partition that is more than 2 GB in size. Use this value when generating Test images for devices with more than 4 GB of storage.</li></ul></td></tr><tr><td>QC8x26</td><td><ul style="list-style-type: none"><li><b>QC8x26:</b> Creates an image without any crash dump partitions. Use this value for production images.</li></ul></td></tr></tbody></table>	APPLICATIONS PROCESSOR	SUPPORTED SOC VALUES	QC8974	<ul style="list-style-type: none"><li><b>QC8974:</b> Creates an image without a crash dump partition. Use this value for Production images.</li><li><b>QC8974_Test:</b> Creates an image with a crash dump partition that is more than 2 GB in size. Use this value when generating Test images for devices with more than 4 GB of storage.</li></ul>	QC8x26	<ul style="list-style-type: none"><li><b>QC8x26:</b> Creates an image without any crash dump partitions. Use this value for production images.</li></ul>
APPLICATIONS PROCESSOR	SUPPORTED SOC VALUES						
QC8974	<ul style="list-style-type: none"><li><b>QC8974:</b> Creates an image without a crash dump partition. Use this value for Production images.</li><li><b>QC8974_Test:</b> Creates an image with a crash dump partition that is more than 2 GB in size. Use this value when generating Test images for devices with more than 4 GB of storage.</li></ul>						
QC8x26	<ul style="list-style-type: none"><li><b>QC8x26:</b> Creates an image without any crash dump partitions. Use this value for production images.</li></ul>						

ELEMENT	DESCRIPTIONS	PROCESSOR	SUPPORTED SOC VALUES
	<ul style="list-style-type: none"> <li> <b>QC8x26_Test:</b>  <b>Creates an</b>            image with two dedicated crash dump partitions on eMMC. The combined size of the two partitions is 3.5 times the size of the total RAM memory on the phone. To create a test image that supports full dumps and offline dumps on a device with at least 8GB of eMMC, use this value and include the DEDICATEDUMP ONEMMC feature.         </li> <li> <b>QC8x26_UseSD_Test:</b>  <b>Creates an</b>            image without dedicated partitions for storing crash dumps on devices with less than 8 GB of eMMC. To create full, kernel, or offline crash dumps, an SD card must be present and the feature DEDICATEDUMP ONSD feature must be specified to redirect dumps to the SD card. The recommended SD card size for offline dumps is 16 or 32 GB. For more information about specifying debug features, see <a href="#">Optional features for building images</a>.         </li> </ul> <p>To build a QC8x26 test image that supports full dumps and offline dumps on a device with less than 8GB of eMMC, use QC8x26_UseSD_Test. Include the DEDICATEDUMPONSD</p>		

ELEMENT	DESCRIPTIONS PROCESSOR	
	<p>feature to redirect <b>SUPPORTED SOC VALUES</b> dumps to the SD card.</p> <p>The recommended SD card size for offline dumps is 16 or 32 GB.</p> <p>To build a QC8x26 test image that supports full dumps and offline dumps on a device with at least 8GB of eMMC, use <b>QC8x26_Test</b> and include the <b>DEDICATEDUMPONEM</b> MC feature.</p> <p>To build a QC8x26 retail or production image where full dumps and offline dumps are not enabled, use <b>QC8x26</b>.</p>	

ELEMENT	DESCRIPTIONS PROCESSOR	SUPPORTED SOC VALUES
	<b>QC8960</b> <ul style="list-style-type: none"> <li>● <b>QC8960:</b> Creates an image without a crash dump partition. Use this value for Production images.</li> <li>● <b>QC8960_Test:</b> Creates an image with a crash dump partition that is more than 2 GB in size. Use this value when generating Test images for phones with more than 4 GB of storage.</li> <li>● <b>QC8960_Test_4_gb:</b> Creates an image with a crash dump partition that is approximately 600 MB in size, large enough to store a single crash dump for a device with 512 MB of RAM. Use this value when generating Test images for phones with only 4 GB of storage.</li> </ul> <div data-bbox="1144 1365 1409 1691" style="border: 1px solid black; padding: 10px;"> <p><b>Important</b></p> <p>The 8960 SOC options must only be used for images that are used to create updates for Windows 10 Mobile. For more information see <a href="#">Update</a>.</p> </div>	
<b>Device</b>		A string that identifies the device model the image is being built for. Use this setting to include packages marked with the corresponding <b>Device</b> attribute of the <b>DeviceSpecificPackages</b> and the <b>OEMDevicePlatformPackages</b> elements in the feature manifest file.

ELEMENT	DESCRIPTION
<b>ReleaseType</b>	<p>A string that indicates the release type of the image. Use this setting to include packages marked with the corresponding <b>ReleaseType</b> attribute value of a <b>ReleasePackages</b> element in the feature manifest file. The following values are supported:</p> <ul style="list-style-type: none"> <li>• <b>Production:</b> This value indicates that all packages in the image are production signed, and the image only includes production packages (that is, packages where the <b>ReleaseType</b> attribute in the package XML is set to <b>Production</b>). In addition, all Microsoft-owned packages must be signed with a certificate issued by Microsoft. This value should only be used when generating the final retail image.</li> <li>• <b>Test:</b> This value indicates that the image can contain test-signed packages as well as production-signed packages, and the image contains a mixture of production and test packages (that is, packages where the <b>ReleaseType</b> attribute in the package XML is set to <b>Test</b> or <b>Production</b>). This value is used in production, test, health, and manufacturing images. For more information about the different image types, see <a href="#">Building a phone image using ImgGen.cmd</a>.</li> </ul>
<b>BuildType</b>	<p>A string that specifies whether to use checked build or free packages when creating the image. To generate a checked build with debug code, specify <b>chk</b>. To generate a free build without debug code, specify <b>fre</b>.</p>
<b>SupportedLanguages</b>	<p>Contains the following child elements that specify the language support in the generated image:</p> <ul style="list-style-type: none"> <li>• <b>UserInterface:</b> A string that contains the language codes for the display languages to include in the image. If multiple language codes are listed, they must each be included in their own <code>&lt;language&gt;</code> block.</li> <li>• <b>Keyboard:</b> A string that contains the language codes for the keyboard languages to include in the image (that is, languages in which text correction and suggestions are supported). If multiple language codes are listed, they must each be included in their own <code>&lt;language&gt;</code> block.</li> <li>• <b>Speech:</b> A string that contains the language codes for the speech languages to include in the image. If multiple language codes are listed, they must each be included in their own <code>&lt;language&gt;</code> block.</li> </ul>
<b>BootUILanguage</b>	<p>A string that specifies the language code for the default display language to include in the image.</p>

ELEMENT	DESCRIPTION
<b>BootLocale</b>	A string that specifies the language code for the default region format for the image.
<b>Resolutions</b>	<p>Contains one <b>Resolution</b> element. The <b>Resolution</b> element contains a string that represents a display resolution supported by the image. The following values are supported: <b>480x800</b>, <b>540x960</b>, <b>720x1280</b>, <b>768x1280</b>, and <b>1080x1920</b>. The build will include the corresponding resolution packages in the image for the specified resolution.</p> <p>Note that some resolutions are supported only with certain applications processors.</p>

## Optional elements in the OEMInput file

The following table describes the optional elements that OEMs can include in the OEMInput file.

ELEMENT	DESCRIPTION
<b>SV</b>	A string that identifies the manufacturer of the SoC used on the device. Use this setting to include packages marked with the corresponding <b>SVPackages</b> element in the feature manifest file.
<b>Product</b>	<p>A string that specifies what OS variant to build. Specify the value <b>Manufacturing OS</b> to build an MMOS (Microsoft Manufacturing OS) image, which includes the minimal set of OS packages required by MMOS. To build a full OS image, omit this element.</p> <p>For more information about building an MMOS image, see <a href="#">MMOS image definition</a>.</p>
<b>FormatDPP</b>	A string that indicates whether to include an empty formatted device provisioning partition (DPP) in the image. Specify the value <b>true</b> to generate an image that includes an empty formatted DPP. Specify the value <b>false</b> or omit this element to prevent the DPP from being overwritten.

ELEMENT	DESCRIPTION
<b>ExcludePrereleaseFeatures</b>	<p>A string that indicates if features considered to be prerelease are excluded from the build. Specify the value <b>true</b> to generate an image that excludes the prerelease features, specify <b>false</b> to include them. Builds that are created with <b>ExcludePrereleaseFeatures</b> set to <b>true</b>, may be referred to as "limited" builds. If this entry is not included in the OEMInput file, the prerelease features will be included in the image.</p> <p>When this option is set to <b>true</b> some replacement features may be present in the image. If there are no replacement features configured, then they will not be present in the build.</p>
<b>Features</b>	<p>Contains one or more child elements that specify the names of optional features to reference when building the image. Each feature corresponds to one or more packages that will be included in the image. In order to use a feature, the feature must be defined in a feature manifest that is listed under the <b>AdditionalFMs</b> element. For more information about features and feature manifests, see <a href="#">Building a phone image using ImgGen.cmd</a> and <a href="#">Feature manifest file contents</a>.</p> <p>The <b>Features</b> element contains one or more of the following child elements:</p> <ul style="list-style-type: none"> <li>• <b>Microsoft</b> Contains one or more <b>Feature</b> elements that specify the names of optional Microsoft features to include in the image. For more info about Microsoft features, see <a href="#">Optional features for building images</a>.</li> <li>• <b>OEM</b> Contains one or more <b>Feature</b> elements that specify the names of optional OEM features to include in the image. For more information about adding OEM features, see <a href="#">Feature manifest file contents</a>.</li> </ul>
<b>AdditionalFMs</b>	<p>Contains one or more <b>AdditionalFM</b> elements. Each <b>AdditionalFM</b> element contains a string value that specifies the full path of a feature manifest file to reference when building the image.</p> <p>Feature manifests define the set of packages that are automatically included in certain types of images, and they also define feature names that can be referenced under the <b>Features</b> element to include additional packages in the image. For more information about feature manifest files, see <a href="#">Feature manifest file contents</a>.</p>

ELEMENT	DESCRIPTION
<b>PackageFiles</b>	<p>Contains a set of <b>PackageFile</b> elements that specify additional packages to include in the image.</p> <div style="border: 1px solid black; padding: 10px;"> <p><b>Important</b></p> <p>The <b>PackageFiles</b> element can only be used in pre-retail images such as Test and Health images. It is only intended to be used in scenarios you need to quickly add an ad-hoc package to a pre-retail image. In retail images, all packages must be referenced using a feature that is listed under the <b>Features</b> element or listed in a feature manifest that is referenced under the <b>AdditionalFMs</b> element. For more information about features and feature manifests, see <a href="#">Building a phone image using ImgGen.cmd</a> and <a href="#">Feature manifest file contents</a>.</p> </div> <p>Each <b>PackageFile</b> element contains a text value that specifies the path and name of a single package. If no additional packages are being added to the image, the <b>PackageFiles</b> element must be omitted from the file. The packages can be in any location on the development computer. An environment variable can be used in the path to each package in the <b>PackageFile</b> element.</p>

## XML schema for the OEMInput file

The OEMInput file is validated against the following XML schema. You can use this schema to validate the OEMInput XML files that you create. To do this in Visual Studio, first save this in a file with an XSD extension. In Visual Studio select **XML > Schemas** and select the file that you created. Any deviations in your XML from the schema will be highlighted. Hover over the highlighted items to see additional information.

```
<?xml version="1.0" encoding="utf-8"?>
<xsschema targetNamespace="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate"
elementFormDefault="qualified"
xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate"
xmlns:mstns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xss:element name="OEMInput">
<xss:complexType>
<xss:all>
<xss:element name="Product" type="xs:string" minOccurs="0" maxOccurs="1"/>
<xss:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1" />
<xss:element name="SV" type="xs:string" minOccurs="0" maxOccurs="1"/>
<xss:element name="SOC" type="xs:string" minOccurs="1" maxOccurs="1"/>
<xss:element name="Device" type="xs:string" minOccurs="1" maxOccurs="1"/>

<xss:element name="ReleaseType" minOccurs="1" maxOccurs="1">
<xss:simpleType>
<xss:restriction base="xs:string">
<xss:enumeration value="Test"/>

```

```

        <xs:enumeration value="Production"/>
    </xs:restriction>
</xs:simpleType>
</xs:element>

<xs:element name="BuildType" minOccurs="1" maxOccurs="1">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="fre"/>
            <xs:enumeration value="chk"/>
            <xs:enumeration value="%BUILDTYPE%"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>

<xs:element name="FormatDPP" minOccurs="0" maxOccurs="1">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="true"/>
            <xs:enumeration value="false"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>

<xs:element name="ExcludePrereleaseFeatures" minOccurs="0" maxOccurs="1">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="true"/>
            <xs:enumeration value="false"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>

<xs:element name="OEMDevicePlatform" type="xs:string" minOccurs="0" maxOccurs="1"/>

<xs:element name="SupportedLanguages" minOccurs="1" maxOccurs="1">
    <xs:complexType>
        <xs:all>
            <xs:element name="UserInterface" minOccurs="1" maxOccurs="1">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Language" type="xs:string"
                            minOccurs="1" maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Keyboard" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Language" type="xs:string"
                            minOccurs="1" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Speech" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Language" type="xs:string"
                            minOccurs="1" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:all>
    </xs:complexType>
</xs:element>

<xs:element name="BootUILanguage" type="xs:string" minOccurs="1" maxOccurs="1"/>

<xs:element name="BootLocale" type="xs:string" minOccurs="1" maxOccurs="1"/>

```

```

<xs:element name="Resolutions" minOccurs="0" maxOccurs="1">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Resolution" type="xs:string" minOccurs="1"
                maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="Features" minOccurs="0" maxOccurs="1">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Microsoft" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Feature" type="xs:string"
                            minOccurs="1" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="OEM" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Feature" type="xs:string"
                            minOccurs="1" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="AdditionalFMs" minOccurs="0" maxOccurs="1">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="AdditionalFM" type="xs:string"
                minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<!--This element is only for use with Windows 8 Phone device update images -->
<xs:element name="UserStoreMapData" minOccurs="0" maxOccurs="1">
    <xs:complexType>
        <xs:attribute name="SourceDir" type="xs:string" />
        <xs:attribute name="UserStoreDir" type="xs:string" />
    </xs:complexType>
</xs:element>

<xs:element name="PackageFiles" minOccurs="0" maxOccurs="1">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="PackageFile" type="xs:string"
                minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:all>
</xs:complexType>
</xs:element>
</xs:schema>

```

## Related topics

[Building an image using ImgGen.cmd](#)

# Optional features for building mobile images

1/25/2019 • 17 minutes to read

You can add optional features to images by including them under the **Features** element in the OEMInput XML file. Adding a feature will add the packages associated with the feature to the image. Some features can only be used with certain types of images. For more information about using optional features, see [Building an image using ImgGen.cmd](#).

Updates should be tested by submitting OS update requests using the Ingestion Client and verifying successful OS updates.

## Conditional features

If a device meets the installation conditions listed for a feature in the following table, then the update path for the device will fail unless the feature is installed. For example, if there's an update for Rich Communications Suite Platform support for Windows 10 Mobile, and you removed that feature from a device that's otherwise capable of using it, then that entire update package (and all subsequent updates) will fail to install. To get updates again, you need to deliver an image that includes the feature to customers, and have them re-flash the device.

FEATURE NAME AND ID	UPDATE PATH	CONDITION
Rich Communications Suite Platform support (RCS_FEATURE_PACK)	Windows 8.1 to Windows 10	Install if PhoneManufacturer=NOKIA Update if installed
Nearby Numbers/Block and Filter (MS_COMMSENHANCEMENT* apps)	Windows 8.1 to Windows 10	Install MS_COMMSENHANCEMENTCHINA if HKEY_LOCAL_MACHINE\SYSTEM\Platform\ DeviceTargetingInfo\phoneromlanguage = 0804. Update if it's already installed.  Install MS_COMMSENHANCEMENTGLOBAL if HKEY_LOCAL_MACHINE\SYSTEM\Platform\ DeviceTargetingInfo\phoneromlanguage <> 0804. Update if it's already installed.

## Retail features defined by Microsoft

The following table describes the Microsoft-defined features that can be used by OEMs in the **Features** element in the OEMInput file for retail devices.

Refer to the Mobile Operator guides for any additional retail features that are used for specific mobile operators.

FEATURE	DESCRIPTION
OPTIMIZED_BOOT	Modifies the behavior of the OS boot process to start some system processes and services before all device drivers are started. Enabling this feature may decrease boot time, but it may also cause regressions in boot behavior in some scenarios.
STANDARD_FEATURE_1	This feature includes standard features that must be included in all images.
NETLOG_RETAIL	This feature adds network capture logging to assist in troubleshooting network connectivity issues. This feature is used to gather network diagnostic information by Field Medic.
NAVIGATIONBAR	This feature adds a phone setting that enables users to configure the color of the software buttons.

FEATURE	DESCRIPTION
FACEBOOK	This feature includes Facebook in the image.
SKYPE	This feature includes the Skype Windows Phone Silverlight app in the image.
BINGAPPS	This feature adds Bing News, Finance, Sports and Weather.
BINGFOOD	This feature adds Bing Food & Drink.
BINGHEALTH	This feature adds Bing Health & Fitness.
BINGTRAVEL	This feature adds Bing Travel.
WIFI_FEATURE_PACK	This feature removes all cellular-related functionality from the operating system and is intended only for devices that will not be connected to a cellular network. It removes all cellular related tiles, icons, and settings from the user interface. Including this feature reduces memory usage and improves the user experience by not displaying nonfunctional cellular settings and icons.
RELEASE_PRODUCTION	Deprecated, no longer in use.
COMMSENHANCEMENTGLOBAL	The feature includes message and call filtering application in image.
COMMSENHANCEMENTCHINA	The feature includes message&call filter, call location, and category application in the image. It's used for China only.
4GBFEATURESONDATA	On devices with 4GB of storage, provisioned packages are stored on the data partition.
8GBFEATURESONDATA	On devices with 8GB of storage, provisioned packages are stored on the data partition. <div data-bbox="822 1590 1440 1680" style="border: 1px solid black; padding: 5px;"><p><b>Note</b> We strongly recommend that you also use 8GBFEATURESONSYSTEM when you specify this feature.</p></div>
8GBFEATURESONSYSTEM	On devices with 8GB of storage, provisioned packages are stored on the OS partition.
16GBFEATURESONDATA	On devices with 16GB of storage, provisioned packages are stored on the data partition. <div data-bbox="822 1927 1440 2016" style="border: 1px solid black; padding: 5px;"><p><b>Note</b> We strongly recommend that you also use 16GBFEATURESONSYSTEM when you specify this feature.</p></div>

FEATURE	DESCRIPTION
16GBFEATURESONSYSTEM	On devices with 16GB of storage, provisioned packages are stored on the OS partition.
SPATIALSOUND_FILTERDATA	Contains all the data files and registry keys necessary to enable Spatial Audio functionality on a mobile device. Spatial audio is used to make audio sound like it is coming from a specific direction, and to make the audio sound like it exists naturally inside a specific type of room.
TAIWAN_ENABLEMENT	Remove Taiwan references from the image.
MYANMAR_ZAWGYI_ENABLEMENT	Enable this image to use the Zawgyi encoding in Myanmar. You should also add the Zawgyi keyboard when adding this feature.
SOCProdTest_HSTI	<p>This feature installs the correct driver for the security watermark checks and is required for Windows 10 Mobile. If this package is not installed, you will see the <b>Not For Retail</b> watermark.</p> <div data-bbox="822 788 1429 878" style="border: 1px solid black; padding: 5px;"> <p><b>Important</b> This feature must be included in your OS image. If this feature is not included, the device might not boot.</p> </div>
ATTWIFI	This feature installs a third-party plugin that allows devices with an AT&T SIM to automatically connect to AT&T hotspots.
DISABLE_ABNORMAL_RESET	This feature disables abnormal resets and does not report it to Watson.
Docking	This feature enables Continuum for Windows 10 Mobile.
RESET_PROTECTION	This feature enables Reset Protection on a retail image. When the device boots for the first time, the appropriate secure UEFI variable are written.
PERF_TRACING_TOOLS	Contains tools for doing performance analysis, such as tools for stopping and merging ETL tracing.
ENABLE_BOOT_PERF_BASIC_TRACING	Enables boot performance tracing events to be generated.
ENABLE_BOOT_PERF_CPU_PROFILING_TRACING	Enables CPU profiling events. in addition to the boot performance tracing events that are enabled with ENABLE_BOOT_PERF_BASIC_TRACING.
MESSAGINGGLOBAL	This feature installs the Messaging package that includes Skype integration. It must be used for any devices except those being submitted for NAL certification in China.
MESSAGINGLITE	This feature installs the Messaging package without Skype integration. It must be used for any phone or other device that is submitted for NAL certification in China (excluding Hong Kong SAR and Macau).
SPATIALSOUND_FILTERDATA	This feature installs <a href="#">Spatial sound</a> .

## Retail boot sequence settings

The next two settings control the phone start up. Only one can be selected.

BOOTSEQUENCE_RETAIL	This feature enables the standard retail boot sequence.
---------------------	---

## Microsoft internal retail features

The following components are reserved for Microsoft internal use only, but are documented here for completeness.

SELFHOST	This feature adds components used exclusively for self-hosting scenarios in Microsoft.
FieldMedicCustomProfiles	This feature adds additional tracing profiles used by Field Medic.
RESET_PROTECTION_INTERNAL	This feature enables Reset Protection on a test image. When the device boots for the first time, the appropriate secure UEFI variable are written.

## Retail Demo Experience features

The Retail Demo Experience FOD contains offline retail demo content that is critical to a great retail demo experience. This offline content contains Windows and Office content that is shown along with any OEM or Retailer retail demo content. Adding of this Retail Demo Experience FOD is required and necessary for any OEMs participating in enabling a Retail Demo Experience on their Windows 10 devices.

- MS\_RETAILDEMOCONTENT\_AR-SA
- MS\_RETAILDEMOCONTENT\_BG-BG
- MS\_RETAILDEMOCONTENT\_CS-CZ
- MS\_RETAILDEMOCONTENT\_DA-DK
- MS\_RETAILDEMOCONTENT\_DE-DE
- MS\_RETAILDEMOCONTENT\_EL-GR
- MS\_RETAILDEMOCONTENT\_EN-GB
- MS\_RETAILDEMOCONTENT\_EN-US
- MS\_RETAILDEMOCONTENT\_ES-ES
- MS\_RETAILDEMOCONTENT\_ES-MX
- MS\_RETAILDEMOCONTENT\_ET-EE
- MS\_RETAILDEMOCONTENT\_FI-FI
- MS\_RETAILDEMOCONTENT\_FR-CA
- MS\_RETAILDEMOCONTENT\_FR-FR
- MS\_RETAILDEMOCONTENT\_HE-IL
- MS\_RETAILDEMOCONTENT\_HR-HR
- MS\_RETAILDEMOCONTENT\_HU-HU
- MS\_RETAILDEMOCONTENT\_ID-ID
- MS\_RETAILDEMOCONTENT\_IT-IT
- MS\_RETAILDEMOCONTENT\_JA-JP
- MS\_RETAILDEMOCONTENT\_KO-KR
- MS\_RETAILDEMOCONTENT\_LT-LT
- MS\_RETAILDEMOCONTENT\_LV-LV
- MS\_RETAILDEMOCONTENT\_NB-NO
- MS\_RETAILDEMOCONTENT\_NEUTRAL
- MS\_RETAILDEMOCONTENT\_NL-NL
- MS\_RETAILDEMOCONTENT\_PL-PL
- MS\_RETAILDEMOCONTENT\_PT-BR
- MS\_RETAILDEMOCONTENT\_PT-PT
- MS\_RETAILDEMOCONTENT\_RO-RO

- MS\_RETAILDEMOCONTENT\_RU-RU
- MS\_RETAILDEMOCONTENT\_SK-SK
- MS\_RETAILDEMOCONTENT\_SL-SI
- MS\_RETAILDEMOCONTENT\_SR-LATN-RS
- MS\_RETAILDEMOCONTENT\_SV-SE
- MS\_RETAILDEMOCONTENT\_TH-TH
- MS\_RETAILDEMOCONTENT\_TR-TR
- MS\_RETAILDEMOCONTENT\_UK-UA
- MS\_RETAILDEMOCONTENT\_VI-VN
- MS\_RETAILDEMOCONTENT\_ZH-CN
- MS\_RETAILDEMOCONTENT\_ZH-HK
- MS\_RETAILDEMOCONTENT\_ZH-TW

## Test features defined by Microsoft

The following table describes the Microsoft-defined test features that can be used by OEMs in the **Features** element in the OEMInput file. These features are defined in MSOptionalFeatures.xml, which is included with the MobileOS under %WPDKCONTENTROOT%\FMFiles.

### Boot option features

FEATURE	DESCRIPTION
BOOTSEQUENCE_TEST	<p>This feature includes the BCD boot sequence configuration for booting into the full OS in a test image. This feature also includes a pre-boot crash dump application and a pre-boot crash dump entry.</p> <div style="border: 1px solid black; padding: 5px;"> <p><b>Note</b></p> <p>The following features are mutually exclusive; only one of them can be referenced in an OEMInput file: BOOTSEQUENCE_TEST, MMOSLOADER_TEST.</p> </div>
MMOSLOADER_TEST	<p>This feature includes the BCD boot sequence configuration to support booting into MMOS in a test image. This feature also includes a pre-boot crash dump application and a pre-boot crash dump entry. For more information about MMOS, see <a href="#">MMOS image definition</a>.</p> <div style="border: 1px solid black; padding: 5px;"> <p><b>Note</b></p> <p>The following features are mutually exclusive; only one of them can be referenced in an OEMInput file: BOOTSEQUENCE_TEST, MMOSLOADER_TEST.</p> </div>
MOBILECOREBOOTSH	Enables the bootsh service (bootshscv) so that startup.bsc features, such as telnet and ftp, can be used.
MOBILECORE_TEST	Information on this feature will be provided in a later release of this documentation.
PRODUCTION	Includes core OS files that are used to support base OS functionality for production builds in the main OS, UEFI and the update OS. This feature is used for PRODUCTION Image types.

FEATURE	DESCRIPTION
PRODUCTION_CORE	<p>This feature adds production boot and main OS binaries to the image. PRODUCTION_CORE automatically includes the following features:</p> <ul style="list-style-type: none"> <li>• CODEINTEGRITY_PROD</li> </ul> <p>Because these features are already included, they should not be manually included in PRODUCTION_CORE builds.</p>
DISABLE_FFU_PLAT_ID_CHECK	<p>Disables the device platform validation in the Microsoft flashing application. For more information about the platform check in flashing, see <a href="#">Use the flashing tools provided by Microsoft</a>.</p> <div style="border: 1px solid black; padding: 5px;"> <p><b>Important</b></p> <p>The device platform validation for flashing must not be disabled in retail images.</p> </div>
RESET_PROTECTION_INTERNAL	<p>This feature enables Reset Protection on the device. When the device boots for the first time, the appropriate secure UEFI variable are written.</p>

### Boot option feature constraints

You can specify feature constraints to avoid illogical or inappropriate build configurations.

Some settings are mutually exclusive and only one setting should be specified at a time. For example, consider the features, RELEASE\_PRODUCTION and RELEASE\_TEST. These features are mutually exclusive. This means that if RELEASE\_TEST is set, RELEASE\_PRODUCTION must not be set. For more information about how constraints are specified in XML see, [Feature groupings and constraints](#).

When <ReleaseType>Production</ReleaseType> is set in the OEMInput file, this maps to RELEASE\_PRODUCTION. When <ReleaseType>Test</ReleaseType> is set in the OEMInput file, this maps to RELEASE\_TEST. For more information about the release type, see [OEMInput file contents](#).

The release constraints for RELEASE\_PRODUCTION can be summarized as follows.

- Either RELEASE\_PRODUCTION or CODEINTEGRITY\_PROD can be selected. This is because production code integrity is automatically enabled when RELEASE\_PRODUCTION is selected and therefore can't be manually enabled.

These feature constraints can be used to prevent incorrect build option configurations. These feature constraints specify that PRODUCTION\_CORE is mutually exclusive with RELEASE\_PRODUCTION and TEST but is not mutually exclusive with HEALTH, PRODUCTION, or SELFHOST.

This table provides a summary of build options and indicates if the option is exclusive to any other option.

	RELEASE_PRODUCTION	TEST	HEALTH	PRODUCTION	SELFHOST	PRODUCTION_CORE
RELEASE_PRODUCTION	NA	Yes	Yes	Yes	Yes	Yes
TEST	Yes	NA	Yes	Yes	Yes	Yes
HEALTH	Yes	Yes	NA	Yes	Yes	No
PRODUCTION	Yes	Yes	Yes	NA	Yes	No

	RELEASE_PRODUCTION	TEST	HEALTH	PRODUCTION	SELFHOST	PRODUCTION_CORE
SELFHOST	Yes	Yes	Yes	Yes	NA	No
PRODUCTION_CORE	Yes	Yes	No	No	No	NA

### Other boot related features

FEATURE	DESCRIPTION
STARTUPOVERRIDES	This feature starts the FTP service (ftpd.exe) and Telnet service (telnetd.exe) automatically on startup.
LABIMAGE	This feature causes the device to enter the FFU download mode automatically when the device is booted. For more information, see <a href="#">Use the flashing tools provided by Microsoft</a> .
BOOTKEYACTIONS_RETAIL	This feature enables a set of button actions for use in retail devices.
SKIPOOBE	This feature disables the initial setup process wizard. This feature is supported in Test, Health and Production image types. This feature must not be used in Retail images.

### Security related features

There are two code signing modes on a Windows 10 Mobile device, retail and test. With retail, all code must be production signed to be able to run on the device. For test signing, all code must be signed with test certificates. For more information about code signing, see [Code signing](#).

The two code signing modes are automatically managed in the build system. The previous DISABLETESTSIGNING and ENABLETESTSIGNING feature settings are no longer used. Instead test code signing is automatically enabled in the following build types:

- TEST
- HEALTH
- PRODUCTION
- SELFHOST

Test code signing cannot be enabled in the RELEASE\_PRODUCTION build type that is used for retail devices

The following table summarizes the security related features.

FEATURE	DESCRIPTION
CODEINTEGRITY_PROD	This feature includes code integrity binaries that are used for signature verification on production images. Code integrity verifies the integrity of binary files as they are loaded into memory by the operating system. This feature is automatically included when PRODUCTION_CORE is selected. Because of this, CODEINTEGRITY_PROD should not be manually added when PRODUCTION_CORE is selected.
CODEINTEGRITY_TEST	This feature includes code integrity binaries that are used for signature verification on test images. Code integrity verifies the integrity of binary files as they are loaded into memory by the operating system.

FEATURE	DESCRIPTION
GWPCERTTESTPROV	This feature provisions a set of test Genuine Windows Phone Certificates (GWPC) certificates in the image.

## Test related features

FEATURE	DESCRIPTION
TEST	This feature adds many test drivers, applications, and other components to be used for testing the OS in different conditions.
TESTINFRASTRUCTURE	This feature adds the following components to the image: <ul style="list-style-type: none"> <li>MinTE.exe and other components that support a minimal test harness environment for Test Authoring and Execution Framework (TAEF) tests.</li> <li>The Verifier.cmd script, which is used to configure Driver Verifier.</li> <li>Tux.exe and other components related to the Tux test harness for native code.</li> <li>TuxNet.exe and other components related to the TuxNet test harness for managed code.</li> </ul>
HEALTH	This feature adds components for running tests related to power and performance.
DRIVERS_WDTFINFRA	Information on this feature will be provided in a later release of this documentation.
DRIVERS_WDTFPPOWER	Information on this feature will be provided in a later release of this documentation.
DRIVERS_WDTPLUGINS	Information on this feature will be provided in a later release of this documentation.
DRIVERS_WDTFDRVCOV	Information on this feature will be provided in a later release of this documentation.
DRIVERS_WDTFIOSPY	Information on this feature will be provided in a later release of this documentation.

## Debug related features

Use the following settings to specify the transport that is used for debugging. The previous DEBUGGERON feature has been deprecated.

FEATURE	DESCRIPTION
	<b>Debug transport settings</b>

FEATURE	DESCRIPTION
KDNETUSB_ON	<p>Includes all kernel debugger transports and enables KDNET over USB.</p> <p>The default debug transport settings for this feature are an IP address of "1.2.3.4", a port address of "50000", and a debugger key of "4.3.2.1". To use the default IP address of 1.2.3.4, run VirtEth.exe with the /autodebug flag. To establish a kernel debugger connection to the phone, use the following command.</p> <pre>Windbg -k net:port=50000,key=4.3.2.1</pre>
KDUSB_ON	<p>Includes all kernel debugger transports and enables KDUSB.</p> <p>The default debug transport target name for this feature is WOATARGET. To establish a kernel debugger connection to the phone, use the following command.</p> <pre>Windbg -k usb:targetname=WOATARGET</pre> <div style="border: 1px solid black; padding: 5px;"> <p><b>Note</b></p> <p>Do not include either KDUSB_ON or KDNETUSB_ON if you need to enable MTP or IP over USB in the image. If the kernel debugger is enabled in the image and the debug transports are used to connect to the device, the kernel debugger has exclusive use of the USB port and prevents MTP and IP over USB from working.</p> </div>
KDSERIAL_ON	Includes all kernel debugger transports and enables KDSERIAL with the following settings: 115200 Baud, 8 bit, no parity.
KD	Includes all kernel debugger transports in the image, but does not enable the kernel debugger.
KD_TEST	Includes all kernel debugger transports in the image, but does not enable the kernel debugger.
KDNETUSB_TEST_ON	Includes all kernel debugger transports and enables KDNET over USB in test images. This option must only be used with test images.

### Other debug settings

FEATURE	DESCRIPTION
	Dumpsize setting features - The following three features must only be used with the Test and Health image types. These features must not be used with retail images. Only one dumpsize setting can be selected at a time.
DUMPSIZE512MB	Specifies a pre-allocated crash dump file size of 592 MB. This is intended for a phone with 512 MB of memory.
DUMPSIZE1G	Specifies a pre-allocated crash dump file size of 1104 MB. This is intended for a phone with 1024 MB of memory.
DUMPSIZE2G	Specifies a pre-allocated crash dump file size of 2128 MB. This is intended for a phone with 2048 MB of memory.

FEATURE	DESCRIPTION
DUMPSIZE4G	Specifies a pre-allocated crash dump file size of 4 GB. This is intended for a phone with 4096 MB of memory.
	Dump data storage location - The next two settings control if crash dump data is stored on eMMC or if crash dump data is stored on an SD card. Only one of these settings can be selected at a time. These two features must only be used with the Test, Health and Selfhost image types. These features must not be used with retail images.
DEDICATEDDUMPONEMMC	Specifies that the DedicatedDumpFile location as c:\crashdump\dedicateddump.sys.
DEDICATEDDUMPONSD	<p>DEDICATEDDUMPONSD – Specifies that the DedicatedDumpFile location as d:\dedicateddump.sys</p> <p>When DEDICATEDDUMPONSD is used, crash dump will be disabled if the user removes the SD card or if the card is not present when the device booted. To re-enable crash dump:</p> <ol style="list-style-type: none"> <li>Set this registry key  <code>HKLM\System\CurrentControlSet\Control\CrashControl\CrashDumpEnabled</code>            to the value of  <code>HKLM\System\CurrentControlSet\Control\CrashControl\ExpectedCrashDumpDevice</code></li> <li>Reboot the phone.</li> </ol>
	Test disk idle power behavior - The next two settings determine if storage devices (internal and SD card) can enter into a low power state after they become idle. They must not be used with retail images.
TEST_ENABLE_DISK_IDLE	This feature will allow the storage devices (internal and SD card) to go into a low power state shortly after they become idle. This setting may prevent the collection of crashdumps on the MSM8960 processor devices. If crashdumps need to be gathered from an MSM8960 device, use the TEST_DISABLE_DISK_IDLE feature instead.
TEST_DISABLE_DISK_IDLE	This feature will prevent the storage devices (internal and SD card) from going into a low power state after they become idle. This setting should be used if crashdumps need to be gathered from MSM8960 devices.
DRVRF_SIPLAT	<p>Sets driver verifier parameters for OEM and SoC drivers. This feature sets the VerifyDriverLevel to a value of 002209BB and VerifierOptions to 00000009. Windows drivers provided by Microsoft are excluded using the VerifyDrivers key. You can use the following debug command to list the drivers that are being verified in your environment.</p> <pre>!Verifier 1</pre>

FEATURE	DESCRIPTION
DBGCHKDISABLE	<p>This feature disables debugger connection checking and the debugger connection status is ignored.</p> <p>The effect on the debugger behavior is different depending on the SoC that is being used.</p> <ul style="list-style-type: none"> <li>For QC8x26 and QC8974 devices, include DBGCHKDISABLE to ensure that offline dumps will be generated even if the device is connected to a debugger. Otherwise, an Offline Dump (Bug Check Code 0x14C dump) will not be created but a raw dump will still be created.</li> <li>For 8960 devices, include DBGCHKDISABLE to ensure that offline dumps will be generated even if the device is connected to a debugger. Otherwise, an Offline Dump (i.e. Bug Check Code 0x14C dump) will not be created.</li> </ul>
MSVCRT_DEBUG	<p>This feature adds support for explicit linking of debug c-runtime libs by including msvc120d.dll and msrvcr120d.dll in the image. For more information, see this topic on MSDN: <a href="#">CRT Library Features</a>.</p>
MWDBGSRV	<p>This feature adds support for the user mode debug server.</p>
NOLIVEDUMPS	<p>Disables non-fatal kernel error reports. These reports contain debugging information for OS and driver developers.</p>
TELEMETRYONSDCARD	<p>This feature enables temporary storage of debugging logs and files on the SD card. This feature is only appropriate for test/self-host images and only on devices with less than 8 GB of free space of primary storage.</p>

### Non-production features

The following are some features that can be used for development and testing support.

FEATURE	DESCRIPTION
POWERTRACINGTOOL	<p>This feature adds PowerTracingTool.exe, which is used to collect power-related ETW logs, to the image.</p>
TASKSCHEDULERTOOL	<p>This feature adds TaskSchTestUtil.exe, which is used to do task scheduling-related operations, to the image.</p>
DISABLEPREBOOTCRASHDUMP	<p>This feature adds a registry setting that disables offline crash dumps.</p>
ALWAYSKEEPMEMORYDUMP	<p>This feature adds a registry setting that tells the device to keep kernel dump files.</p>
AUTOREBOOT	<p>This feature adds a registry setting that restarts the device immediately after a crash dump is complete.</p>
DISABLEDEBUGCHECKSCREEN	<p>This feature adds a registry setting that suppresses the bug check screen.</p>

### Other features

FEATURE	DESCRIPTION
FAKEVIBRA	This feature adds a test hardware notification driver for controlling the vibration feedback mechanism.
IMGFAKELED	This feature adds a test hardware notification driver for controlling LEDs to the image.
LOCATIONFRAMEWORKAPP	This feature adds LFApp, a test and debug application for the location framework.
PSEUDOLOCALES	This feature enables the use of pseudo-locales for localization testing. For more information, see <a href="#">Using Psuedo-Locales for Localization Testing</a> on MSDN.
GRAPHICS SOFTWARE DRIVERS	This feature adds BasicDisplay and WARP graphics drivers. This feature is normally used by the SoC vendor in early device bring-up.

## Microsoft internal features

The following components are reserved for Microsoft internal use only, but are documented here for completeness.

FEATURE	DESCRIPTION
SELFHOST	This feature adds components used exclusively for self-hosting scenarios in Microsoft.
IMGUPD_POWER_TOYS	This feature includes several utility applications related to updating packages on devices.
INSTRUMENT_FOR_COVERAGE	Information on this feature will be provided in a later release of this documentation.
IMGFAKE_MODEM	Information on this feature will be provided in a later release of this documentation.
USE_WMC	This feature is used for Microsoft testing.
CORTANA_DBG_TEST_PROTECTED	This feature is for Microsoft use only.
LIMITED	This feature is for Microsoft use only.
TEST_PROTECTED	This feature is for Microsoft use only.
SELFHOST_PROTECTED	This feature is for Microsoft use only.

## Related topics

[Building and flashing images](#)

# Feature manifest file contents

1/25/2019 • 24 minutes to read

*Feature manifest (FM) files* are used to define specific types of image builds that contain different sets of optional packages. This topic describes the required and optional elements in a FM file. For more information about how FM files are used while generating images, see [Building a phone image using ImgGen.cmd](#).

## Note

Most of the elements in a FM file include a path to a package. If the package is under the root directory for Microsoft packages (%WPDKCONTENTROOT%\MS Packages), this path can use the \$(mspackageroot) macro in the path name. If the package is in some other location, you can use an environment variable, such as %oempackageroot%, and set this environment variable in the command-line window.

## Example FM file

The following example shows a sample OEM FM file.

```
<?xml version="1.0" encoding="utf-8"?>
<FeatureManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate">
    <BasePackages>
        <PackageFile Path="%oempackageroot%\common\" Name="Contoso.Phone.Test.BaseOs.EnvPath.spkg" />
    </BasePackages>
    <Features>
        <OEM>
            <PackageFile Path="%oempackageroot%\test\" Name="Contoso.Test.MinTE.spkg">
                <FeatureIDs>
                    <FeatureID>TEST_FEATURE1</FeatureID>
                </FeatureIDs>
            </PackageFile>
        </OEM>
    </Features>
    <ReleasePackages>
        <PackageFile FeatureIdentifierPackage="true" Name="Contoso.BaseOS.BootApplications_Test.spkg" Path="%oempackageroot%\test" ReleaseType="Test"/>
    </ReleasePackages>
    <PrereleasePackages>
        <PackageFile ID="Contoso.MainOS.Protected_Replacement_Production" FeatureIdentifierPackage="true" Name="Contoso.MainOS.Protected_Replacement_Production.spkg" Path="%oempackageroot%\Merged\" Resolution="*" Type="replacement" Language="*"/>
    </PrereleasePackages>
    <OEMDevicePlatformPackages>
        <PackageFile Name="SoCVendor.DCD6000.OEMDevicePlatform.spkg" Path="%oempackageroot%\DCD6000\" Device="DCD6000"/>
    </OEMDevicePlatformPackages>
    </FeatureManifest>
```

You may wish to examine MSOptionalFeaturesFM.xml that is included with the mobileOS packages under %WPDKCONTENTROOT%\FMFiles to see additional FM XML files.

## Elements in a FM file

The following sections describe the supported elements in a FM file.

## FeatureManifest

The **FeatureManifest** element is the root XML element in the FM file. This element is the base container element for all other elements in the file. It must occur only once.

## BasePackages

The **BasePackages** element specifies packages that are always included in the image if the FM file is referenced in the **AdditionalFMs** element of the OEMInput XML file. The **BasePackages** element has no supported attributes.

The following table describes the child elements of the **BasePackages** element.

ELEMENT	DESCRIPTION
<b>PackageFile</b>	<p>Optional. This element describes a package that will be included in the image. This element supports the following attributes:</p> <ul style="list-style-type: none"><li>• <b>Path</b> – Required. The path to the package.</li><li>• <b>Name</b> – Required. The file name of the package.</li><li>• <b>Resolution</b> – Optional. A string that contains the display resolutions supported by the package. This attribute can be specified with the following values:<ul style="list-style-type: none"><li>◦ "": The "" character means that the package supports every resolution.</li><li>◦ "(720x1280;768x1280)": This syntax indicates the set of resolutions that the package supports. The package is included only in images that are built for one of the resolutions in this list.</li><li>◦ "!(720x1280;768x1280)": A '!' in front of the resolution list specifies that the package supports all resolutions except for those in the list. The package is included only in images that are not built for one of the resolutions in this list.</li></ul></li><li>• <b>Language</b> – Optional. A string that contains the display language codes supported by the package. This attribute can be specified with the following values:<ul style="list-style-type: none"><li>◦ "": The "" character means that the package supports every language.</li><li>◦ "(en-US;zh-CN)": This syntax indicates the set of languages that the package supports. The package is included only in images that contain one of the display languages in this list.</li><li>◦ "!(en-US;zh-CN)": A '!' in front of the language list specifies that the package supports all languages except or those in the list. The package is included only in images that do not contain one of the display languages in this list.</li></ul></li><li>• <b>Partition</b> – Optional. A string that specifies the target partition for the package. By default, packages are installed to the MainOS partition unless another is explicitly specified.</li></ul>

ELEMENT	DESCRIPTION
	<p>These elements are for Microsoft internal use only.</p> <ul style="list-style-type: none"> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and\or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> </ul>

## Features

There are both **Microsoft** and **OEM** elements, each of which can contain **Features**. The OEM should only add their features to the <**OEM**> element. This provides multiple benefits, including the easier integration of newer versions of the OEMInput.xml files into the OEM's build system.

The **Features** element defines one or more optional features that can be referenced in the **Features** element in the OEMInput file to include optional packages in the image. The **Features** element has no supported attributes.

The following table describes the child elements of the **Features** element.

ELEMENT	DESCRIPTION
<b>PackageFile</b>	<p>Optional. This element describes a package that will be included in the image if the OEMInput file references this feature.</p> <p>This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>Path</b> – Required. The path to the package.</li> <li>• <b>Name</b> – Required. The file name of the package.</li> <li>• <b>Resolution</b> – Optional. A string that contains the display resolutions supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>◦ "": The "" character means that the package supports every resolution.</li> <li>◦ "(720x1280;768x1280)": This syntax indicates the set of resolutions that the package supports. The package is included only in images that are built for one of the resolutions in this list.</li> <li>◦ "!(720x1280;768x1280)": A '!' in front of the resolution list specifies that that package supports all resolutions except for those in the list. The package is included only in images that are not built for one of the resolutions in this list.</li> </ul> </li> <li>• <b>Language</b> – Optional. A string that contains the display language codes supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>◦ "": The "" character means that the package supports every language.</li> </ul> </li> </ul>

ELEMENT	DESCRIPTION
	<ul style="list-style-type: none"> <li>◦ "(en-US;zh-CN)": This syntax indicates the set of languages that the package supports. The package is included only in images that contain one of the display languages in this list.</li> <li>◦ "!(en-US;zh-CN)": A '!' in front of the language list specifies that that package supports all languages except for those in the list. The package is included only in images that do not contain one of the display languages in this list.</li> </ul> <ul style="list-style-type: none"> <li>• <b>Partition</b> – Optional. A string that specifies the target partition for the package. By default, packages are installed to the MainOS partition unless another is explicitly specified.</li> </ul> <p>This element supports the following child elements:</p> <ul style="list-style-type: none"> <li>• <b>FeatureIDs</b> – Required. Contains one or more <b>FeatureID</b> elements. Each <b>FeatureID</b> element contains a string value that specifies the name of a feature that will be associated with the package specified by the parent <b>PackageFile</b> element.</li> </ul> <p>These elements are for Microsoft internal use only.</p> <ul style="list-style-type: none"> <li>• <b>CPUType</b> – Required. String value. Sets the CPU type. Can be set to either <i>x86</i> or <i>arm</i>. This attribute should not be used by OEMs.</li> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and\or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> </ul>

## Feature groupings and constraints

Feature groupings and constraints can be used to manage features. For more information, see [Feature groupings and constraints](#).

## ReleasePackages

The **ReleasePackages** element specifies packages that are included in images of a specific release type, as specified by the **ReleaseType** element in the OEMInput file. The **ReleasePackages** element have no supported attributes.

The following table describes the child elements of the **ReleasePackages** element.

ELEMENT	DESCRIPTION
<b>PackageFile</b>	<p>Optional. This element describes a package.</p> <p>This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>ReleaseType</b> – Required. The type of release supported by the package, either <b>Production</b> or <b>Test</b>. The package will only be included in images</li> </ul>

ELEMENT	DESCRIPTION
	<p>of the specified release type.</p> <ul style="list-style-type: none"> <li>● <b>Path</b> – Required. The path to the package.</li> <li>● <b>Name</b> – Required. The file name of the package.</li> <li>● <b>Resolution</b> – Optional. A string that contains the display resolutions supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>○ "": The "" character means that the package supports every resolution.</li> <li>○ "(720x1280;768x1280)": This syntax indicates the set of resolutions that the package supports. The package is included only in images that are built for one of the resolutions in this list.</li> <li>○ "!(720x1280;768x1280)": A '!' in front of the resolution list specifies that the package supports all resolutions except for those in the list. The package is included only in images that are not built for one of the resolutions in this list.</li> </ul> </li> <li>● <b>Language</b> – Optional. A string that contains the display language codes supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>○ "": The "" character means that the package supports every language.</li> <li>○ "(en-US;zh-CN)": This syntax indicates the set of languages that the package supports. The package is included only in images that contain one of the display languages in this list.</li> <li>○ "!(en-US;zh-CN)": A '!' in front of the language list specifies that the package supports all languages except for those in the list. The package is included only in images that do not contain one of the display languages in this list.</li> </ul> </li> <li>● <b>Partition</b> – Optional. A string that specifies the target partition for the package. By default, packages are installed to the MainOS partition unless another is explicitly specified.</li> </ul> <p>These elements are for Microsoft internal use only.</p> <ul style="list-style-type: none"> <li>● <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>● <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and\or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>● <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> </ul>

Describes packages that can be excluded using the **ExcludePrereleaseFeatures** element in the OEMInput file. For more information, see [OEMInput file contents](#).

ELEMENT	DESCRIPTION
<b>PackageFile</b>	<p>Optional. This element describes a prelease package.</p> <p>This element supports the following attributes:</p> <ul style="list-style-type: none"><li>• <b>Type</b> – Required. Can be either <b>protected</b> or <b>replacement</b>.<p>The <b>protected</b> packages are excluded when <b>ExcludePrereleaseFeatures</b> is set to <b>Yes</b> and the replacement packages will instead be included. For example a replacement feature can be created by the OEM to enable scenarios such as mobile operator testing, while not distributing builds with sensitive functionality. This approach is one of many, and is not required, but is one option to consider, to help manage feature disclosure. For more information see <a href="#">OEMInput file contents</a>.</p><div style="border: 1px solid black; padding: 5px;"><p><b>Important</b></p><p>No replacement packages should be included in a retail image.</p></div></li><li>• <b>Path</b> – Required. The path to the package.</li><li>• <b>Name</b> – Required. The file name of the package.</li><li>• <b>Resolution</b> – Optional. A string that contains the display resolutions supported by the package. This attribute can be specified with the following values:<ul style="list-style-type: none"><li>◦ "": The "" character means that the package supports every resolution.</li><li>◦ "(720x1280;768x1280)": This syntax indicates the set of resolutions that the package supports. The package is included only in images that are built for one of the resolutions in this list.</li><li>◦ "!(720x1280;768x1280)": A '!' in front of the resolution list specifies that the package supports all resolutions except for those in the list. The package is included only in images that are not built for one of the resolutions in this list.</li></ul></li><li>• <b>Language</b> – Optional. A string that contains the display language codes supported by the package. This attribute can be specified with the following values:<ul style="list-style-type: none"><li>◦ "": The "" character means that the package supports every language.</li><li>◦ "(en-US;zh-CN)": This syntax indicates the set of languages that the package supports. The package is included only in images that contain one of the display languages in this list.</li></ul></li></ul>

ELEMENT	DESCRIPTION
	<ul style="list-style-type: none"> <li>◦ "!(en-US;zh-CN)": A '!' in front of the language list specifies that the package supports all languages except for those in the list. The package is included only in images that do not contain one of the display languages in this list.</li> </ul> <ul style="list-style-type: none"> <li>• <b>Partition</b> – Optional. A string that specifies the target partition for the package. By default, packages are installed to the MainOS partition unless another is explicitly specified.</li> </ul> <p>These elements are for Microsoft internal use only.</p> <ul style="list-style-type: none"> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and/or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> </ul>

## SOC Packages

The **SOC Packages** element specifies packages that are included in images that are generated for a specific SoC, as specified by the **SOC** element in the OEMInput file. The **SOC Packages** element has no supported attributes.

The following table describes the child elements of the **SOC Packages** element.

ELEMENT	DESCRIPTION
<b>PackageFile</b>	<p>Optional. This element describes a package.</p> <p>This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>SOC</b> – Required. The type of SoC supported by the package. For a list of the supported values, see the description of the <b>SOC</b> element in <a href="#">OEMInput file contents</a>. The package will only be included in images generated for the specified SoC.</li> <li>• <b>Path</b> – Required. The path to the package.</li> <li>• <b>Name</b> – Required. The file name of the package.</li> <li>• <b>Resolution</b> – Optional. A string that contains the display resolutions supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>◦ "": The "" character means that the package supports every resolution.</li> <li>◦ "(720x1280;768x1280)": This syntax indicates the set of resolutions that the package supports. The package is included only in images that are built for one of the resolutions in this list.</li> <li>◦ "!(720x1280;768x1280)": A '!' in front of the resolution list specifies that the package supports all resolutions except</li> </ul> </li> </ul>

ELEMENT	DESCRIPTION
	<p>for those in the list. The package is included only in images that are not built for one of the resolutions in this list.</p> <ul style="list-style-type: none"> <li>● <b>Language</b> – Optional. A string that contains the display language codes supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>○ "": The "" character means that the package supports every language.</li> <li>○ "(en-US;zh-CN)": This syntax indicates the set of languages that the package supports. The package is included only in images that contain one of the display languages in this list.</li> <li>○ "!(en-US;zh-CN)": A '!' in front of the language list specifies that the package supports all languages except for those in the list. The package is included only in images that do not contain one of the display languages in this list.</li> </ul> </li> <li>● <b>Partition</b> – Optional. A string that specifies the target partition for the package. By default, packages are installed to the MainOS partition unless another is explicitly specified.</li> </ul> <p>These elements are for Microsoft internal use only.</p> <ul style="list-style-type: none"> <li>● <b>CPUType</b> – Required. String value. Sets the CPU type. Can be set to either <i>x86</i> or <i>arm</i>. This attribute should not be used by OEMs. This attribute should not be used by OEMs.</li> <li>● <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>● <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and\or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>● <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> </ul>

## SVPackages

The **SVPackages** element specifies packages that are included in images that are generated for a specific SoC manufacturer, as specified by the **SV** element in the OEMInput file. The **SVPackages** element has no supported attributes.

The following table describes the child elements of the **SVPackages** element.

ELEMENT	DESCRIPTION
<b>PackageFile</b>	<p>Optional. This element describes a package. This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>● <b>SV</b> – Required. The vendor for the SoC that is supported by the package. The package will only be included in images generated for the specified SoC vendor.</li> </ul>

ELEMENT	DESCRIPTION
	<p>The path to the package.</p> <ul style="list-style-type: none"> <li>● <b>Name</b> – Optional.</li> <li>● <b>Resolution</b> – Optional. A string that contains the display resolutions supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>○ "": The "" character means that the package supports every resolution.</li> <li>○ "(720x1280;768x1280)": This syntax indicates the set of resolutions that the package supports. The package is included only in images that are built for one of the resolutions in this list.</li> <li>○ "!(720x1280;768x1280)": A '!' in front of the resolution list specifies that the package supports all resolutions except for those in the list. The package is included only in images that are not built for one of the resolutions in this list.</li> </ul> </li> <li>● <b>Language</b> – Optional. A string that contains the display language codes supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>○ "": The "" character means that the package supports every language.</li> <li>○ "(en-US;zh-CN)": This syntax indicates the set of languages that the package supports. The package is included only in images that contain one of the display languages in this list.</li> <li>○ "!(en-US;zh-CN)": A '!' in front of the language list specifies that the package supports all languages except for those in the list. The package is included only in images that do not contain one of the display languages in this list.</li> </ul> </li> <li>● <b>Partition</b> – Optional. A string that specifies the target partition for the package. By default, packages are installed to the MainOS partition unless another is explicitly specified.</li> </ul> <p>These elements are for Microsoft internal use only.</p> <ul style="list-style-type: none"> <li>● <b>CPUType</b> – Required. String value. Sets the CPU type. Can be set to either <i>x86</i> or <i>arm</i>. This attribute should not be used by OEMs. This attribute should not be used by OEMs.</li> <li>● <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>● <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and\or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>● <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> </ul>

ELEMENT	DESCRIPTION				
<b>OEMDevicePlatformPackages</b>	<p>The <b>OEMDevicePlatformPackages</b> element specifies the device platform package to include in an image for a specific device type. OEMs must specify the device platform package by using this element in a FM file. For more information about device platform packages, see <a href="#">Set device platform information</a>. The <b>OEMDevicePlatformPackages</b> element has no supported attributes.</p> <p>The following table describes the child elements of the <b>OEMDevicePlatformPackages</b> element.</p> <table border="1"> <thead> <tr> <th data-bbox="165 422 810 480">ELEMENT</th><th data-bbox="810 422 1447 480">DESCRIPTION</th></tr> </thead> <tbody> <tr> <td data-bbox="165 480 810 1466"><b>PackageFile</b></td><td data-bbox="810 480 1447 1466"> <p>Optional. This element describes a device platform package to include in the image for a specific device type.</p> <p>This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>Device</b> – Required. The device type that is supported by the device platform package. The package will only be included in images generated for the specified device type.</li> <li>• <b>Path</b> – Required. The path to the package.</li> <li>• <b>Name</b> – Required. The file name of the package.</li> </ul> <p>These elements are for Microsoft internal use only.</p> <ul style="list-style-type: none"> <li>• <b>CPUType</b>– Required. String value. Sets the CPU type. Can be set to either <i>x86</i> or <i>arm</i>. This attribute should not be used by OEMs.</li> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and/or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> </ul> </td></tr> </tbody> </table>	ELEMENT	DESCRIPTION	<b>PackageFile</b>	<p>Optional. This element describes a device platform package to include in the image for a specific device type.</p> <p>This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>Device</b> – Required. The device type that is supported by the device platform package. The package will only be included in images generated for the specified device type.</li> <li>• <b>Path</b> – Required. The path to the package.</li> <li>• <b>Name</b> – Required. The file name of the package.</li> </ul> <p>These elements are for Microsoft internal use only.</p> <ul style="list-style-type: none"> <li>• <b>CPUType</b>– Required. String value. Sets the CPU type. Can be set to either <i>x86</i> or <i>arm</i>. This attribute should not be used by OEMs.</li> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and/or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> </ul>
ELEMENT	DESCRIPTION				
<b>PackageFile</b>	<p>Optional. This element describes a device platform package to include in the image for a specific device type.</p> <p>This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>Device</b> – Required. The device type that is supported by the device platform package. The package will only be included in images generated for the specified device type.</li> <li>• <b>Path</b> – Required. The path to the package.</li> <li>• <b>Name</b> – Required. The file name of the package.</li> </ul> <p>These elements are for Microsoft internal use only.</p> <ul style="list-style-type: none"> <li>• <b>CPUType</b>– Required. String value. Sets the CPU type. Can be set to either <i>x86</i> or <i>arm</i>. This attribute should not be used by OEMs.</li> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and/or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> </ul>				

## DeviceSpecificPackages

The **DeviceSpecificPackages** element specifies packages to include in images that are generated for a specific device type, as specified by the **Device** element in the OEMInput file. The **DeviceSpecificPackages** element has no supported attributes.

The following table describes the child elements of the **DeviceSpecificPackages** element.

ELEMENT	DESCRIPTION
<b>PackageFile</b>	<p>Optional. This element describes a package to include in the image for a specific device type.</p> <p>This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>Device</b> – Required. The device type that is supported by the device platform package. The package will only be included in images generated for the specified device type.</li> <li>• <b>Path</b> – Required. The path to the package.</li> </ul>

ELEMENT	DESCRIPTION
	<ul style="list-style-type: none"> <li>• <b>Name</b> – Required. The file name of the package.</li> <li>• <b>Resolution</b> – Optional. A string that contains the display resolutions supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>◦ "": The "" character means that the package supports every resolution.</li> <li>◦ "(720x1280;768x1280)": This syntax indicates the set of resolutions that the package supports. The package is included only in images that are built for one of the resolutions in this list.</li> <li>◦ "!(720x1280;768x1280)": A '!' in front of the resolution list specifies that the package supports all resolutions except for those in the list. The package is included only in images that are not built for one of the resolutions in this list.</li> </ul> </li> <li>• <b>Language</b> – Optional. A string that contains the display language codes supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>◦ "": The "" character means that the package supports every language.</li> <li>◦ "(en-US;zh-CN)": This syntax indicates the set of languages that the package supports. The package is included only in images that contain one of the display languages in this list.</li> <li>◦ "!(en-US;zh-CN)": A '!' in front of the language list specifies that the package supports all languages except for those in the list. The package is included only in images that do not contain one of the display languages in this list.</li> </ul> </li> <li>• <b>Partition</b> – Optional. A string that specifies the target partition for the package. By default, packages are installed to the MainOS partition unless another is explicitly specified.</li> </ul> <p>These elements are for Microsoft internal use only.</p> <ul style="list-style-type: none"> <li>• <b>CPUType</b> – Required. String value. Sets the CPU type. Can be set to either <i>x86</i> or <i>arm</i>. This attribute should not be used by OEMs.</li> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and/or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> </ul>

Microsoft internal use only

The following components are reserved for Microsoft internal use only, but are documented here for completeness.

### **CPUPackages**

Reserved for internal Microsoft use. This element should not be used by OEMs.

ELEMENT	DESCRIPTION

ELEMENT	DESCRIPTION
<b>PackageFile</b>	<p>Optional. This element describes a package.</p> <p>This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>CPUType</b> – Required. String value. Sets the CPU type. Can be set to either <i>x86</i> or <i>arm</i>. This attribute should not be used by OEMs.</li> <li>• <b>Path</b> – Required. The path to the package.</li> <li>• <b>Name</b> – Required. The file name of the package.</li> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and\or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> <li>• <b>Resolution</b> – Optional. A string that contains the display resolutions supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>◦ "": The "" character means that the package supports every resolution.</li> <li>◦ "(720x1280;768x1280)": This syntax indicates the set of resolutions that the package supports. The package is included only in images that are built for one of the resolutions in this list.</li> <li>◦ "!(720x1280;768x1280)": A '!' in front of the resolution list specifies that the package supports all resolutions except for those in the list. The package is included only in images that are not built for one of the resolutions in this list.</li> </ul> </li> <li>• <b>Language</b> – Optional. A string that contains the display language codes supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>◦ "": The "" character means that the package supports every language.</li> <li>◦ "(en-US;zh-CN)": This syntax indicates the set of languages that the package supports. The package is included only in images that contain one of the display languages in this list.</li> <li>◦ "!(en-US;zh-CN)": A '!' in front of the language list specifies that that package supports all languages except for those in the list. The package is included only in images that do not contain one of the display languages in this list.</li> </ul> </li> <li>• <b>Partition</b> – Optional. A string that specifies the</li> </ul>

ELEMENT	DESCRIPTION
---------	-------------

## BootUILanguagePackageFile

Reserved for internal Microsoft use. This element should not be used by OEMs.

ELEMENT	DESCRIPTION
<b>BootUILanguagePackageFile</b>	<p>Optional. This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>Path</b> – Required. The path to the package.</li> <li>• <b>Name</b> – Required. The file name of the package.</li> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and\or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> <li>• <b>Partition</b> – Optional. A string that specifies the target partition for the package.</li> </ul>

## BootLocalePackageFile

Reserved for internal Microsoft use. This element should not be used by OEMs.

ELEMENT	DESCRIPTION
<b>BootLocalePackageFile</b>	<p>Optional. This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>Path</b> – Required. The path to the package.</li> <li>• <b>Name</b> – Required. The file name of the package.</li> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and\or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> <li>• <b>Partition</b> – Optional. A string that specifies the target partition for the package.</li> </ul>

## DeviceLayoutPackages

Reserved for internal Microsoft use. This element should not be used by OEMs.

ELEMENT	DESCRIPTION
<b>PackageFile</b>	<p>Optional. This element describes a package.</p> <p>This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>SOC</b> – Required.</li> <li>• <b>Path</b> – Required. The path to the package.</li> <li>• <b>Name</b> – Required. The file name of the package.</li> <li>• <b>Partition</b> – Optional. A string that specifies the target partition for the package. By default, packages are installed to the MainOS partition unless another is explicitly specified.</li> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and\or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> <li>• <b>CPUType</b> – Required. String value. Sets the CPU type. Can be set to either <i>x86</i> or <i>arm</i>. This attribute should not be used by OEMs. This attribute should not be used by OEMs.</li> <li>• <b>Partition</b> – Optional. A string that specifies the target partition for the package.</li> </ul>

## KeyboardPackages

Reserved for internal Microsoft use. This element should not be used by OEMs.

ELEMENT	DESCRIPTION
---------	-------------

ELEMENT	DESCRIPTION
<b>PackageFile</b>	<p>Optional. This element describes a package.</p> <p>This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>Path</b> – Required. The path to the package.</li> <li>• <b>Name</b> – Required. The file name of the package.</li> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and\or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> <li>• <b>Language</b> – Optional. A string that contains the display language codes supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>◦ "": The "" character means that the package supports every language.</li> <li>◦ "(en-US;zh-CN)": This syntax indicates the set of languages that the package supports. The package is included only in images that contain one of the display languages in this list.</li> <li>◦ "!(en-US;zh-CN)": A '!' in front of the language list specifies that the package supports all languages except for those in the list. The package is included only in images that do not contain one of the display languages in this list.</li> </ul> </li> <li>• <b>Partition</b> – Optional. A string that specifies the target partition for the package. By default, packages are installed to the MainOS partition unless another is explicitly specified.</li> </ul>

## SpeechPackages

Reserved for internal Microsoft use. This element should not be used by OEMs.

ELEMENT	DESCRIPTION
---------	-------------

ELEMENT	DESCRIPTION
<b>PackageFile</b>	<p>Optional. This element describes a package.</p> <p>This element supports the following attributes:</p> <ul style="list-style-type: none"> <li>• <b>Path</b> – Required. The path to the package.</li> <li>• <b>Name</b> – Required. The file name of the package.</li> <li>• <b>ID</b> – Optional. String value. The ID is the package name. This attribute should not be used by OEMs.</li> <li>• <b>NoBasePackage</b> – Optional. Boolean value. Set to true for Language and\or Resolution packages that do not contain base packages. This attribute should not be used by OEMs.</li> <li>• <b>FeatureIdentifierPackage</b> – Optional. Boolean value. This attribute should not be used by OEMs.</li> <li>• <b>Language</b> – Optional. A string that contains the display language codes supported by the package. This attribute can be specified with the following values: <ul style="list-style-type: none"> <li>◦ "": The "" character means that the package supports every language.</li> <li>◦ "(en-US;zh-CN)": This syntax indicates the set of languages that the package supports. The package is included only in images that contain one of the display languages in this list.</li> <li>◦ "!(en-US;zh-CN)": A '!' in front of the language list specifies that the package supports all languages except for those in the list. The package is included only in images that do not contain one of the display languages in this list.</li> </ul> </li> <li>• <b>Partition</b> – Optional. A string that specifies the target partition for the package. By default, packages are installed to the MainOS partition unless another is explicitly specified.</li> </ul>

## Related topics

[Building a phone image using ImgGen.cmd](#)

# Create a feature and include it in an image

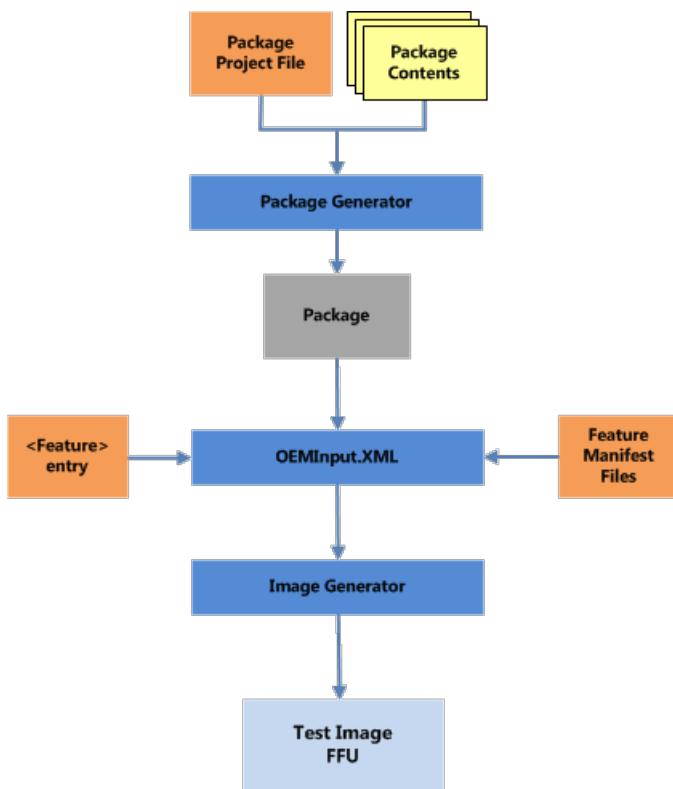
1/25/2019 • 4 minutes to read

This topic shows you how to create a feature and add it to an image.

## Process overview – adding a test app feature

To create a feature and add it to an image, you must complete the following steps.

1. Generate the package that contains the test app.
2. Create the feature manifest file that references the package.
3. Add the feature manifest file and the test app feature to the OEMInput.xml file.
4. Generate the image, sign the image, and flash it to the device.
5. Verify that the feature works as expected.



## Walkthrough – adding a test app to a test image

This section reviews the steps that you need to perform to add a test app to a test image. Before you can start this walkthrough, you must first create a simple test app. After the app is created, you can continue with this walkthrough.

### Generate the test app package

This walkthrough assumes that you have already created a test app named *TestApplication.exe*. Generate a package that contains this app by completing the following steps.

1. Create a directory called *TestApplication* and copy the *TestApplication.exe* file that you created in Visual Studio, to that directory.

2. Create a text file named *TestApplication.pkg.xml* that contains the following package definition XML.

```
<?xml version="1.0" encoding="utf-8" ?>
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00" Owner="Contoso"
OwnerType="OEM"
ReleaseType="Test"
Platform="DCD600"
Component="TestApps"
SubComponent="TestApplication"
Partition="Data">

<Macros>
<Macro Id="testDir" Value="\test" />
</Macros>

<Components>
<OSComponent>
<Files>
<File Source="TestApplication.exe" DestinationDir="$(testDir)" />
</Files>
</OSComponent>
</Components>
</Package>
```

You can update the owner and platform attributes to match the name of your organization name and the name of your device. These attribute changes will modify the name of the generated package.

The `<Macro>` element is used to specify the `\data\test` destination directory on the device.

3. Open an administrator command prompt window to build the package.
4. Display the environment variables by typing **SET** in the command prompt window. Look for `WPDKCONTENTROOT` to confirm that the build environment is properly configured. On a Windows 64-bit PC, the path should look similar to the following.

```
...
WPDKCONTENTROOT=C:\Program Files (x86)\Windows Phone Kits\10
```

5. Generate the package using PkgGen. Provide the version number of 1.0.0.0. The `/config` parameter points to the location of the `pkggen.cfg.xml` file.

```
C:\TestApplication>PkgGen TestApplication.pkg.xml /version:1.0.0.0 /config:"%WPDKCONTENTROOT%
\Tools\bin\i386\pkggen.cfg.xml"
```

6. If PkgGen creates the package successfully, it should return output that is similar to the following.

```
Microsoft (C) pkggen 8.0.12134.0

info: Using external macro file: 'C:\Program Files (x86)\Windows Phone Kits\10\
Tools\bin\i386\pkggen.cfg.xml'
info: Building project file C:\TestApplication\TestApplication.
pkg.xml
info: Building package '.\Contoso.TestApps.TestApplication.spkg'
info: Adding file 'TestApplication.exe' to package '.\Contoso.TestApps.TestApplication.spkg' as
'\test\TestApplication.exe'
info: Done package ".\Contoso.TestApps.TestApplication.spkg"
info: Packages are generated to . successfully
```

For more information about working with packages, see [Creating packages](#).

## Create the feature manifest file

Create a feature manifest file that will define a TEST\_APP OEM feature by completing the following steps.

1. Create a feature manifest file named *OEMCustomAppFM.xml* in the following directory:

```
%WPDKCONTENTROOT%\FMFiles
```

2. Define the TEST\_APP feature by adding the following XML to the *OEMCustomAppFM.xml* file.

```
<?xml version="1.0" encoding="utf-8"?>
<FeatureManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate">
  <!-- TEST_APP FM File 4/30/2015 -->
  <Features>
    <OEM>
      <PackageFile Path="C:\TestApplication\" Name="Contoso.TestApps.TestApplication.spkg">
        <FeatureIDs>
          <FeatureID>TEST_APP</FeatureID>
        </FeatureIDs>
      </PackageFile>
    </OEM>
  </Features>
</FeatureManifest>
```

For more info about feature manifests, see [Feature manifest file contents](#).

## Add the feature to the OEMInput file

Add the TEST\_APP OEM feature to the OEMInput.xml file by completing the following steps.

1. This walkthrough assumes that you have an existing, functional test OEMInput file that enables TShell. For more information about creating test images, see [Building and flashing images](#). For more information about specifying optional features, see [Optional features for building images](#).
2. Edit the OEMinput.xml file to include the *OEMCustomAppFM.xml* feature manifest file that you created in the previous step. The XML will be similar to the following.

```
...
<AdditionalFMs>
  ...
  <AdditionalFM>%WPDKCONTENTROOT%\FMFiles\OEMCustomAppFM.xml</AdditionalFM>
</AdditionalFMs>
```

3. Later in the <Features> section of the OEMInput.xml file add the new TEST\_APP feature to the list of existing features.

```
<Features>
  <Microsoft>
    ...
  </Microsoft>
  <OEM>
    ...
    <Feature>TEST_APP</Feature>
  </OEM>
</Features>
```

## Generate, sign, and flash the image

Complete the following steps to generate, sign, and flash the image.

1. Generate the image using ImgGen and the OEMInput.xml file that you customized in the previous step.

```
C:\>ImgGen Flash.ffu OEMInput.xml "%WPDKCONTENTROOT%\10\MSPackages"
```

2. Before you can sign images, you must first install the test OEM certificates on the PC by following the steps in [Set up the signing environment](#).
3. Sign the generated catalog using the /pk option.

```
C:\> Set SIGN_OEM=1  
C:\> Sign.cmd /pk TestSigned.cat
```

4. Sign the FFU with the signed catalog file using ImageSigner.

```
C:\> ImageSigner Sign Flash.FFU Flash.Cat
```

5. Flash the image to the phone using FFUTool.

```
C:\> FFUTool -Flash Flash.ffu
```

For more information about generating and flashing images, see [Building and flashing images](#).

### Verify that the TestApplication executes as expected

Verify that the TestApplication executes as expected by completing the following steps.

1. Configure a TShell connection to test the image.
2. Establish a connection to the device using the **Open-device** TShell command. Provide the MAC address of the device.

```
PS C:\> Open-device 001122334455
```

3. Confirm that the TestApplication is on the device by using the TShell **Dir-Device** command. The short form of the command, DirD, is shown.

```
PS C:\> DirD \TestApplication.exe /s
```

4. Execute the application by entering the **Exec-Device** cmdlet in the TShell window. The **Exec-Device** cmdlet starts a process on the connected device. By default, the **Exec-Device** cmdlet waits for the process to exit before returning. Use the -Async switch to return immediately. Use the -displayoutput parameter to echo the output.

```
PS C:\> ExecD -displayoutput -filename \Data\Test\TestApplication.exe
```

5. Output similar to the following should be displayed.

```
Output      : Testing console output  
Error       :  
Exit Code   : 0
```

## Related topics

[Creating packages](#)

[Building and flashing images](#)

# Adding a driver to a test image

7/5/2019 • 11 minutes to read

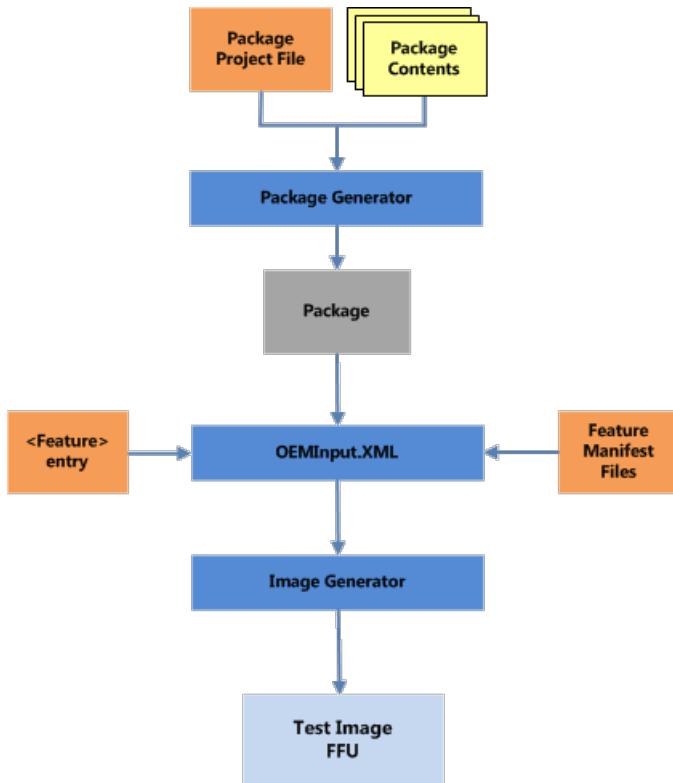
This topic shows you how to create a feature and add it to a test image.

## Process overview – adding a driver feature to a test image

To create a feature that contains a driver and add it to a test image, complete the following steps.

1. [Create a test KMDF driver](#)
2. [Generate the driver package](#)
3. [Create a feature manifest file that references the package](#)
4. [Add the feature to the OEMInput.xml file](#)
5. [Generate, sign, and flash the image to the device](#)
6. [Verify that the KMDFDriver1 is on the device](#)
7. [Verify that the driver loads using Windows debug](#)

The following diagram summarizes the packaging and image generation elements that are used to add the driver to the device.



## Walkthrough – adding a driver to a test image

This topic describes the steps that you need to perform to add a driver to a test image. Before you can start this walkthrough, you must first create a simple KMDF driver.

This walkthrough assumes that you named the project *KmdfDriver1*.

## Create a test KMDF driver

To create a test KMDF driver, complete the following steps.

1. In the **Solution Explorer** window, expand **KmdfDriver1** and then expand the **Source Files Folder**.
2. Edit the "Driver.c" file by clicking it.
3. Right after the variable declaration section, add the **IoReportRootDevice** routine as show below. The **IoReportRootDevice** routine reports a device that cannot be detected by a PnP bus driver to the PnP Manager. This allows the software-only driver to remain loaded.

```
...
{
    WDF_DRIVER_CONFIG config;
    NTSTATUS status;
    WDF_OBJECT_ATTRIBUTES attributes;

    //
    // IoReportRootDevice routine reports a device that cannot be detected by
    // a PnP bus driver to the PnP Manager. This allows our software-only
    // driver to remain loaded.
    IoReportRootDevice(DriverObject);

    //
    // Initialize WPP tracing.
    //
    WPP_INIT_TRACING( DriverObject, RegistryPath );
    TraceEvents(TRACE_LEVEL_INFORMATION, TRACE_DRIVER, "%!FUNC! Entry");

...
}
```

4. Save Driver.c.
5. To build your driver and create a driver package, choose **Build Solution** from the **Build** menu. Visual Studio displays the build progress in the **Output** window. (If the *Output* window is not visible, choose **Output** from the **View** menu.)
6. To view the built driver package, navigate in Windows Explorer to your **KmdfDriver1** folder, and then to **ARM\Win8.1Debug\**. This directory contains the following files:
  - KmdfDriver1.sys - A kernel-mode driver file.
  - KmdfDriver1.inf - An information file that Windows uses to install the driver.
  - KmdfDriver1.pdb - A file that contains the debug symbols that will be used for debugging.
  - KmdfDriver1.cat - A catalog file that is not used in this walkthrough.
- There are also co-installer files for the Windows Driver Frameworks (WDF) that will not be used with Windows Phone.
7. Because of the differences in Windows 10 Mobile INF files, we will not use the desktop INF file that is generated by Visual Studio.

```

;
; KmfdDriver1.inf
;
[Version]
Signature="$WINDOWS NT$"
Class=System
ClassGuid={4d36e97d-e325-11ce-bfc1-08002be10318}
Provider=%MSFT%
DriverVer=11/01/2012,1.00.00.00
BootCritical=1

[DestinationDirs]
DefaultDestDir = 12

[Manufacturer]
%StdMfg%=KMDFDriver1_DevDesc, NTARM, NTx86

;*****
;** models section (required) **
;*****
; For ARM platforms
[KMDFDriver1_DevDesc.NTARM]
; DisplayName      Section      DeviceId
; -----      -----      -----
%KMDFDriver1_DevDesc%=KMDFDriver1, Root\KMDFDriver1

; For Win2K+
[KMDFDriver1_DevDesc.NTx86]
; DisplayName      Section      DeviceId
; -----      -----      -----
%KMDFDriver1_DevDesc%=KMDFDriver1, Root\KMDFDriver1

;*****
;* ddinstall section (required) *
;*****
[KMDFDriver1.NT]
CopyFiles=KMDFDriver1.NT.Copy

[KMDFDriver1.NT.Copy]
KMDFDriver1.sys

;----- Service installation

[KMDFDriver1.NT.Services]
AddService = KMDFDriver1, %SPSVCINST_ASSOCSERVICE%, KMDFDriver1_Service_Inst

[KMDFDriver1_Service_Inst]
DisplayName    = %KMDFDriver1_DevDesc%
ServiceType   = %SERVICE_KERNEL_DRIVER%
StartType     = %SERVICE_SYSTEM_START%
ErrorControl  = %SERVICE_ERROR_NORMAL%
ServiceBinary = %12%\KMDFDriver1.sys

[Strings]
SPSVCINST_ASSOCSERVICE= 0x00000002
MSFT = "Microsoft"
KMDFDriver1_DevDesc = "KMDF Driver 1"
SERVICE_KERNEL_DRIVER = 1
SERVICE_AUTO_START = 2
SERVICE_SYSTEM_START = 1
SERVICE_BOOT_START = 0
SERVICE_ERROR_NORMAL = 1

```

8. Sign the driver using the following commands:

```
C:>set SIGN_OEM=1
sign KMDFDriver1.sys
```

## Generate a package that contains the driver

Generate a package that contains this driver by completing the following steps.

1. Create a directory called *KMDFDriver1* and copy the KMDFDriver1.sys and KMDFDriver1.inf files that you created in Visual Studio, to that directory.
2. Create a text file named *KmddfDriver1.pkg.xml* that contains the following package definition XML.

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
    Owner="Contoso"
    OwnerType="OEM"
    ReleaseType="Test"
    Platform="DCD6000"
    Component="Phone.Test.BaseOS"
    SubComponent="KmddfDriver1"
    Partition="MainOS"
    BinaryPartition="false">
    <Components>
        <OSComponent>
            <Files>
                <File Source="KmddfDriver1.sys" DestinationDir="$(runtime.system32)\drivers"/>
            </Files>
        </OSComponent>
        <Driver InfSource="KmddfDriver1.inf">
            <Reference Source="KmddfDriver1.sys"/>
        </Driver>
    </Components>
</Package>
```

You can update the owner and platform attributes to match the name of your organization name and the name of your device. These attribute changes will modify the name of the generated package.

Specify the *Windows\System32\Drivers* directory on the device using the \$(runtime.system32) macro.

3. The next step is to generate the package by opening an administrator command prompt window.
4. Display the environment variables by typing **SET** in the administrator command prompt window. Look for **WPKCONTENTROOT** to confirm that the build environment is properly configured. You should see that WPKCONTENTROOT is set. On a Windows 64-bit PC, the path should look similar to the following.

```
...
WPKCONTENTROOT=C:\Program Files (x86)\Windows Kits\10
```

5. Generate the package using PkgGen. Provide the version number of 1.0.0.0. The /config parameter points to the location of the pkggen.cfg.xml file provided with the Windows DriverKit. The /hive parameter points to the location of the hive root.

```
C:\KmddfDriver1>PkgGen KmddfDriver1.pkg.xml /version:1.0.0.0
/variables:"HIVE_ROOT=%WPKCONTENTROOT%\CoreSystem"
/config:"%WPKCONTENTROOT%\Tools\bin\i386\pkggen.cfg.xml"
```

6. If PkgGen creates the package successfully, it will display many lines of information as the registry entries are created based on the provided INF file. The last part of the output will be similar to the following.

```
...
info: Import Log: :      sto: Unloaded hive key '{bf1a281b-ad7b-4476-ac95-f47682
990ce7}C:/Users/User1/AppData/Local/Temp/gs5gvfgc.pou/windows/system32/config/S
OFTWARE'. Time = 0 ms
info: Import Log: : <<< Section end 2015/08/06 13:10:22.413
info: Import Log: : <<< [Exit status: SUCCESS]
info: Import Log: :
info: Import Log: :
info: Import Log: : >>> [Unload Offline Registry Hive - DRIVERS]
info: Import Log: : >>> Section start 2014/08/06 13:10:22.413
info: Import Log: :      os: Version = 6.3.9600, Service Pack = 0.0, Suite = 0
x0100, ProductType = 1, Architecture = x86
info: Import Log: :      cmd: PkgGen KmfdDriver1.pkg.xml /version:1.0.0.0 /var
iables:"HIVE_ROOT=C:\Program Files (x86)\Windows Kits\10\CoreSystem" /con
fig:"C:\Program Files (x86)\Windows Kits\10\Tools\bin\i386\pkgen.cfg.xml
"
info: Import Log: :      sto: Closed hive key '{bf1a281b-ad7b-4476-ac95-f4768299
0ce7}C:/Users/User1/AppData/Local/Temp/gs5gvfgc.pou/windows/system32/config/DRI
VERS'.
info: Import Log: :      sto: Unloaded hive key '{bf1a281b-ad7b-4476-ac95-f47682
990ce7}C:/Users/User1/AppData/Local/Temp/gs5gvfgc.pou/windows/system32/config/D
RIVERS'. Time = 0 ms
info: Import Log: : <<< Section end 2014/08/06 13:10:22.513
info: Import Log: : <<< [Exit status: SUCCESS]
info: Import Log: :
info: Building package '.\Contoso.Phone.Test.BaseOS.KmfdDriver1.spkg'
info: Adding file 'KmfdDriver1.sys' to package '.\Contoso.Phone.Test.BaseOS.Kmfd
Driver1.spkg' as '\windows\System32\drivers\KmfdDriver1.sys'
info: Adding file 'C:\Users\User1\AppData\Local\Temp\5bgjkx3p.j01\reg.reg' to p
ackage '.\Contoso.Phone.Test.BaseOS.KmfdDriver1.spkg' as '\windows\Packages\Regi
stryFiles\Contoso.Phone.Test.BaseOS.KmfdDriver1.reg'
info: Adding file 'C:\Users\User1\AppData\Local\Temp\5bgjkx3p.j01\regMultiSz.rg
a' to package '.\Contoso.Phone.Test.BaseOS.KmfdDriver1.spkg' as '\windows\Packag
es\RegistryFiles\Contoso.Phone.Test.BaseOS.KmfdDriver1.rga'
info: Done package ".\Contoso.Phone.Test.BaseOS.KmfdDriver1.spkg"
info: Packages are generated to . successfully
```

For more information about working with packages, see [Creating packages](#).

### Create a feature manifest file that references the package

Create a feature manifest file that will define a DRIVER1 OEM feature by completing the following steps.

1. Create a feature manifest file named *OEMCustomAppFM.xml* in the following directory.

```
%WPDKCONTENTROOT%\FMFiles
```

2. Define the DRIVER1 feature by adding the following XML to the *OEMCustomDriverFM.xml* file. Update the package name to match the name of the package file generated in the previous step.

```

<?xml version="1.0" encoding="utf-8"?>
<FeatureManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate">
  <!-- DRIVER1 FM File 7/31/2014 -->
  <Features>
    <OEM>
      <PackageFile Path="C:\KmdfDriver1\" Name="Contoso.Phone.Test.BaseOS.KmdfDriver1">
        <FeatureIDs>
          <FeatureID>DRIVER1</FeatureID>
        </FeatureIDs>
      </PackageFile>
    </OEM>
  </Features>
</FeatureManifest>

```

For more information about feature manifests, see [Feature manifest file contents](#).

### Add the feature to the OEMInput.xml file

Add the DRIVER1 feature to the OEMInput.xml file by completing the following steps.

1. This walkthrough assumes that you have an existing, functional test OEMInput file that enables TShell. For more information about creating test images, see [Building and flashing images](#). For more information about specifying optional features, see [Optional features for building images](#).

Confirm that that you are using a test image that has debugging enabled. For more information, see [Optional features for building images](#).

2. Edit the OEMInput.xml file to include the *OEMCustomAppFM.xml* feature manifest file that you created in the previous step. The XML will be similar to the following.

```

...
<AdditionalFMs>
  ...
  <AdditionalFM>%WPDKCONTENTROOT%\FMFiles\OEMCustomDriverFM.xml</AdditionalFM>
</AdditionalFMs>

```

3. In the <Features> section of the OEMInput.xml file, add the new DRIVER1 feature to the list of existing features.

```

<Features>
  <Microsoft>
    ...
  </Microsoft>
  <OEM>
    ...
    <Feature>DRIVER1</Feature>
  </OEM>
</Features>

```

### Generate, sign, and flash the image to the device

Complete the following steps to generate, sign, and flash the image.

1. Generate the image using ImgGen and the OEMInput.xml file that you customized in the previous step.

```
C:\>ImgGen Flash.ffu OEMInput.xml "%WPDKCONTENTROOT%\10\MSPackages"
```

2. Before you can sign images, you must first install the test OEM certificates on the PC by following the steps

in [Set up the signing environment](#).

3. Sign the generated catalog using the Sign.cmd with the **/pk** option.

```
C:\> Set SIGN_OEM=1  
C:\> Sign.cmd /pk TestSigned.cat
```

4. Sign the FFU with the signed catalog file using ImageSigner.

```
C:\> ImageSigner Sign Flash.FFU Flash.Cat
```

5. Flash the image to the phone using FFUTool.

```
C:\> FFUTool -Flash Flash.ffu
```

For more information about generating and flashing images, see [Building and flashing images](#).

### Verify that the KMDFDriver1 is on the device

Verify that the KMDFDriver1 is present on the device by using TShell.

1. Configure a TShell connection to test the image.
2. Establish a connection to the device using the **Open-device** TShell command. Provide the MAC address of the device.

```
PS C:\> Open-device 001122334455
```

3. Confirm that the KMDFDriver1 is on the device by using the **DirDevice** TShell command. The short form of the command, DirD, is shown.

```
PS C:\> DirD \KMDFDriver1.sys /s
```

4. You should see output similar to the following.

```
Volume in drive C is MainOS  
Volume Serial Number is 965E-2180  
Directory of C:\windows\system32\DRIVERS  
04/21/2014  05:23 PM           8864 KMDFDriver1.sys  
                  1 File(s)      8864 bytes  
Total Files Listed:  
                  1 File(s)      8864 bytes  
                  0 Dir(s)    409976832 bytes free
```

5. Enable debug output for KdPrint(), KdPrintEx(), and DbgPrint() messages by setting a registry key. Set the registry key so that it is persistent across reboots (for non-retail images) by using the **RegD Add** TShell command.

```
DEVICE C:\  
PS C:\Windows\system32> RegD Add "HKLM\SYSTEM\ControlSet001\Control\Session Manager\Debug Print Filter"  
/v DEFAULT /t REG_DWORD /d 0xFFFFFFFF  
  
The operation completed successfully.
```

6. Confirm that registry key was set by using the **RegD Query** TShell command.

```
DEVICE C:\  
PS C:\Windows\system32> RegD Query "HKLM\SYSTEM\ControlSet001\Control\Session Manager\Debug Print Filter"  
  
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Session Manager\Debug Print Filter  
DEFAULT REG_DWORD 0xffffffff
```

### Verify that the driver loads using Windows debug

To view the driver load process, complete the following steps to set a debug statement in the driver and connect the device to the kernel debugger.

1. Open the driver project in Visual Studio.
2. Add a debug break statement in Driver.c right after the variable declaration section of the DriverEntry routine.

```
...  
WDF_DRIVER_CONFIG config;  
NTSTATUS status;  
WDF_OBJECT_ATTRIBUTES attributes;  
  
// Debug break statement to stop loading of OS  
// Debugger must be attached to allow the device to boot  
__debugbreak();  
...
```

#### Warning

The debug break statement will stop the phone from booting unless a debugger is attached to the device.

3. Rebuild the project in Visual Studio.
4. If necessary, copy the updated .sys file to the directory that you will use to create to generate the package.
5. Then, repeat the steps described above to:
  - a. Generate an updated driver package.
  - b. Generate, sign, and flash the image.

#### Note

It is also possible to update the version of the driver by including the driver version in the INF file and then using the IUTool to deploy the updated driver to the phone. For more information about using IUTool, see [IUTool.exe: Update packages on a phone](#). This process is similar to the process that is used to create a driver update. For more information about that, see [Update a KMDF device driver](#).

6. Set up a KDNET over USB connection to the phone for debugging.
7. Start the Windows debugger.

```
windbg -k net:port=5000,key=1.2.3.4
```

8. After the device has booted, it will hit the break point and code execution will stop. The debugger will indicate that break point has been hit.
9. Set the symbol path to the location of the Windows Driver Kit symbols and reload the symbols.

```
.SYMPATH+ C:\SYMBOLS  
.reload /f
```

10. Set the source path to the location of your .c source code that you created earlier in Visual Studio.

```
.srcpath+ C:\KMDFDriver1\Source\
```

11. List the call stack by using the **k** command.

```
0: kd> k  
# Child-SP RetAddr Call Site  
00 85679c68 90fd2abe KMDFDriver1!DriverEntry+0xa  
01 85679ce0 90fd2b38 KMDFDriver1!FxDriverEntryWorker+0x6a  
02 85679d00 81770990 KMDFDriver1!FxDriverEntry+0x18  
*** ERROR: Symbol file could not be found. Defaulted to export symbols for ntkrnlmp.exe -  
03 85679d10 818f8004 nt!SeTokenIsAdmin+0x1790  
04 85679e10 8190d630 nt!KeFindConfigurationNextEntry+0x7c48  
05 85679e68 8185719e nt!KeHwPolicyLocateResource+0x4698  
06 85679e70 814c5e0c nt!IoReplacePartitionUnit+0x192  
07 85679e80 81460172 nt!IoAllocateIrp+0x57c  
08 85679ec8 00000000 nt!KiDispatchInterrupt+0x234a
```

The call stack is the chain of function calls that have led to the current location of the program counter. The top function on the call stack is the current function, and the next function is the function that called the current function, and so on.

12. Display symbol information associated with KMDFDriver1 using the **x** command.

```
0: kd> x KMDFDriver1!*  
90fd4750      KMDFDriver1!WdfDriverGlobals = 0x88ff4de8  
90fd4754      KMDFDriver1!WdfDriverStubDriverObject = 0x88fed768  
90fd4004      KMDFDriver1!__security_cookie_complement = 0x568f867  
90fd336c      KMDFDriver1!_gsfailure_xdata_end = <unknown base type 80000013>  
90fd33b4      KMDFDriver1!_memcpy_reverse_large_neon_xdata = <unknown base type 80000013>  
90fd30c0      KMDFDriver1!WPP_c0dfc8e231dc218098dc0c2ed20d6e73_Traceguids = struct _GUID [1]  
90fd30a0      KMDFDriver1!GUID_DEVINTERFACE_KmdfDriver1 = struct _GUID {e7a4cfb0-56d4-4234-bf60-  
fe627cb5e981}  
90fd474c      KMDFDriver1!WdfDriverStubDisplacedDriverUnload = 0x00000000  
90fd3370      KMDFDriver1!memcpy_xdata_prolog = <unknown base type 80000013>  
90fd3388      KMDFDriver1!_memcpy_forward_large_neon_xdata = <unknown base type 80000013>  
90fd4758      KMDFDriver1!WdfDriverStubOriginalWdfDriverMiniportUnload = 0x00000000
```

13. List information about the modules on KMDFDriver1 by using the **!m m** command.

```
0: kd> !m m KMDFDriver1 v  
Browse full module list  
start   end     module name  
90fd1000 90fd9000  KMDFDriver1  (private pdb symbols)  c:\kmfdfdriver1\KmdfDriver1.pdb  
    Loaded symbol image file: KMDFDriver1.sys  
    Image path: KMDFDriver1.sys  
    Image name: KMDFDriver1.sys  
    Browse all global symbols  functions  data  
    Timestamp:      Thu Jul 31 14:39:34 2014 (53DAB796)  
    CheckSum:       000037D2  
    ImageSize:      00008000  
    Translations:   0000.04b0 0000.04e4 0409.04b0 0409.04e
```

14. Load the WDF driver extensions using the **.load** command.

```
.load C:\Program Files (x86)\Windows Kits\10\Debuggers\x86\winext\Wdfkd.dll
```

15. Confirm that the WDF debug extension is loaded by using the **!wdfkd.help** WDK driver extension help command.

```
!wdfkd.help
```

16. Display information about the driver using the **!wdfkd.wdfdriverinfo** command.

```
0: kd> !wdfkd.wdfdriverinfo KMDFDriver1
-----
Default driver image name: KMDFDriver1
WDF library image name: Wdf01000
FxDriverGlobals 0x90584008
WdfBindInfo      0x913d8008
Version          v1.11
Library module   0x8283b528
ServiceName      \Registry\Machine\System\CurrentControlSet\Services\Wdf01000
ImageName        Wdf01000
-----
Driver Handles is NULL
```

17. End the debugging session using the **qd** quit and detach command.

```
0: kd> qd
```

# Feature groupings and constraints

1/25/2019 • 3 minutes to read

Feature groups and feature constraints allow additional logic to be added to the build system to support intelligent processing of the OEMInput XML.

## Feature groupings

Feature groupings allow for better management of optional features and allow the organization of packages for easier management. Feature grouping is used to implement feature constraints by Microsoft and optionally, by the OEM.

## Feature constraints

Feature constraints can be specified to avoid illogical or inappropriate build configurations.

Some settings are mutually exclusive and only one setting should be specified at a time. For example, consider the features, RELEASE\_PRODUCTION and RELEASE\_TEST. These features are mutually exclusive. This means that if RELEASE\_TEST is set, RELEASE\_PRODUCTION must not be set.

Constraints are grouped at the Microsoft and OEM level of features. OEMs cannot override Microsoft constraints. The following constraints are supported:

ELEMENT	DESCRIPTION	AT LEAST ONE FEATURE IS REQUIRED	FEATURES ARE MUTUALLY EXCLUSIVE
OneOrMore	One or more features must be specified.	true	false
ZeroOrOne	Either one feature or no features can be specified.	false	true
OneAndOnlyOne	One feature is required and only one feature can be specified.	true	true
ZeroOrMore	This is the default and indicates that there are no constraints. This option is only used to group features for publishing purposes and will be ignored during imaging.	false	false

The following XML sample illustrates the use of constraints to appropriately restrain the fake modem feature.

```

<Features>
  <Microsoft>
    <FeatureGroup Constraint="OneAndOnlyOne">
      <FeatureIDs>
        <FeatureID>MS_IMGFAKEMODEM</FeatureID>
        <FeatureID>MS_IMGNOFAKEMODEM</FeatureID>
      </FeatureIDs>
    </FeatureGroups >
  </Microsoft>
</Features>

```

The constraints in the sample specify that either IMGFAKEMODEM or IMGNOFAKEMODEM must be selected. Both values cannot be set at the same time. One and only one value must be set at a time. This constraint is required because the fake modem test feature must either be enabled or disabled.

The following XML sample illustrates the use of constraints to appropriately restrain the production build settings. This sample shows how multiple constraints can be associated with a single feature.

```

<Features>
  <Microsoft>
    <FeatureGroups>
      <FeatureGroup Constraint="ZeroOrOne">
        <FeatureIDs>
          <FeatureID>RELEASE_PRODUCTION</FeatureID>
          <FeatureID>MS_CODEINTEGRITY_PROD</FeatureID>
        </FeatureIDs>
      </FeatureGroup>
      <FeatureGroup Constraint="ZeroOrOne">
        <FeatureIDs>
          <FeatureID>RELEASE_PRODUCTION</FeatureID>
          <FeatureID>MS_DISABLETESTSIGNING</FeatureID>
        </FeatureIDs>
      </FeatureGroup>
    </Microsoft>
</Features>

```

When <ReleaseType>Production</ReleaseType> is set in the OEMInput file, this maps to RELEASE\_PRODUCTION. For more information about release type, see [OEMInput file contents](#).

The constraints in the sample specify that:

- Either RELEASE\_PRODUCTION or MS\_CODEINTEGRITY\_PROD can be selected, but they both may not be selected at the same time. This is because production code integrity is automatically enabled when RELEASE\_PRODUCTION is selected and therefore can't be manually enabled.
- Either RELEASE\_PRODUCTION or MS\_DISABLETESTSIGNING can be selected, but they both may not be selected at the same time. This is because test signing is automatically disabled when RELEASE\_PRODUCTION is selected and therefore can't be manually disabled.

The build options are more complex and are expressed in the following XML.

```

<FeatureGroup Constraint="OneAndOnlyOne">
  <FeatureIDs>
    <FeatureID>RELEASE_PRODUCTION</FeatureID>
    <FeatureID>MS_TEST</FeatureID>
    <FeatureID>MS_HEALTH</FeatureID>
    <FeatureID>MS_PRODUCTION</FeatureID>
    <FeatureID>MS_SELFHOST</FeatureID>
  </FeatureIDs>
</FeatureGroup>
<FeatureGroup Constraint="OneAndOnlyOne">
  <FeatureIDs>
    <FeatureID>RELEASE_PRODUCTION</FeatureID>
    <FeatureID>MS_PRODUCTION_CORE</FeatureID>
    <FeatureID>MS_TEST</FeatureID>
  </FeatureIDs>
</FeatureGroup>

```

These settings in the sample specify that PRODUCTION\_CORE is mutually exclusive with RELEASE\_PRODUCTION and TEST, but is not mutually exclusive with HEALTH, PRODUCTION, or SELFHOST.

For additional information about the build features, see [Optional features for building images](#).

## Implicit feature IDs

Implicit feature IDs are generated based on the XML input that is used to define features in feature manifest files. Feature constraints must use the implicit feature IDs. This section provides the mapping between the XML input values and the generated implicit feature IDs.

### Pre-defined release type implicit feature IDs

There are two pre-defined implicit feature IDs that can be used for feature constraints.

- RELEASE\_TEST
- RELEASE\_PRODUCTION

For more information on release types, see [Optional features for building images](#).

### OEM and Microsoft implicit feature IDs

For each OEM and Microsoft feature, implicit feature IDs are automatically created. This is done by prepending either *MS\_* for Microsoft or *OEM\_* for OEM defined features.

For example if an OEM creates a feature called TEST\_FEATURE1 using the XML shown below, the implicit Feature ID will be *OEM\_TEST\_FEATURE1*.

```

<Features>
  <OEM>
    <PackageFile Path="%oempackageroot%\test\" Name="Contoso.Test.MinTE.spkg">
      <FeatureIDs>
        <FeatureID>TEST_FEATURE1</FeatureID>
      </FeatureIDs>
    </PackageFile>
  </OEM>
</Features>

```

To create a feature constraint to make sure this test feature is only shipped with test release types, use the following XML.

```
<FeatureGroup Constraint="ZeroOrOne">
  <FeatureIDs>
    <FeatureID>RELEASE_PRODUCTION</FeatureID>
    <FeatureID>OEM_TEST_FEATURE1</FeatureID>
  </FeatureIDs>
</FeatureGroup>
```

## SOC implicit feature IDs

SOC features are pre-appended with SOC\_. For example if *DCD6000* is specified in the feature manifest XML, the implicit feature ID would be *SOC\_DCD6000*.

## SV implicit feature IDs

SV features are pre-appended with SV\_. For example if *CONTOSO* is specified in the feature manifest XML, the implicit feature ID would be *SV\_CONTOSO*.

## DEVICE implicit feature IDs

Device features are pre-appended with Device\_. For example if *BETA* is specified in the feature manifest XML, the implicit feature ID would be *DEVICE\_BETA*.

For more information about working with the SOC, SV and DEVICE attributes, see [Feature manifest file contents](#).

## Related topics

[Optional features for building images](#)

# Set device platform information

1/25/2019 • 6 minutes to read

To prepare for building an image, OEMs must perform the following tasks to specify required information for the targeted device platform:

- Set several device platform values in the SMBIOS system information structure on the devices.
- Build a *device platform* package.

The engineering flashing tool set provided by Microsoft compares the values in the SMBIOS system information structure with corresponding values in the device platform package before flashing an image. This check helps to ensure that an image can be flashed to a particular device only if it was built for the corresponding device platform. It is recommended that the flashing tools that OEMs create for their manufacturing environments do the same verification. For more information, see [Validating the device platform information before flashing an image to a device](#) in this topic.

**Note** This topic describes prerequisites for building an image that can be flashed to a device. OEMs must set additional device platform information including partner names, version numbers, and device names before an image is finalized for retail devices.

## Setting SMBIOS system information values

On each device platform, OEMs must ensure that the following values in the SMBIOS system information structure are set:

- PhoneManufacturer
- Family
- Product Name
- Version

These values may be set to defaults by the SoC vendor; OEMs must replace these with values for their device platform. OEMs may also need to set other SMBIOS values as specified by the SoC vendor. For more information about how to set or read these values, refer to documentation provided by the SoC vendor.

For more information about the expected sizes and data types for these values, refer to section 7.2 in the [System Management BIOS \(SMBIOS\) Reference Specification](#) (PDF).

**Important** The PhoneManufacturer setting must contain a code specified by Microsoft that corresponds to the OEM. This setting is used for targeting device updates, for connecting to the store-within-a-store in the Microsoft Store, and for Watson reports. The value must be a valid OEM ID. To get the valid OEM ID that applies to you, contact your Microsoft representative.

## Creating the device platform package

The device platform package contains just one file: an XML file named **OEMDevicePlatform.xml** that includes information specific to the device platform for which the image is being built. Every image must include a device platform package. OEMs must specify the device platform package by using the **OEMDevicePlatformPackages** element in a FM file that is included in the image.

To create the device platform package, first create an **OEMDevicePlatform.xml** file that contains the device

platform information in the required schema format. Then, create a package that includes this XML file.

## Creating the XML file

The OEMDevicePlatform.xml file contains a single **OEMDevicePlatform** element with the following child elements.

ELEMENT	DESCRIPTION
<b>MinSectorCount</b>	<p>This value specifies the minimum sectors that are expected on a device store. The imaging tool uses this value to ensure that the content will fit on the device. The actual sector count may be more than this minimum. The sector size is 512 bytes, as defined in the device layout package provided by Microsoft.</p>
<b>DevicePlatformID</b>	<p>This is one of the following strings that consists of values from the SMBIOS system information structure concatenated together, with each value separated by a period (.):</p> <ul style="list-style-type: none"><li>• <i>PhoneManufacturer.Family.Product Name</i></li><li>• <i>PhoneManufacturer.Family.Product Name.Version</i></li></ul> <p>OEMs can choose whether or not to include the <i>Version</i> value in this string.</p> <p>When device platform validation is enabled in the Microsoft flashing application, the values specified in this string will be compared with the corresponding values in the SMBIOS system information structure. For more information, see <a href="#">Use the flashing tools provided by Microsoft</a>.</p> <p>The device platform XML file can have only a <b>DevicePlatformID</b> or <b>DevicePlatformIDs</b> element, but not both.</p>
<b>DevicePlatformIDs</b>	<p>OEMs can set multiple device platform IDs using the following XML syntax, where each string has the same format as described for the <b>DevicePlatformID</b> element.</p> <div data-bbox="853 1507 1398 1641" style="border: 1px solid black; padding: 10px;"><pre>&lt;DevicePlatformIDs&gt;   &lt;ID&gt;Contoso.ContosoPhoneFamily.Z101._012&lt;/ID&gt;   &lt;ID&gt;Contoso.ContosoPhoneFamily.Z101._013&lt;/ID&gt;   &lt;ID&gt;Contoso.ContosoPhoneFamily.Z102&lt;/ID&gt; &lt;/DevicePlatformIDs&gt;</pre></div> <p>Device platform validation will succeed if any of the IDs match either the <i>PhoneManufacturer.Family.Product.Version</i> or the <i>PhoneManufacturer.Family.Product</i> specified.</p> <p>The device platform XML file can have only a <b>DevicePlatformID</b> or <b>DevicePlatformIDs</b> element, but not both.</p>

ELEMENT	DESCRIPTION
<b>AdditionalMainOSFreeSectorsRequest</b>	<p>Optional. This value specifies the number of sectors to add to the pool of storage reserved by the OS for future updates. There is no guarantee that the sectors specified by using the <b>AdditionalMainOSFreeSectorsRequest</b> element will always be available for OEM-specific updates; the sectors are instead added to a single pool of storage that is reserved for all updates from Microsoft and OEMs.</p>
<b>MainOSRTCDataReservedSectors</b>	<p>Optional. This value specifies the number of sectors to add to the pool of storage reserved by the OS for use by runtime configuration data during backup and restore operations. Up to 100 MB can be reserved.</p> <p>This following example demonstrates how to reserve 50 MB.</p> <pre data-bbox="849 729 1404 786">&lt;MainOSRTCDataReservedSectors&gt;102400&lt;/MainOSTCDataReservedSectors&gt;</pre>
<b>CompressedPartitions</b>	<p>Optional. Use this element to enable CompactOS in your mobile image.</p> <p>This element contains one or more child <b>Name</b> elements that specify which partitions to compress. Currently, the only supported partition is the Main OS partition, and the only supported value under this element is <b>MainOS</b>, as shown in the example below.</p> <pre data-bbox="849 1179 1118 1257">&lt;CompressedPartitions&gt;   &lt;Name&gt;MainOS&lt;/Name&gt; &lt;/CompressedPartitions&gt;</pre> <p><b>Important</b> The host computer where the image is created must be running Windows 10. You can also create the image on a host computer running Windows 8.1 or Windows Server 2012 R2 with <a href="#">KB3066427</a> installed.</p>

The following example demonstrates an OEMDevicePlatform.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<OEMDevicePlatform xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate">
  <MinSectorCount>20971520</MinSectorCount>
  <DevicePlatformID>Contoso.ContosoFamily.ContosoDevice.v1</DevicePlatformID>
  <AdditionalMainOSFreeSectorsRequest>204800</AdditionalMainOSFreeSectorsRequest>
</OEMDevicePlatform>
```

## Creating the package

To create a package that includes the OEMDevicePlatform.xml file, follow the guidance in [Creating packages](#). The following example demonstrates a device platform XML file that specifies a single device platform ID.

```

<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="urn:Microsoft.WindowsPhone/PackageSchema.v8.00"
    Owner="OEMName" OwnerType="OEM" Component="OEMDevicePlatform"
    ReleaseType="Production" Platform="DeviceName">
    <Components>
        <OSComponent>
            <Files>
                <File Source="OEMDevicePlatform.xml" DestinationDir="$(runtime.windows)\ImageUpdate"
                    Name="OEMDevicePlatform.xml"/>
            </Files>
        </OSComponent>
    </Components>
</Package>

```

Note the following details concerning this example:

- Be sure to replace the "OEMName" and "DeviceName" entries with appropriate values.
- The `$(runtime.windows)` string in the path for the **DestinationDir** attribute is a globally defined macro. The **DestinationDir** path must start with a globally defined macro for a directory. For more information about the **DestinationDir** attribute, see [Specifying files and registry entries in a package project file](#). For more information about macros, see [Primary elements and attributes of a package project file](#).

### Including the device platform package in the image

After the device platform package is created, it must be specified using the OEMDevicePlatformPackages element in a feature manifest file. For more information, see [Feature manifest file contents](#).

## Validating the device platform information before flashing an image to a device

To help ensure that an image about to be flashed to a device was actually designed for that device, the flashing tool set created by OEMs should check the *Manufacturer*, *Family*, and *Product Name* SMBIOS system information structure values on the device and compare these values against the *Manufacturer.Family.Product Name* portion of the **DevicePlatformID** string in the device platform package. The flashing tool should proceed with the flashing process only if these values match. The flashing tool can optionally verify that the *Version* value also matches, but this is not required. For more information, see [Developing custom OEM flashing tools](#).

By default, the device-side UEFI flashing application provided by Microsoft validates that the device platform information in SMBIOS matches the device platform information in the image. For more information, see [Use the flashing tools provided by Microsoft](#).

When it is necessary to migrate a phone to a new set of device platform values, Microsoft recommends the following process:

- Build a transition image that contains the new SMBIOS values but still references the old **DevicePlatformID** string in the device platform package.
- Flash this image to the phone. This process overwrites the SMBIOS values on the phone with the new SMBIOS values.
- In the image definition, update the **DevicePlatformID** string in the device platform package to match the new SMBIOS values. This enables the image to be flashed to the phone.

## Related topics

[Developing custom OEM flashing tools](#)

Use the flashing tools provided by Microsoft

# Sign a full flash update (FFU) image

1/25/2019 • 5 minutes to read

You must cryptographically sign an FFU image before you can deploy it to a device. This step is required to help prevent tampering and malicious attacks. You can only flash images to a Windows 10 Mobile device after they are properly signed.

The OEM must sign the FFU image using a certificate that chains to the UEFI PK boot key that an OEM provisions.

During development and testing, a test certificate is used to sign the FFU image instead of a retail certificate. Use the secure boot test tool to provision the appropriate test certificates to the device

For retail phones, retail certificates are provisioned on the device using retail secure boot tools.

## Test signing images using Windows Imaging and Configuration Designer (ICD)

After you use the secure boot test tool to provision the appropriate test certificates to a phone, you can use the Windows Imaging and Configuration Designer (ICD) to generate test signed images. Windows ICD will automatically sign images using the correct test certificates. This is the recommended process, as it is the easiest method to prepare properly signed images. For more information, see [Windows ICD](#).

## Retail signing FFU images

Before images can be shipped on a retail device, they must be signed by Microsoft. Use the process described in this section to create a retail signed image FFU file that can be flashed to retail devices.

1. Before you can sign binaries, you must first install the test OEM certificates on the PC by following the steps in [Set up the signing environment](#).
2. Add the directory for the sign.cmd script that is located in %WPDKCONTENTROOT%\tools\bin\i386 to your path using the path command.

```
C:\> PATH = %PATH%;%WPDKCONTENTROOT%\Tools\bin\I386
```

3. Open a developer prompt with administrator rights in the directory that contains the output from the image generation process.
4. Confirm that you have the latest version of the kit that includes updated versions of imagesigner.exe and imagecommon.dll by typing the following command. Confirm that the truncate option is displayed.

```
C:\> ImageSigner /?
Usage:
  imagesigner sign <FFU> <catalog file>
  imagesigner getcatalog <FFU> <catalog file>
  imagesigner truncate <FFU> <truncated FFU>
```

If you do not have a recent version of the kit that includes the updated ImageSigner, download and install the latest kit.

5. Extract the first one MB of the FFU image which contains the FFU catalog and associated metadata using

the truncate option.

```
C:\> ImageSigner TRUNCATE OEM.ffu OEM.trunc
```

6. Extract the catalog from the truncated FFU image in Step 5.

```
C:\> ImageSigner GETCATALOG OEM.trunc OEM.cat
Platform ID: <OEMID>.QC8960.P728
Successfully extracted catalog.
C:\>
```

7. Validate that the <OEMID> returned in Step 6 is identical to the OEMID assigned to you by Microsoft.
8. Create and submit a TFS ticket to the Microsoft Service Desk to sign the FFU catalog. Ensure that you attach the truncated FFU image that you created in Step 4.
  - a. Open an *Update* type ticket.
  - b. For the ticket title, enter *FFU Catalog Signing Request from <OEMID>*. Specify your OEM ID in the ticket title where <OEMID> is shown.
  - c. Set the **Category** to *Ingestion and Publishing*.
  - d. Set the **Issue Type** to *New*.
9. Microsoft will process the request and resolve the ticket back to the OEM. Within the resolved ticket, Microsoft will include a retail signed version of the FFU catalog extracted from the FFU truncated image attached to the original ticket. For this example, assume that Microsoft returns the signed FFU catalog named OEM.cat.
10. Sign the FFU with the retail Microsoft signed catalog file using ImageSigner.

```
C:\> ImageSigner SIGN OEM.ffu OEM.cat
```

11. Flash the retail signed retail image on to a device and verify that it behaves as expected.

```
C:\> FFUTool -flash OEM.ffu
```

12. After the retail signed image has been tested it can be used to flash to retail devices. For more information see [Flashing tools](#) and [Use the flashing tools provided by Microsoft](#).

**Important** Secure all of the retail signed binaries using industry best practices.

## Test signing images manually

If your development environment includes work outside of Windows ICD, you can manually sign images using sign.cmd. After the test certificate has been provisioned on the device using the secure boot test tool, you can use the /pk option of the sign.cmd tool to sign images, by completing the following steps.

1. Before you can sign binaries, you must first install the test OEM certificates on the PC by following the steps in [Set up the signing environment](#).
2. Add the directory for the sign.cmd script that is located in %WPKCONTENTROOT%\tools\bin\i386 to your path using the path command.

```
C:\> PATH = %PATH%;%WPDKCONTENTROOT%\Tools\bin\I386
```

3. Open a developer prompt with administrator rights in the directory that contains the output from the image generation process.
4. Extract the catalog of the unsigned FFU file.

```
C:\> ImageSigner GETCATALOG TestSigned.FFU TestSigned.Cat
```

5. Sign the catalog using the /pk option.

```
C:\> Set SIGN_OEM=1  
C:\> Sign.cmd /pk TestSigned.cat
```

6. Sign the FFU with the signed catalog file using ImageSigner.

```
C:\> ImageSigner SIGN TestSigned.FFU TestSigned.Cat
```

## Creating custom signed images

**Note** Some OEMs may want to use custom keys to manage images. This option is more complex and is not recommended.

When an image is created, an associated catalog file is also created. This catalog file is signed and then used by the ImageSigner tool to sign the FFU image. Sign the .cat file with a certificate that chains to the UEFI boot keys that are provisioned by the OEM. Different certificates are used for test and retail signing.

Sign the catalog file using an appropriate certificate by performing the following steps.

1. Open a developer prompt with administrator rights in the directory that contains the output from the image generation process. For more information, see [Building an image using ImgGen.cmd](#).
2. Test sign a catalog file named `TestRetailSigned.cat` using a certificate named `TestCertName.pfx` by typing the following command.

```
C:\> SignTool sign /f TestCertName.pfx TestRetailSigned.cat
```

**Important** You should only use the signtool.exe with the local file `/f` option internally in a development test environment. When a hardware security modules (HSM) is used, the `/csp` option is used to specify the cryptographic service provider (CSP) that contains the private key container. You should follow industry best practices when signing image update packages for final distribution.

3. Create a signed .ffu file from the unsigned .ffu file and the matching signed .cat file using ImageSigner.exe tool.

```
C:\> ImageSigner SIGN TestRetailSigned.FFU TestRetailSigned.Cat
```

## ImageSigner syntax reference

```
ImageSigner {SIGN|GETCATALOG|TRUNCATE} FFUfile CatalogFile|TruncatedFFU
```

- **SIGN** – The **SIGN** action is used for signing an FFU file.
- **GETCATALOG** – The **GETCATALOG** action extracts a catalog from an FFU file and writes it to a catalog file. This option can be used to determine if an FFU was prepared properly, by examining the extracted catalog file by using file properties or tools such as SignTool.
- **TRUNCATE** – The truncate action is used to create a truncated FFU.
  - *FFUFile* – The path to the FFU image file.
  - *CatalogFile* – The path to the catalog file.
  - *TruncatedFFU* – The path to the truncated FFU file.

## Related topics

[Building a phone image using ImgGen.cmd](#)

# Use the flashing tools provided by Microsoft

1/25/2019 • 7 minutes to read

Microsoft provides a tool set for flashing images to devices. This tool set includes ffutool.exe, a command-line tool that runs on the development computer, and ffuloader.efi, a device-side UEFI flashing application. This topic describes how to set up your device and development computer to use this tool set, and provides usage instructions.

## Important

- In Windows 10, version 1607, ffutool.exe is installed by the Assessment and Deployment Kit (ADK). In earlier versions of Windows 10, ffutool.exe is installed by the Windows Driver Kit (WDK), Enterprise WDK (EWDK), or Windows Hardware Lab Kit (HLK).
- The device-side UEFI flashing application from Microsoft is automatically included in all images. This application must be included in all retail devices.
- For flashing images to devices during manufacturing, OEMs can build their own flashing tools by using the information provided in [Developing custom OEM flashing tools](#) or by using ffutool.exe. If you use ffutool to flash your image, it might be slower than other flashing tools.

## Initial device-side setup

To prepare a device for flashing with a Windows 10 Mobile image, perform the following steps:

1. Use tools provided by the SoC vendor to get a UEFI onto any device that needs to be flashed. Devices must be bootable at least to the UEFI for flashing to work.

Typically, this is performed with the eMMC software downloader. This UEFI should be the same UEFI that is included in device images.

2. After a bootable UEFI is in place, the flashing application (ffuloader.efi) and supporting simple I/O driver (efisimpleio.dll) must be added to the device and run on boot. To do this, run the apply-ffubinaries.bat script while the device is connected to the development computer. All of these files are provided the kit in %ProgramFiles(x86)%\Windows Kits\10\Tools\bin\i386.

The apply-ffubinaries.bat script copies ffuloader.efi and efisimpleio.dll to the root of the ESP directory on the device and sets up the boot configuration database to immediately enter flashing mode on boot. This script requires bcdedit.exe in the path.

**Note** These steps only need to be performed once on each device. After these steps are completed, you will be able to flash different images to the device

## Host-side setup

Flashing on the host side is performed by using a connection established with WinUSB, the Microsoft generic USB device driver. The necessary drivers are included by default in Windows 8 and later.

In Windows 7, the necessary drivers can be installed from Windows Update. To configure a Windows 7 computer to install the necessary drivers:

1. Click Start.
2. Type **Device Installation Settings**.

3. Select **Yes, do this automatically (recommended)**.

4. Click **Save Changes**.

## Flashing procedure

After the device-side and host-side setup is complete, perform the following steps to flash a device:

1. Boot the device into the FFU download mode while it is connected to the host computer. There are several ways to force the device into the FFU download mode during boot:

- Include the Microsoft.BCD.Lab.spkg package in your image by specifying the **LABIMAGE** optional feature when generating the test image. When this package is included in the test image, the device automatically enters the FFU download mode when it is booted. For more information about generating an image, see [Building a phone image using ImgGen.cmd](#).
- To force the device into the FFU download mode manually, press and release the power button to boot the device, and then immediately press and hold the volume up button. This option is available only after an initial FFU has been flashed to the device.

2. Run ffutool.exe from the command line to flash an image. This program is in %ProgramFiles(x86)%\Windows Kits\10\Tools\bin\i386. The following is a usage example.

```
ffutool -flash <FFU file>
```

The following table describes the command-line parameters for ffutool.exe.

PARAMETER	DESCRIPTION
<b>-flash FFU file</b>	Specifies the path to the FFU file to be flashed to the device.
<b>-skip</b>	For devices that include the <b>LABIMAGE</b> optional feature to automatically boot into the FFU download mode, this parameter boots into the main OS (skipping the FFU download mode).

You can flash more than one device at a time by using ffutool.exe. To do this, make sure that all devices are connecting before running ffutool.exe. Also, we recommend that you use a USB card that contains a dedicated root hub per port so an issue flashing one device does not affect all devices.

**Note** Flashing speed will decrease as you add devices.

## Validations performed by the Microsoft flashing tool

Before flashing an image, the device-side UEFI flashing application provided by Microsoft performs the following validations on the image:

- The application validates the image signatures against the Platform Key (PK) certificate and the Microsoft Windows Phone Production PCA 2012 certificate (for retail images) or Microsoft Test Root Authority certificate (for non-retail images).
- The application validates the hashes of each chunk of data in the image against the table of hashes signed by the catalog file.
- The application validates that the image supports the current device platform. To perform this validation,

the application compares several values in the SMBIOS system information structure with corresponding values in the device platform package. These checks help to ensure that an image can be flashed to a particular device only if it was built for the corresponding device platform. More details about these checks are provided below.

### Device platform validation checks

To validate that the image supports the current device platform before flashing, the application performs the following tasks:

1. It retrieves the **DevicePlatformID** string from the device platform package in the image that is being flashed. For more information about this string, see [Set device platform information](#).
2. If the string has the format *Manufacturer.Family.Product Name.Version* and all four values in the string match the corresponding *Manufacturer*, *Family*, *Product Name*, and *Version* SMBIOS values on the device, the platform validation succeeds and the flashing operation continues.
3. If the string has the format *Manufacturer.Family.Product Name* and all three values in the string match the corresponding *Manufacturer*, *Family*, and *Product Name* SMBIOS values on the device, the platform validation succeeds and the flashing operation continues.
4. Otherwise, the platform validation fails and the image is not flashed to the device.

In a non-retail image, OEMs can disable the device platform validation for flashing by adding the DISABLE\_FFU\_PLAT\_ID\_CHECK feature to the OEMInput file that is used to generate the image.

**Important** The device platform validation for flashing must not be disabled in retail images.

## FFU tool error codes

When using the FFU tool to flash an image to your device, you may encounter an error as shown below.

```
Error: Failed to flash with device error { 0x18, 0x0, 0x0, 0x2, 0xa, 0x5 }
```

The first hexadecimal number is an event code that indicates the type of flashing failure. The following table provides a description of the FFU tool flashing errors.

### Common errors

ERROR CODE	DESCRIPTION
0xC	<p>While applying the image to disk, a read failed to return all of the blocks specified in the current data descriptor.</p> <p>This error is typically caused by a corrupted image. The image will normally need to be rebuilt. For more information, see <a href="#">Building a phone image using ImgGen.cmd</a>.</p>
0x18	<p>While initializing hash checks, a catalog signature check failed.</p> <p>When this error occurs, it is typically related to the signing of the image. For more information, see <a href="#">Sign a full flash update (FFU) image</a>.</p>

ERROR CODE	DESCRIPTION
0x1C	<p>One or more write descriptors refer to invalid disk locations.</p> <p>This error typically indicates that the image was built for a phone with more storage than the current phone actually has. Either locate the proper image or update the image that you have by reducing the MinSectorCount specified in OEMDevicePlatform.xml. For more information, see <a href="#">Set device platform information</a>.</p>

## Additional errors

ERROR CODE	DESCRIPTION
0x8	An invalid binary manifest header was read while preparing to flash the device.
0x9	The application failed to allocate enough memory to copy the disk write descriptor table.
0xB	Before writing data, the flashing application failed to read all the disk write descriptors.
0xD	While applying the image to disk, a block write operation failed.
0xE	A packet read by the flashing application contained an invalid checksum.
0xF	While preparing to flash the image, the flashing application was unable to read the full security header.
0x10	While preparing to flash the image, the flashing application read an invalid security header.
0x11	While preparing to flash the image, the flashing application failed to read the expected amount of padding after the security header.
0x12	While preparing to flash the image, the flashing application read an invalid image header.
0x13	While preparing to flash the image, the flashing application failed to read the expected amount of padding after the image header.

ERROR CODE	DESCRIPTION
0x14	While preparing to apply the image to disk, the block flasher failed to buffer enough bytes in the stream to flash safely.
0x15	While applying the image to disk, the block flasher reached the end of the data stream unexpectedly. This indicates an image was built incorrectly.
0x16	The platform ID specified in the image does not match the ID of the device to be flashed.
0x17	While reading image data, a hash check failed.
0x1A	Failed to acquire a handle to the UEFI firmware de-synchronization event.
0x1B	Failed to query the BCD for the platform ID check settings.
0x1D	The image does not have Reset Protection enabled or an unsupported Reset Protection image was used.
0x20	The image cannot be flashed on removable media.
0x21	Cannot use an optimized flashing method.

## Related topics

[Building and flashing images](#)

# IUTool.exe: Update packages on a device

1/25/2019 • 2 minutes to read

The Windows Driver Kit (WDK) includes a tool for updating packages on a device or to add a new package to a device. (IUTool.exe). This tool is available under %WPDKCONTENTROOT%\Tools\bin\i386.

## Using IUTool.exe to update packages on a device

IUTool.exe must be used in a command-line window that is opened as an administrator. The command-line syntax for IUTool.exe is the following.

```
IUTool -p <path to packages>
```

The following table describes the command-line parameters for IUTool.exe.

PARAMETER	DESCRIPTION
<code>-p path to packages</code>	<p>Specifies one or more packages to update on the device or to add to the device. The path to packages parameter can have the following formats:</p> <ul style="list-style-type: none"><li>• To update or add a single package, specify the full path to the package on the development computer.</li><li>• To update or add multiple packages, specify a semicolon-delimited list of packages on the development computer. For example:</li></ul> <div style="border: 1px solid #ccc; padding: 5px; margin-left: 20px;"><pre>IUTool -p C:\ContosoPackages\Contoso.Device.SampleDriver.spkg;C:\ContosoPackages\Contoso.Device.SampleApplication.spkg</pre></div> <ul style="list-style-type: none"><li>• To update or add an entire directory of packages, specify the path to the directory. For example:</li></ul> <div style="border: 1px solid #ccc; padding: 5px; margin-left: 20px;"><pre>IUTool -p C:\ContosoPackages</pre></div>

### Package versioning

If the specified package already exists on the device, the new version of the package must have a higher version than the package currently on the device or the update will fail. To specify the version for a package, use the /version command-line parameter for PkgGen.exe when generating the package. For more information, see [Command-line arguments for package generator](#).

### Using GetDULogs.exe to get package update logs from a device

Use GetDULogs.exe to get package update logs from a device.

```
GetDULogs -o <output file path>
```

For more info, see [GetDULogs: Get package update logs](#).

# IUTool error codes

1/25/2019 • 8 minutes to read

The following are error codes from IUTool.exe. For more info, see [IUTool.exe: Update packages on a device](#).

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x00000000	SUCCESS	Success (no error)
0x80004001	E_NOTIMPL	
0x80004002	E_NOINTERFACE	
0x80004003	E_POINTER	
0x80004004	E_ABORT	
0x80004005	E_FAIL	
0x8000FFFF	E_UNEXPECTED	
0x80040154	E_CLASSNOTREG	Class not registered
0x80070002	ERROR_FILE_NOT_FOUND	
0x80070003	E_PATH_NOT_FOUND	
0x80070005	E_ACCESSDENIED	
0x80070006	E_HANDLE	
0x80070008	ERROR_NOT_ENOUGH_MEMORY	
0x80070009	ERROR_INVALID_BLOCK	
0x8007000B	ERROR_BAD_FORMAT	
0x8007000D	ERROR_INVALID_DATA	

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x8007000E	ERROR_OUTOFMEMORY	
0x80070013	ERROR_WRITE_PROTECT	
0x80070015	ERROR_NOT_READY	
0x80070017	ERROR_CRC	
0x80070019	ERROR_SEEK	
0x8007001F	ERROR_GEN_FAILURE	
0x80070020	ERROR_SHARING_VIOLATION	
0x80070021	ERROR_LOCK_VIOLATION	
0x80070026	ERROR_HANDLE_EOF	
0x80070032	ERROR_NOT_SUPPORTED	
0x80070057	E_INVALIDARG	An argument does not meet the contract of the method
0x80070070	ERROR_DISK_FULL	Free more space
0x80070076	ERROR_INVALID_VERIFY_SWITCH	
0x8007007A	ERROR_INSUFFICIENT_BUFFER	
0x8007007B	ERROR_INVALID_NAME	
0x8007007E	ERROR_MOD_NOT_FOUND	
0x8007007F	ERROR_PROC_NOT_FOUND	
0x80070098	ERROR_TOO_MANY_MUXWAITERS	
0x800700B7	ERROR_ALREADY_EXISTS	

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x800700C1	ERROR_BAD_EXE_FORMAT	Normally an incorrect USS detection.dll
0x800700CE	ERROR_FILENAME_EXCED_RANGE	
0x80070160	ERROR_FAIL_RESTART	
0x800701e3	ERROR_DEVICE_HARDWARE_ERRO R	Free up eMMC space for update. There is something wrong with SD Card, or Timed out waiting for SD Card.
0x80070216	ERROR_ARITHMETIC_OVERFLOW	
0x80070241	ERROR_INVALID_IMAGE_HASH	
0x800703E5	ERROR_IO_PENDING	
0x800703FA	ERROR_KEY_DELETED	
0x80070422	ERROR_SERVICE_DISABLED	
0x80070423	ERROR_CIRCULAR_DEPENDENCY	
0x80070424	ERROR_SERVICE_DOES_NOT_EXIST	
0x80070426	ERROR_SERVICE_NOT_ACTIVE	
0x80070428	ERROR_EXCEPTION_IN_SERVICE	
0x8007042B	ERROR_PROCESS_ABORTED	
0x8007042E	ERROR_SERVICE_START_HANG	
0x80070435	ERROR_SERVICE_NEVER_STARTED	No attempts to start the service have been made since the last boot
0x80070459	ERROR_NO_UNICODE_TRANSLATIO N	

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x80070486	ERROR_NO_MORE_USER_HANDLES	
0x80070490	ERROR_NOT_FOUND	This can happen if there is leftover registration information on the PC. To resolve this issue, remove all registered mobile devices from the Devices and Printers control panel.
0x800704C7	ERROR_CANCELLED	
0x800704EC	ERROR_ACCESS_DISABLED_BY_POLICY	
0x80070522	ERROR_PRIVILEGE_NOT_HELD	
0x80070525	ERROR_NO_SUCH_USER	
0x8007053C	ERROR_BAD_INHERITANCE_ACL	
0x8007054E	ERROR_INTERNAL_DB_CORRUPTION	
0x80070570	ERROR_FILE_CORRUPT	
0x80070571	ERROR_DISK_CORRUPT	
0x800705AA	ERROR_NO_SYSTEM_RESOURCES	
0x800705B4	E_TIMEOUT	This operation returned because the timeout period expired.
0x8007065D	ERROR_DATATYPE_MISMATCH	
0x800706B5	RPC_S_UNKNOWN_IF	
0x800706BA	RPC_S_SERVER_UNAVAILABLE	
0x800706BE	RPC_S_CALL_FAILED	
0x800706BF	RPC_S_CALL_FAILED_DNE	

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x800706D9	EPT_S_NOT_REGISTERED	
0x80072EE2	ERROR_WINHTTP_TIMEOUT	
0x80072EE4	ERROR_WINHTTP_INTERNAL_ERROR	
0x80072EE5	ERROR_WINHTTP_INVALID_URL	
0x80072EE6	ERROR_WINHTTP_UNRECOGNIZED_SCHEME	
0x80072EE7	ERROR_WINHTTP_NAME_NOT_RESOLVED	
0x80072EE9	ERROR_WINHTTP_INVALID_OPTION	
0x80072EEC	ERROR_WINHTTP_SHUTDOWN	
0x80072EEF	ERROR_WINHTTP_LOGIN_FAILURE	
0x80072EF1	ERROR_WINHTTP_OPERATION_CANCELLED	
0x80072EF3	ERROR_WINHTTP_INCORRECT_HANDLE_STATE	
0x80072EFD	ERROR_WINHTTP_CANNOT_CONNECT	
0x80072EFE	ERROR_WINHTTP_CONNECTION_ABORTED	
0x80072EFF	ERROR_WINHTTP_CONNECTION_RESET	
0x80072F05	ERROR_WINHTTP_SEC_CERT_DATE_INVALID	
0x80072F06	ERROR_WINHTTP_SEC_CERT_CN_INVALID	

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x80072F0C	ERROR_WINHTTP_CLIENT_AUTH_CERT_NEEDED	
0x80072F0D	ERROR_WINHTTP_INVALID_CA	
0x80072F30	ERROR_WINHTTP_NO_CM_CONNECTION	
0x80072F76	ERROR_WINHTTP_HEADER_NOT_FOUND	
0x80072F78	ERROR_WINHTTP_INVALID_SERVER_RESPONSE	
0x80072F7A	ERROR_WINHTTP_INVALID_QUERY_REQUEST	
0x80072F7C	ERROR_WINHTTP_REDIRECT_FAILED	
0x80072F7D	ERROR_WINHTTP_SECURE_CHANNEL_ERROR	
0x80072F89	ERROR_WINHTTP_SECURE_INVALID_CERT	
0x80072F8F	ERROR_WINHTTP_SECURE_FAILURE	
0x80072F98	ERROR_WINHTTP_RESPONSE_DRAIN_OVERFLOW	
0x80073B01	ERROR_MUI_FILE_NOT_LOADED	
0x80090305	ERROR_BAD_ACCESSOR_FLAGS	
0x80092002	CRYPT_E_BAD_ENCODE	
0x80092003	CRYPT_E_FILE_ERROR	
0x80096004	TRUST_E_CERT_SIGNATURE	

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x80096005	TRUST_E_TIME_STAMP	
0x80096010	TRUST_E_BAD_DIGEST	
0x800B0003	TRUST_E SUBJECT FORM UNKNOWN	Most certificate issues can be fixed in server publishing, unless the user has changed anything on the device side.
0x800B0100	TRUST_E_NOSIGNATURE	
0x800B0101	CERT_E_EXPIRED	
0x800B010A	CERT_E_CHAINING	
0x800B010B	TRUST_E_FAIL	
0x801881C0	DUA_ERR_BASE	This is a DUA error base and not a real error code.
0x801881C3	DUA_E_NEWER_UPDATE_EXISTS	An update with a higher revision number exists.
0x801881C4	DUA_E_UPDATE_ALREADY_IN_EVALUATION	An update is being updated while asking for evaluation.
0x801881C5	DUA_E_DEPENDENT_UPDATE_MISSING	A dependent update is missing.
0x801881C6	DUA_E_DUPLICATE_ATTRIBUTE	An attribute is specified more than once.
0x801881C7	DUA_E_MISSING_ATTRIBUTE	A required attribute is not found.
0x801881C9	DUA_E_INVALIDATTRIBUTEVALUE	Attribute value is invalid.
0x801881CA	DUA_E_XMLSTRINGTOOLONG	String/text is too long
0x801881CB	DUA_E_ATTRIBUTE_MISMATCH	Attribute mismatch

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x801881CD	DUA_E_OUT_OF_DISK_SPACE	Not enough disk space for the operation.
0x801881CF	DUA_E_CONNECTION_MANAGER	Call to Connection Manager API failed.
0x801881D0	DUA_E_INVALID_HASH	Given hash is not match to the hash generated from file.
0x801881D1	DUA_E_INVALID_EULA	An invalid or missing EULA was detected.
0x801881D2	DUA_E_INVALID_EXTENDED_UPDATE_INFO	Invalid extended update info was detected.
0x801881D3	DUA_E_MAX_SYNCUPDATES_CALLS	SyncUpdate call reaches the maximum scan limit.
0x801881D4	DUA_E_AUTO_SCAN_OFF	Scan cannot run because auto scan is turned off.
0x801881D5	DUA_E_XML_CONTENT_TOO_LARGE	XML content in string is larger than maximum allowed.
0x801881D6	DUA_E_SDCARD_REMOVED	An SD card was removed during update after the download was started to the SD card.
0x801881D7	DUA_E_SDCARD_DISABLED	The SD card is disabled by policy.
0x801881D8	DUA_E_SDCARD_HWFAILURE	The SD card hardware failed.
0x801881E1	DUA_E_SOAPFAULT_UNKNOWN	Unknown SOAP fault
0x801881E2	DUA_E_SOAPFAULT_COOKIEEXPIRED	CookieExpired SOAP fault
0x801881E3	DUA_E_SOAPFAULT_INVALIDCOOKIE	InvalidCookie SOAP fault
0x801881E4	DUA_E_SOAPFAULT_CONFIGCHANGED	ConfigChanged SOAP fault

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x801881E5	DUA_E_SOAPFAULT_SERVERCHANNEL	ServerChanged SOAP fault
0x801881E6	DUA_E_SOAPFAULT_INVALIDAUTHORIZATIONCOOKIE	InvalidAuthorizationCookie SOAP fault
0x801881E7	DUA_E_SOAPFAULT_REGISTRATIONREQUIRED	RegistrationRequired SOAP fault
0x801881E8	DUA_E_SOAPFAULT_INTERNALSERVERERROR	InternalServerError SOAP fault
0x801881E9	DUA_E_SOAPFAULT_INVALIDpMETERS	Invalidpmeters SOAP fault
0x801881EA	DUA_E_SOAPFAULT_REGISTRATIONNOTREQUIRED	RegistrationNotRequired SOAP fault
0x801881EB	DUA_E_SOAPFAULT_SERVERBUSY	ServerBusy SOAP fault
0x801881EC	DUA_E_SOAPFAULT_FILELOCATIONCHANGED	FileLocationChanged SOAP fault
0x801881EE	DUA_E_SOAPFAULT_MALFORMEDMESSAGE	Malformed SOAP message
0x801881EF	DUA_E_CABDOWNLOAD_HTTPSTATUSFAILED	Failed to download content from the server.
0x80188200	DUA_ERR_REPORTINGERROR_BASE	Definition. The error code itself actually not used.
0x80188201	DUA_E_REPORTINGFAILED	ReportEventBatch web service call returned FALSE
0x80188203	DUA_E_INVALIDSAMPLINGRATE	Sampling rate in redirect cab is zero.
0x80188210	DUA_ERR_CLIENTERROR_BASE	Definition; the error code itself actually not used.
0x80188215	DUA_E_CLIENT_INVALIDBUTTONID	The invalid button ID is used.

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x80188216	DUA_E_CLIENT_REGKEY_INVALID	Registry key value is not valid.
0x80188217	DUA_E_CLIENT_PLUGINID_NOT_FOUND	The plugin ID was not found.
0x80188220	DUA_ERR_INSTALLATION_BASE	Definition; the error code itself actually not used.
0x80188226	DUA_E_BATTERY_LOW	Cannot stage update because battery is low.
0x80188227	DUA_E_UPDATE_NOT_STAGED	Update not staged.
0x80188228	DUA_E_USS_UPDATE_FOUND	A USS update was detected
0x8018822A	DUA_E_INSTALL_INVALIDUPDATESSTATE	Update was not committed
0x8018822B	DUA_E_INSTALL_UNEXPECTED_COMMIT_RETURN	Image Update Commit API unexpectedly returned S_OK.
0x8018822C	DUA_E_UPDATE_NOT_COMMITED	Device rebooted before IU Commit was called.
0x8018822D	DUA_E_USS_UPDATE_FOUND_RESET_REQUIRED	A USS update was installed and a reboot is required for it to take effect.
0x80188230	DUA_ERR_DATA_MIGRATION_BASE	Definition; the error code itself actually not used.
0x80188231	DUA_E_COLD_BOOT_REQUIRED	A data migrator failed and requested a cold boot.
0x80188240	DUA_ERR_UPDATE_STORE_BASE	Definition; the error code itself actually not used.
0x80188241	DUA_E_UPDATE_NOT_FOUND	Update not found.
0x80188242	DUA_E_UPDATE_PROPERTY_NOT_FOUND	Update property not found.

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x80188243	DUA_E_UPDATE_ALREADY_EXISTS	Update already exists.
0x80188244	DUA_E_EULA_ALREADY_EXISTS	EULA already exists
0x80188245	DUA_E_EULA_NOT_FOUND	EULA not found.
0x80188246	DUA_E_BUNDLE_NOT_FOUND	Bundle not found.
0x80188247	DUA_E_EXTENDED_PROPERTY_NOT_FOUND	Extended property not found.
0x80188248	DUA_E_GETEULA	GetEULAs call failed.
0x80188250	DUA_ERR_SESSION_BASE	Definition; the error code itself actually not used.
0x80188251	DUA_E_INVALID_SESSIONID	The specified DuaSessionID is not valid.
0x80188252	DUA_E_DIFF_SESSION_IN_PROGRESS	There is a DUA session already in progress with a different DuaSessionID.
0x80188253	DUA_E_NO_SESSION_IN_PROGRESS	There is no DUA session in progress.
0x80188254	DUA_E_CANNOT_CLOSE_SESSION	We cannot close this session because a client has successfully joined the current session.
0x80188255	DUA_E_ALL_RETRY_CONSUMED	All retries are consumed.
0x80188256	DUA_E_SERVICE_IS_SHUTTING_DOWN	Service is shutting down and requested operation can't be performed.
0x80188257	DUA_E_SESSION_TIMEDOUT	The active session has timed out.
0x80188260	DUA_E_DOWNLOAD_BASE	Definition; the error code itself actually not used.

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x80188261	DUA_E_DOWNLOAD_NOT_CREATED	The download request has not been created.
0x80188262	DUA_E_DOWNLOAD_NOT_IDLE	Update Download Manager is not ready yet.
0x80188263	DUA_E_DOWNLOAD_USER_CANCELLED	Update Download is cancelled by user.
0x80188264	DUA_E_DOWNLOAD_NEEDUSERAGREE	Need user to agree to proceed with download.
0x80188265	DUA_E_DOWNLOAD_NEEDOVERCELLULAR	Need user agreement to download updates via cellular.
0x80188266	DUA_E_DOWNLOAD_USERPOSTPONE	User postponed download
0x80188267	DUA_E_DOWNLOAD_ENGINEEXIT	Download handler is stopped to wait for BTS status due to engine exit.
0x80188269	DUA_E_DOWNLOAD_OUT_OF_DISK_SPACE_NEED_SDCARD	Not enough internal space for download and the SD slot is empty.
0x8018826A	DUA_E_DOWNLOAD_OUT_OF_DISK_SPACE_BOTH_STORAGE	Not enough space for download on both internal storage and SD slot.
0x80188270	DUA_E_CONNECTION_BASE	Definition; the error code itself actually not used.
0x80188271	DUA_E_NO_CONNECTIONS	Connection Manager could not find a candidate connection.
0x80188272	DUA_E_CONNECTION_NOT_RELEASED	Connection Manager failed because an acquired connection was not released.
0x80188273	DUA_E_CONNECTION_ACQUIRE_FAILED	Connection Manager failed to acquire a connection.
0x80188288	DUA_E_DUAAPL_BASE	Definition; the error code itself actually not used.

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x80188289	DUA_E_DUAAPISHELLNOTREADY	WNF_SHL_START_READY is not ready yet.
0x801882C0	CABAPI_ERR_BASE; BTS_ERR_BASE	Base for CAPAPI; BTS errors.
0x801882C1	E_CABAPI_NOT_CABINET	BTS_E_PER_APP_REQUEST_LIMIT_REACHED
0x801882C2	E_CABAPI_UNKNOWN_FILE	BTS_E_UNABLE_TO_ENQUEUE_REQUEST
0x801882CB	BTS_E_HTTP_PROVIDER_ERROR	BTS_E_HTTP_PROVIDER_ERROR
0x801882CC	BTS_E_HTTP_PROVIDER_ERROR_RANGE RELATED .	BTS_E_HTTP_PROVIDER_ERROR_RANGE RELATED .
0x801882D1	E_CABAPI_FCI_OPEN_SRC	BTS_E_HTTP_PROVIDER_ERROR_NETWORK
0x801882D2	E_CABAPI_FCI_READ_SRC	BTS_E_HTTP_PROVIDER_ERROR_SLAVE_TRANSFER
0x801882D3	E_CABAPI_FCI_ALLOC_FAIL	BTS_E_HTTP_PROVIDER_ERROR_INVALIDID_pMETER.
0x801882D4	E_CABAPI_FCI_TEMP_FILE	BTS_W_TRANSFER_WAITING_BATTERY_SAVER_MODE
0x801882D5	E_CABAPI_FCI_BAD_COMPR_TYPE	BTS_W_SHUTTING_DOWN
0x801882D6	E_CABAPI_FCI CAB FILE	BTS_E_API_FAILED_DUE_TO_SHUTDOWN
0x801882D7	E_CABAPI_FCI_USER_ABORT	FCI: aborted
0x801882D8	E_CABAPI_FCI_MCI_FAIL	FCI: Failure compressing data
0x801882D9	E_CABAPI_FCI CAB FORMAT LIMIT	FCI: Data-size or file-count exceeded CAB format limits.
0x801882DA	E_CABAPI_FCI_INVALID_FILE_PATH	FCI: File path is empty or invalid

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x801882DB	E_CABAPI_FCI_DUPLICATE_FILE	FCI: Either a file is being added twice to the CAB; or two source files have the same target path
0x801882DF	E_CABAPI_FCI_UNKNOWN	FCI: An unknown error.
0x801882E0	CABAPI_ERR_FDI_BASE	Definition; the error code itself actually not used.
0x801882E1	E_CABAPI_FDI_CABINET_NOT_FOUND	FDI: the cabinet file was not found.
0x801882E2	E_CABAPI_FDI_NOT_A_CABINET	FDI: The cabinet file does not have the correct format.
0x801882E3	E_CABAPI_FDI_UNKNOWN_CABINET_VERSION	FDI: The cabinet file has an unknown version number.
0x801882E4	E_CABAPI_FDI_CORRUPT_CABINET	FDI: The cabinet file is corrupt.
0x801882E5	E_CABAPI_FDI_ALLOC_FAIL	FDI: Insufficient memory.
0x801882E6	E_CABAPI_FDI_BAD_COMPR_TYPE	FDI: Unknown compression type used in the cabinet folder.
0x801882E7	E_CABAPI_FDI_MDI_FAIL	FDI: Failure decompressing data from the cabinet file.
0x801882E8	E_CABAPI_FDI_TARGET_FILE	Failure writing to the target file. This could be that the device is out of space.
0x801882E9	E_CABAPI_FDI_RESERVE_MISMATCH	FDI: The cabinets within a set do not have the same RESERVE sizes.
0x801882EA	E_CABAPI_FDI_WRONG_CABINET	FDI: The cabinet returned by fdintNEXT_CABINET is incorrect.
0x801882EB	E_CABAPI_FDI_USER_ABORT	FDI: FDI aborted.
0x801882EF	E_CABAPI_FDI_UNKNOWN	FDI: An unknown error.

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x80188300	IMGUPD_ERROR_BASE	Definition; the error code itself actually not used.
0x80188301	E_BASE_PACKAGE_NOT_INSTALLED	The base package DSM file was not found in the DSM files store.
0x80188302	E_PACKAGE_ALREADY_INSTALLED	Package cannot be installed because it is already present on the image
0x80188303	E_NO_UPDATE_PATH_TO_TARGET	An update path to highest target version could not be found.
0x80188304	E_MULTIPLE_UPDATE_PATHS_FOUND	Multiple update paths to highest target version exist.
0x80188305	E_DUPLICATE_UPDATE_PACKAGE	Duplicate packages found in update set.
0x80188306	E_FILE_COLLISION	More than one package targeted for the same partition contained the same file
0x80188307	E_INVALID_PACKAGE	Package DSM; or contents are invalid
0x80188308	E_INSUFFICIENT_SPACE_FOR_UPDATE	Not enough space available in one or more partitions to complete update
0x80188309	E_REGISTRY_COLLISION	Two or more packages have registry settings that conflict
0x8018830A	E_INVALID_REGISTRY_FILE	A registry file included in the package is invalid
0x8018830B	E_DEPARED	Deprecated
0x8018830C	E_TARGET_NOT_APPLICABLE	This package has the target field set but this device is not part of any groups specified in the target.

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x8018830D	E_PACKAGE_HIGHER_VERSION_INSTALLED	This package cannot be installed because a newer version is already installed on the device.
0x8018830E	E_REMOVAL_PACKAGE_CANNOT_TARGET_BINARY_PARTITION	This package is a removal package targeted to a binary partition. Binary partition packages cannot be deleted.
0x8018830F	E_DIFF_SOURCEVERSION_DOES_NOT_MATCH	This package is a diff that targets a source version that is not the one on the device.
0x80188310	E_STAGED_FILE_NOT_FOUND	A file that was expected to be staged is missing from the staging area.
0x80188311	E_PACKAGE_TARGETS_WRONG_CPU	The device's CPU type does not match the package's targeted CPU.
0x80190190	HTTP_E_STATUS_BAD_REQUEST	BG_E_HTTP_ERROR_400
0x80190191	HTTP_E_STATUS_DENIED	BG_E_HTTP_ERROR_401
0x80190192	HTTP_E_STATUS_PAYMENT_REQ	BG_E_HTTP_ERROR_402
0x80190193	HTTP_E_STATUS_FORBIDDEN	BG_E_HTTP_ERROR_403
0x80190194	HTTP_E_STATUS_NOT_FOUND	BG_E_HTTP_ERROR_404
0x80190195	HTTP_E_STATUS_BAD_METHOD	BG_E_HTTP_ERROR_405
0x80190196	HTTP_E_STATUS_NONE_ACCEPTABLE	BG_E_HTTP_ERROR_406
0x80190197	HTTP_E_STATUS_PROXY_AUTH_REQ	BG_E_HTTP_ERROR_407
0x80190198	HTTP_E_STATUS_REQUEST_TIMEOUT	BG_E_HTTP_ERROR_408
0x80190199	HTTP_E_STATUS_CONFLICT	BG_E_HTTP_ERROR_409

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0x801901F4	HTTP_E_STATUS_SERVER_ERROR	BG_E_HTTP_ERROR_500
0x801901F5	HTTP_E_STATUS_NOT_SUPPORTED	BG_E_HTTP_ERROR_501
0x801901F6	HTTP_E_STATUS_BAD_GATEWAY	BG_E_HTTP_ERROR_502
0x801901F7	HTTP_E_STATUS_SERVICE_UNAVAIL	BG_E_HTTP_ERROR_503
0x801901F8	HTTP_E_STATUS_GATEWAY_TIMEOUT	BG_E_HTTP_ERROR_504
0x801901F9	HTTP_E_STATUS_VERSION_NOT_SUP	BG_E_HTTP_ERROR_505
0xC00CE01D	XML_E_INVALID_DECIMAL	
0xC00CE01E	XML_E_INVALID_HEXIDECIMAL	
0xC00CE01F	XML_E_INVALID_UNICODE	
0xC00CE06E	XML_E_INVALIDENCODING	
0xC00CEE00	MX_E_MX	DUA_E_MALFORMEDXML
0xC00CEE01	MX_E_INPUTEND	
0xC00CEE02	MX_E_ENCODING	
0xC00CEE03	MX_E_ENCODINGSWITCH	
0xC00CEE04	MX_E_ENCODINGSIGNATURE	
0xC00CEE20	WC_E_WC	
0xC00CEE21	WC_E_WHITESPACE	
0xC00CEE22	WC_E_SEMICOLON	

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0xC00CEE23	WC_E_GREATERTHAN	
0xC00CEE24	WC_E_QUOTE	
0xC00CEE25	WC_E_EQUAL	
0xC00CEE26	WC_E_LESSTHAN	
0xC00CEE27	WC_E_HEXDIGIT	
0xC00CEE28	WC_E_DIGIT	
0xC00CEE29	WC_E_LEFTBRACKET	
0xC00CEE2A	WC_E_LEFTPAREN	
0xC00CEE2B	WC_E_XMLCHARACTER	
0xC00CEE2C	WC_E_NAMECHARACTER	
0xC00CEE2D	WC_E_SYNTAX	
0xC00CEE2E	WC_E_CDSECT	
0xC00CEE2F	WC_E_COMMENT	
0xC00CEE30	WC_E_CONDSECT	
0xC00CEE31	WC_E_DECLATTLIST	
0xC00CEE32	WC_EDECLDOCTYPE	
0xC00CEE33	WC_E_DECLEMENT	
0xC00CEE34	WC_E_DECLEMENT	
0xC00CEE35	WC_E_DECLNOTATION	

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0xC00CEE36	WC_E_NDATA	
0xC00CEE37	WC_E_PUBLIC	
0xC00CEE38	WC_E_SYSTEM	
0xC00CEE39	WC_E_NAME	
0xC00CEE3A	WC_E_ROOTELEMENT	
0xC00CEE3B	WC_E_ELEMENTMATCH	
0xC00CEE3C	WC_E_UNIQUEATTRIBUTE	
0xC00CEE3D	WC_E_TEXTXMLDECL	
0xC00CEE3E	WC_E.LEADINGXML	
0xC00CEE3F	WC_E_TEXTDECL	
0xC00CEE40	WC_E_XMLDECL	
0xC00CEE41	WC_E_ENCNAME	
0xC00CEE42	WC_E_PUBLICID	
0xC00CEE43	WC_E_PESINTERNALSUBSET	
0xC00CEE44	WC_E_PESBETWEENDECLS	
0xC00CEE45	WC_E_NORECURSION	
0xC00CEE46	WC_E_ENTITYCONTENT	
0xC00CEE47	WC_E_UNDECLAREDENTITY	
0xC00CEE48	WC_E_PARSEDENTITY	

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0xC00CEE49	WC_E_NOEXTERNALENTITYREF	
0xC00CEE4A	WC_E_PI	
0xC00CEE4B	WC_E_SYSTEMID	
0xC00CEE4C	WC_E_QUESTIONMARK	
0xC00CEE4D	WC_E_CDSECTEND	
0xC00CEE4E	WC_EMOREDATA	
0xC00CEE4F	WC_E_DTDPROHIBITED	
0xC00CEE50	WC_E_INVALIDXMLSPACE	
0xC00CEE60	NC_E_NC	
0xC00CEE61	NC_E_QNAMECHARACTER	
0xC00CEE62	NC_E_QNAMECOLON	
0xC00CEE63	NC_E_NAMECOLON	
0xC00CEE64	NC_E_DECLAREDPREFIX	
0xC00CEE65	NC_E_UNDECLAREDPREFIX	
0xC00CEE66	NC_E_EMPTYURI	
0xC00CEE67	NC_E_XMLPREFIXRESERVED	
0xC00CEE68	NC_E_XMLNSPREFIXRESERVED	
0xC00CEE69	NC_E_XMLURIRESERVED	
0xC00CEE6A	NC_E_XMLNSURIRESERVED	

ERROR CODE	ERROR NAME	ADDITIONAL NOTES
0xC00CEE80	SC_E_SC	
0xC00CEE81	SC_E_MAXELEMENTDEPTH	
0xC00CEE82	SC_E_MAXENTITYEXPANSION	
0xC00CEF00	WR_E_WR	
0xC00CEF01	WR_E_NONWHITE SPACE	
0xC00CEF02	WR_E_NS PREFIX DECLARED	
0xC00CEF03	WR_E_NS PREFIX WITH EMPTY NSURI	
0xC00CEF04	WR_E_DUPLICATEATTRIBUTE	
0xC00CEF05	WR_E_XMLNS PREFIX DECLARATION	
0xC00CEF06	WR_E_XMLPREFIXDECLARATION	
0xC00CEF07	WR_E_XMLURIDECLARATION	
0xC00CEF08	WR_E_XMLNSURIDICTION	
0xC00CEF09	WR_E_NAMESPACEUNDECLARED	
0xC00CEF0A	WR_E_INVALIDXMLSPACE	
0xC00CEF0B	WR_E_INVALIDACTION	
0xC00CEF0C	WR_E_INVALIDSURROGATEPAIR	

# Update packages on a device and get package update logs

1/25/2019 • 2 minutes to read

The Windows Driver Kit (WDK) includes a tool for updating packages on a device (IUTool.exe) and a tool for getting package update logs from a device (GetDULogs.exe). These tools are available under %WPDKCONTENTROOT%\Tools\bin\i386.

## Using IUTool.exe to update packages on a device

IUTool.exe is a command-line tool that can be used to update an existing package on a device or to add a new package to a device. IUTool.exe must be used in a command-line window that is opened as an administrator. The command-line syntax for IUTool.exe is the following.

```
IUTool -p <path to packages>
```

The following table describes the command-line parameters for IUTool.exe.

PARAMETER	DESCRIPTION
<b>-p</b> <i>path to packages</i>	<p>Specifies one or more packages to update on the device or to add to the device. The <i>path to packages</i> parameter can have the following formats:</p> <ul style="list-style-type: none"><li>To update or add a single package, specify the full path to the package on the development computer.</li><li>To update or add multiple packages, specify a semicolon-delimited list of packages on the development computer. For example:</li></ul> <pre>IUTool -p C:\ContosoPackages\Contoso.Device.SampleDriver.spkg;C:\ContosoPackages\Contoso.Device.SampleApplication.spkg</pre> <ul style="list-style-type: none"><li>To update or add an entire directory of packages, specify the path to the directory. For example:</li></ul> <pre>IUTool -p C:\ContosoPackages</pre>

### Package versioning

If the specified package already exists on the device, the new version of the package must have a higher version than the package currently on the device or the update will fail. To specify the version for a package, use the **/version** command-line parameter for PkgGen.exe when generating the package. For more information, see [Command-line arguments for package generator](#).

## Using GetDULogs.exe to get package update logs from a device

GetDULogs.exe is a command-line tool that can be used to get package update logs from a device.

GetDULogs.exe must be used in a command-line window that is opened as an administrator. The command-line syntax for GetDULogs.exe is the following.

```
GetDULogs -o <output file path>
```

The following table describes the command-line parameters for GetDULogs.exe.

PARAMETER	DESCRIPTION
<b>-o</b> <i>output file path</i>	The full path of the file on the development computer to write the log information to.

# Update packages in an .FFU image file

1/25/2019 • 2 minutes to read

You can use ImageApp.exe to add a new or updated package to an existing .FFU image file for production, health and test images.

ImageApp has the following important limitations:

- ImageApp should only be used for adding packages to production, test and health images. Do not use ImageApp to modify retail images, as it may negatively impact update reliability and the security of the device. ImageApp cannot be used to change the partition layout of an existing image. If a different partition layout is needed, the image will need to be rebuilt. For more information, see [Building a device image using ImgGen.cmd](#).
- ImageApp cannot be used to remove packages from an image.
- To prepare a device platform to use compressed partitions with CompactOS, you'll need a PC with the Windows 10 version of DISM. If your technician PC is running a previous version of Windows, you can get this by installing the Windows Assessment and Deployment Kit (ADK) for Windows 10, or by copying and installing the DISM driver. This process is the same as the one used to install DISM on Windows PE. To learn more, see [Install Windows 10 using a previous version of Windows PE: To add DISM into your Windows PE image](#).

When updating an existing package, be sure to increment the version number. For more information, see [Update requirements](#). When adding a package that does not already exist in the image, any version number can be used.

Specify the packages to be added in an input XML file similar to the one shown here.

```
<?xml version="1.0" encoding="utf-8"?>
<UpdateOSInput xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate">
    <Description>Add Debugger On package</Description>
    <PackageFiles>
        <PackageFile>C:\Program Files\Windows Kits\10\Packages\Microsoft.BCD.DebuggerOn.spkg</PackageFile>
    </PackageFiles>
</UpdateOSInput>
```

For example, if the `updateInput.xml` file is in the C:\temp folder, use this command to add the specified packages to the existing `flash.ffu` image file.

```
ImageApp flash.ffu /UpdateInputXML:C:\temp\updateInput.xml
```

## Command-line syntax for ImageApp.exe

```
ImageApp.exe <imageFile.ffu> </UpdateInputXML:>updateInputfile.xml>
```

The following table describes the command-line parameters for ImageApp.exe.

PARAMETER	DESCRIPTION
<code>imageFile.ffu</code>	The name of the FFU file to be updated.

PARAMETER	DESCRIPTION
/UpdateInputXML: <i>updateInputfile.xml</i>	The name of the XML file that identifies the package to be added to the image. If this file is not in the current directory, you must include the path to the file.

## Troubleshooting

**"STATUS\_FILE\_IS\_A\_DIRECTORY"**: This error message appears when building an image with CompactOS from a PC that doesn't have the Windows 10 version of DISM. You can get this by installing the Windows ADK for Windows 10, or by just installing the DISM driver from another PC with the Windows ADK for Windows 10 installed. To learn more, see [Install Windows 10 using a previous version of Windows PE](#).

## Related topics

[Building an image using ImgGen.cmd](#)

# IoT Core manufacturing

1/18/2019 • 2 minutes to read

Windows 10 IoT Core (IoT Core) is a version of Windows 10 that is optimized for smaller devices with or without a display. IoT Core uses the rich, extensible Universal Windows Platform (UWP) API for building great solutions.

OEMs can manufacture and deploy IoT Core using existing or custom-built hardware. To see existing recommended hardware, see [device options](#) and the [Hardware Compatibility List](#).

When developing your own board, see the [Minimum hardware requirements for IoT Core](#).

## In this section

TOPIC	DESCRIPTION
<a href="#">What's new in IoT Manufacturing</a>	Find out what we've been working on.
<a href="#">IoT Core manufacturing guides</a>	This guide walks you through creating IoT Core images that can be flashed to retail devices and maintained after they have been delivered to your customers.
<a href="#">IoT Core feature list</a>	Here's the features you can add to IoT Core images.
<a href="#">Windows ADK IoT Core Add-ons contents</a>	The <a href="#">IoT Core ADK Add-Ons</a> include tools to help you customize and create new images for your devices with the apps, board support packages (BSPs), drivers, and Windows features that you choose, and a sample structure you can use to quickly create new images.
<a href="#">IoT Core Add-ons Powershell Commands</a>	These tools are part of the <a href="#">IoT Core ADK Add-Ons</a> , in the \Tools folder. To learn more about these tools, see <a href="#">Windows ADK IoT Core Add-ons</a> .
<a href="#">IoT Core Image Wizard</a>	The <a href="#">IoT Core Image Wizard</a> is a GUI tool to help you create a new workspace and image for your device.

## Related topics

[Learn about Windows 10 IoT Core](#)

[IoT Core Developer Resources](#)

# OEM license requirements

1/24/2019 • 2 minutes to read

Have questions about commercialization? Check out our Commercialization FAQ [here](#).

## Windows 10 IoT Core Services

[Windows 10 IoT Core Services](#) is a new cloud services subscription that provides the essential services needed to commercialize a device on Windows 10 IoT Core. For optimal security, update control, and device health, we highly recommend subscribing to Windows 10 IoT Core Services. Through this subscription, OEMs have access to 10 years of support on Windows 10 IoT Core Long Term Servicing Channel (LTSC) releases along with services to manage device updates and assess device health.

## Semi-Annual Channel (SAC)

If you are handling the maintenance and upkeep of commercialized devices on your own, SAC may be another option for you to consider when commercializing a device. While SAC comes at no cost, you will not have access to the features that Windows 10 IoT Core Services provides. You will also need to come back every six months to renew SAC.

To start your commercialization journey with SAC, you will need to sign a license, which you will need to request the license agreement [here](#). A member of the team will get back to you with the agreement form within 48 to 72 hours.

You can view a copy of this agreement before requesting to sign it [here](#).

## SMBIOS Support

The system firmware must implement support for SMBIOS that complies with System Management BIOS Reference Specification, Version 2.4 or later. The SMBIOS implementation must follow all conventions and include all required structures and fields as indicated in the SMBIOS Specification, Section 3.2, and follow all conformance requirements as indicated in Section 4. Bit 2 in the BIOS Characteristics Extension Byte 2 field must be set (Section 3.3.1.2.2 of the specification). The length of the Type 1 (System Information) table must be at least 1Bh bytes (includes SKU Number and Family fields from Version 2.4 of the specification).

The following are the minimum required fields in SMBIOS for IoTCore

- (Table 1, offset 04h) System Manufacturer
- (Table 1, offset 05h) System Product Name
- (Table 1, offset 19h) System SKU
- (Table 1, offset 1Ah) System Family
- (Table 2, offset 05h) Baseboard Product

These fields gain prominence as fields which will be used for identifying unique system configurations for telemetry and servicing. The *Manufacturer*, *Product Name*, *SKU Number* and *Family* fields must not be longer than 64 characters in length. Avoid leading or trailing spaces or other invisible characters.

**TIP**

Design Notes: SKU Number has been moved to a required field in order to improve telemetry reporting. We encourage the OEM to be careful to fill in *Manufacturer* consistently and to fill in *SKU Number* with a value that can identify what the OEM considers a unique system configuration for telemetry and servicing.

For more information, see Section **System.Fundamentals.SMBIOS** in *WHCP-Systems-Specification.pdf* available at [Windows Hardware Compatibility Program Specifications and Policies](#).

**IMPORTANT**

If you are re-using the BIOS/Firmware/UEFI, make sure to update the entries.

## Compliance with minimum hardware requirements

Review the IoT Core sections of [minimum hardware requirements](#).

# Create and install a package

4/5/2019 • 2 minutes to read

Packages are the building blocks of Windows 10 IoT Core. From device drivers to system files, every component must be packaged to install on a device. Packages are the smallest servicable units on the device.

## Step 1: Get set up

### Install the tools

1. [Windows Assessment and Deployment Kit\(Windows ADK\)](#)
2. [IoT Core ADK Add-Ons](#)

### Set up your environment

- Launch `IotCorePShell.cmd` ( this one launches in the elevated prompt )
- Create a new workspace using `new-ws C:\MyWorkspace <oemname> <arch>`

To create your own image (FFU), follow the steps outlined in the "["Create a basic image" lab in the IoT Manufacturing guide](#).

## Step 2: Create a new package

1. Create a **package definition xml file** (.wm.xml file), and specify the files and reg keys you want to add.  
Learn more at [Windows Universal OEM Package Schema](#).
2. Build the package: `buildpkg filename.wm.xml`. The .cab file will be created in the build directory  
`<workspace>\Build\<arch>\pkgs`.

### Create a package with files and reg keys

Below is an example for specifying files and reg keys.

```

<?xml version="1.0" encoding="utf-8"?>
<identity xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    name="SUBNAME" namespace="COMPNAME" owner="Contoso" legacyName="Contoso.COMPNAME.SUBNAME"
    xmlns="urn:Microsoft.CompPlat/ManifestSchema.v1.00">
    <onecorePackageInfo
        targetPartition="MainOS"
        releaseType="Production"
        ownerType="OEM" />
    <regKeys>
        <regKey
            keyName="$(hkLM.software)\Contoso\Test">
            <regValue name="StringValue" type="REG_SZ" value="Test string" />
            <regValue name="DWordValue" type="REG_DWORD" value="0x12AB34CD" />
            <regValue name="BinaryValue" type="REG_BINARY" value="12ABCDEF" />
        </regKey>
        <regKey
            keyName="$(hkLM.software)\Contoso\EmptyKey" />
    </regKeys>
    <files>
        <file
            destinationDir="$(runtime.system32)"
            source="filename.txt" />
        <file
            destinationDir="$(runtime.bootDrive)\OEMInstall"
            source="filename2.txt"
            name="filename2.txt" />
    </files>
</identity>

```

## Create an Appx package

Use [Add-IoTAppxPackage](#) to generate the .wm.xml file for a given appx file. This tool expects the appx dependencies in the sub directory named "dependencies" in the folder containing the appx file.

```

Add-IoTAppxPackage HelloWorld.appx fga Appx.HelloWorld
(or) newappxpkg HelloWorld.appx fga Appx.HelloWorld
New-IoTCabPackage Appx.HelloWorld
(or) buildpkg Appx.HelloWorld

```

`fga` sets the appx as the foreground startup app, `bgt` sets the appx as the background task and `none` skips startup configuration. For older commandline tool, see [newappxpkg.cmd](#)

See [Appx.IoTCoreDefaultApp](#) as an example.

When you have to install multiple applications signed with same certificate, you can add the certificate along with one app and for the remaining apps, you can skip adding the certificate using the `skipcert` flag.

```
newappxpkg AnotherApp.appx none Appx.AnotherApp skipcert
```

See also

- [Lab 1b: Add an app to your image](#)

## Create a driver package

The driver package contains the references (InfSource) to the Inf file for the driver. You can author the driver .wm.xml file manually or use [Add-IoTDriverPackage](#) that generates package xml based on the input inf file.

```
Add-IoTDriverPackage C:\Mydriver\GPIO.inf MyDriver.GPIO  
(or) newdrvpkg C:\Mydriver\GPIO.inf MyDriver.GPIO  
New-IoTCabPackage MyDriver.GPIO  
(or) buildpkg MyDriver.GPIO
```

For the older commandline tool, use [inf2cab.cmd](#) creates the package xml file and also builds the cab file directly by invoking [buildpkg.cmd](#) internally.

**NOTE**

Windows IoT Core supports Universal Inf only.

See also

- [Sample Driver Package](#)

## Step 3: Install on device

- Connect to the device ([using SSH](#) or [using Powershell](#))
- Copy the .cab file to the device to a directory say C:\Oem\Install
- Initiate staging of the package using [applyupdate -stage C:\OemInstall\<filename>.cab](#). Note that this step is be repeated for each package, when you have multiple packages to install.
- Commit the packages using [applyupdate -commit](#).

**NOTE**

You can also install the cab using **Windows Update > CAB Install** option in Windows Device Portal. The device will reboot into the update OS (showing gears) to install the packages and will reboot again to main OS. This process can take a few minutes.

# Windows 10 IoT Core Services

1/24/2019 • 10 minutes to read

Windows 10 IoT Core Services is a new cloud services subscription that provides the essential services needed to commercialize a device on Windows 10 IoT Core. Through this subscription, OEMs have access to 10 years of support on Windows 10 IoT Core Long Term Servicing Channel (LTSC) releases along with services to publish device updates and assess device health.

With the 1809 release of Windows 10 IoT, we are providing 10 years of support for both Windows 10 IoT Core and Windows 10 IoT Enterprise. We've also entered a partnership with NXP to support Windows 10 IoT Core on their i.MX 6, 7, and 8M series processors. The Windows 10 IoT Core Services offering builds on these improvements, as we continue to evolve the Windows IoT platform and make investments to support commercialization of devices running Windows 10 IoT.

## What's included with Windows 10 IoT Core Services?

A subscription to Windows 10 IoT Core Services comes with three main benefits:

BENEFIT	DESCRIPTION
10 years of support	Access to Windows 10 IoT Core LTSC releases with security and reliability updates only (no new features).
Update control with Device Update Center	Create and publish device updates at scale using cloud-side controls.
Device Health Attestation (DHA)	Rights to commercialize a device with DHA to affirm device health remotely.

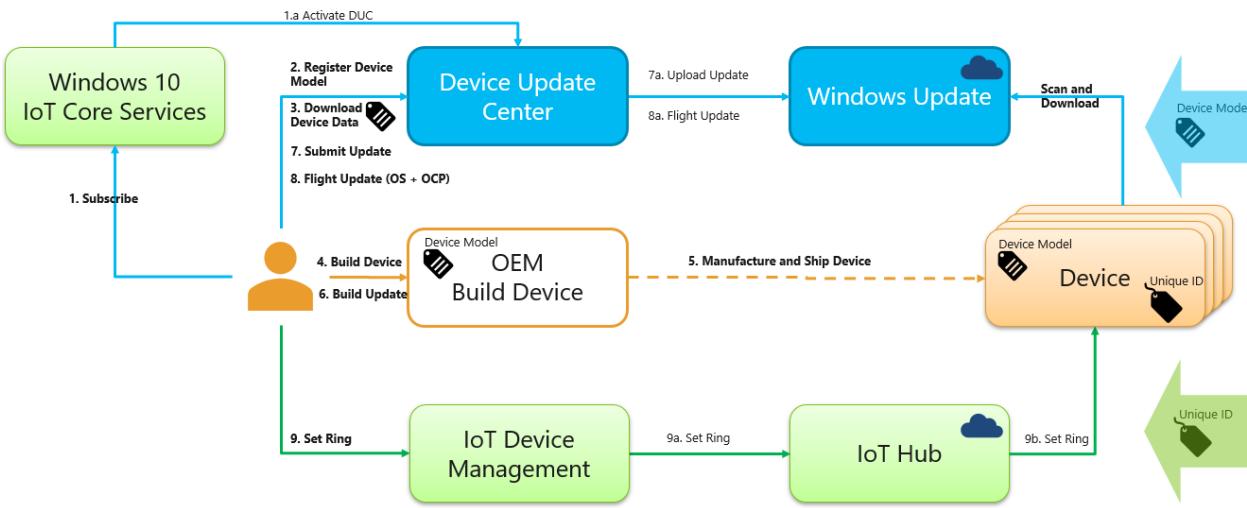
## 10 Years of Support

OEMs get 10 years of support on Windows 10 IoT Core via access to monthly updates from the [Windows Long-Term Servicing Channel \(LTSC\)](#) releases. This includes security and reliability updates for the operating system to keep device security up to date. Devices using LTSC releases won't receive feature updates, enabling OEMs to focus updates on stability by minimizing changes to the base operating system. Microsoft typically offers new LTSC releases every 2-3 years, with each release supported over a 10-year lifecycle.

## Update Control with Device Update Center

Update control with the newly announced [Device Update Center](#) (DUC) provides the ability to create, customize, and publish device updates. These updates are distributed by the same content distribution network (CDN) as Windows Update which is used daily by millions of Windows customers around the world. Updates can be applied to the operating system and device drivers as well as OEM-specific applications and files. Updates can be flighted to test devices prior to broader distribution.

Here's a diagram of the update flow in Device Update Center:



## Commercialize with Device Health Attestation

**Device Health Attestation (DHA)** enables enterprises and OEMs to raise the security bar of their organization's assets with hardware-attested security. Evaluating the trustworthiness of a device at boot time is essential for a trusted IoT system. However, a device cannot attest to its own trustworthiness - this must be done by an external entity such as the [DHA cloud service](#). This service evaluates device health and can be combined with a device management system, such as Azure IoT Device Management. Based on DHA report data, the device management system can take corrective actions such as re-imaging the device, denying network access, or creating a service ticket.

### Is this for me?

If you work with IoT devices, you might be wondering if Windows 10 IoT Core Services is right for you or your organization. The answer depends on how you use and obtain Windows 10 IoT devices. The following information will help you decide whether this subscription service is right for you.

1. If you are creating a device and you control the update and maintenance of the full software image on the device, this service is for you. If you only install apps on the device and someone else updates the full software image, such as the operating system and drivers, this service isn't for you. If you're not sure, contact your supplier and ask if they maintain the operating system, drivers, and other parts of the system image on the device or whether you are expected to maintain the system image. This service is designed for the party that maintains the system image for the device which is typically the ODM or OEM of the device, not the end customer purchasing or using the device.
2. Your devices should be running Windows 10 IoT Core. If they are running Windows 10 IoT Enterprise, another version of Windows, or another operating system, this service isn't for you. Services such as [Azure IoT Device Management](#) and [Microsoft Intune](#) offer cross-platform support which may be useful ingredients in configuring alternate solutions for those operating systems.
3. If you want to build a device with the [Long Term Servicing Channel \(LTSC\)](#) releases of Windows 10 IoT Core, this service is for you. You must subscribe to Windows 10 IoT Core Services to commercialize a device with an LTSC release of Windows 10 IoT Core. If you only want to run Semi-Annual Channel releases of Windows 10 IoT Core, you aren't required to subscribe to this service.
4. If you want to use any one of the three services included in Windows 10 IoT Core Services on your devices, this service is for you. Even if you only need one of the three services included in the subscription, you need to purchase the subscription for your devices. The services are only sold together as a single product - they are not available for purchase separately.

## Getting started

Windows 10 IoT Core Services is available through Azure, Cloud Solutions Provider, and OEM channels. If you already meet the prerequisites below and are familiar with creating resources in Azure, you can skip straight to the [Windows 10 IoT Core Services](#) page on Azure.com and the [Create blade in the Azure Portal](#) to get started. For more details on these steps, keep reading below.

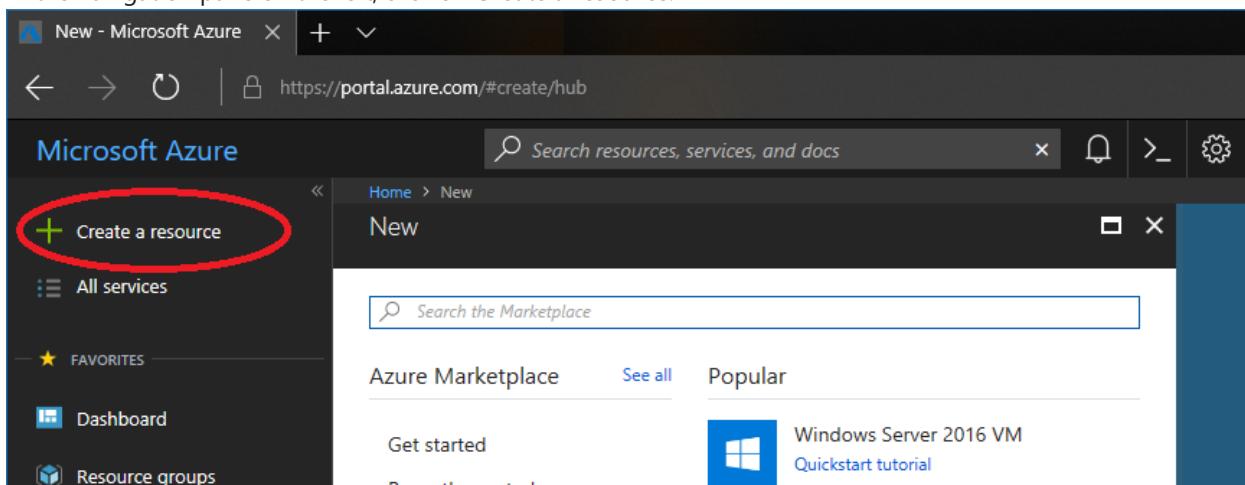
## Prerequisites

1. Create or use an existing [Azure Active Directory](#) domain for registering with Windows Hardware Dev Center. You will need to sign in with an account with administrator rights to your AAD tenant domain to sign legal agreements during the registration process.
2. You must have an [Extended Validation \(EV\) code signing certificate](#). Please check whether your company already has an EV code signing certificate. If your company already has an EV code signing certificate, have the certificate available during the registration process. If your company doesn't have a certificate, you will need to purchase a certificate from an [authorized partner](#) as part of the registration process.
3. Register for a [Windows Hardware Dev Center account](#) to get access to Device Update Center. Make sure you sign in as an administrator with the Azure Active Directory domain you will use for Device Update Center and have your Extended Validation (EV) code signing certificate available.

## Creating a new resource

After you've completed the prerequisites above, follow these steps when you are ready to create a resource for Windows 10 IoT Core Services with your Azure subscription.

1. Log into the [Azure Portal](#) with the AAD account you will use for purchasing the resource.
2. In the navigation pane on the left, click on Create a resource.



The screenshot shows the Microsoft Azure portal interface. On the left, there's a navigation pane with a red oval highlighting the 'Create a resource' button. The main area is titled 'New' and shows a search bar for 'Search resources, services, and docs'. Below it, there's a 'Search the Marketplace' bar. Underneath, there are sections for 'Azure Marketplace', 'See all', and 'Popular'. A 'Windows Server 2016 VM' card is visible, featuring a 'Quickstart tutorial' link. The URL in the browser bar is https://portal.azure.com/#create/hub.

3. In the Search box, search for "IoT" and select "Windows 10 IoT Core Services" from the list.
4. Review the information on the overview blade for Windows 10 IoT Core Services, including the Prerequisites information. When you're ready, click the Create button at the bottom of the overview blade.
5. On the Create blade, fill out the fields in the form. These fields can also be edited later in the resource properties.
  - **Name:** This is a friendly name for this instance of the Windows 10 IoT Core Services resource. This name is only used in the Azure Portal. Pick something that will help you distinguish this resource instance from others you might add to your Azure subscription. For example, consider a name that describes this group of devices such as "Industrial Extraction Monitor Device - Preproduction".
  - **Quantity:** Specify the number of devices to use with this resource. If you need to add more devices later, you can increase the quantity of devices using this resource or you can create a separate resource to track the additional devices. The minimum quantity is 10. If you have more than 500,000 devices, create separate resource instances with at most 500,000 devices per instance.
  - **Billing AAD domain name:** This corresponds to the Azure Active Directory domain name associated with the organization that is purchasing this resource. This field is automatically populated with the AAD domain of the user currently logged into the Azure Portal.
  - **Admin AAD domain name:** This corresponds to the Azure Active Directory domain name associated

with the organization that logs into Device Update Center to publish updates for the system image for the device. By default (or if left blank), this field is populated with the AAD domain of the user currently logged into the Azure Portal. However, it can be changed to a different AAD domain, if needed.

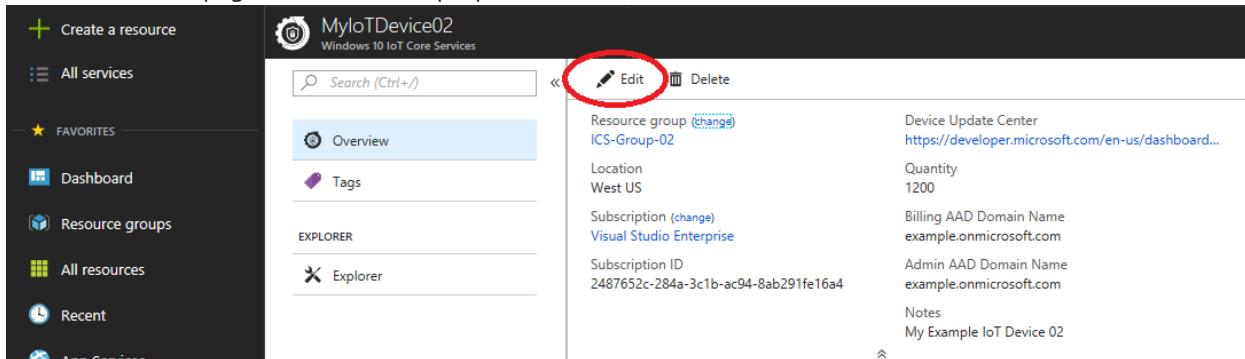
- **Notes:** This is a free-form text field to attach your own custom notes to this resource instance. This optional field is only used in the Azure Portal to help you organize your resources.
- **Subscription:** Specify the Azure subscription to use for purchasing this resource. For more details on which subscriptions are available in your account, see the Subscriptions section of the Azure Portal in the navigation pane on the left.
- **Resource group:** Specify the Resource Group to use for organizing this resource. All resources in Azure are organized into [Resource Groups](#). You can use an existing group or create a new resource group when creating this resource.

6. After filling out the fields above, click the Create button at the bottom of the form. This will start the deployment process to add a new instance of the Windows 10 IoT Core Services resource to your Azure subscription.
7. After the resource is successfully deployed, you can find it on the Dashboard or All Resources section of the Azure Portal. If the deployment fails, check the error message for details. Also verify that you've completed the Prerequisites above and specified the correct Admin AAD domain name for logging into Windows Hardware Dev Center.

## Updating a resource

After you've created a resource for Windows 10 IoT Core Services, you can update the resource by editing its properties in the Azure Portal.

1. Log into the [Azure Portal](#) with the AAD account you used when creating the resource.
2. Find the instance of the resource you want to change by selecting it in the Dashboard or All Resources section of the navigation pane in the Azure Portal.
3. On the Overview page of the resource properties, click the Edit button.



4. Change the fields as needed and then click Save.

## Deleting a resource

When you no longer need the resource for Windows 10 IoT Core Services, you can delete the resource from your Azure subscription in the Azure Portal.

1. Log into the [Azure Portal](#) with the AAD account you used when creating the resource.
2. Find the instance of the resource you want to delete by selecting it in the Dashboard or All Resources section of the navigation pane in the Azure Portal.
3. On the Overview page of the resource properties, click the Delete button.

The screenshot shows the Azure portal interface for managing a Windows 10 IoT Core Services resource named 'MyIoTDevice02'. On the left, there's a sidebar with navigation links like 'Create a resource', 'All services', 'FAVORITES', 'Dashboard', 'Resource groups', 'All resources', 'Recent', and 'App Services'. The main content area displays the resource details. At the top right, there are 'Edit' and 'Delete' buttons, with 'Delete' being circled in red. Below these are sections for 'Resource group (change)', 'Location', 'Subscription (change)', 'Subscription ID', 'Device Update Center', 'Quantity', 'Billing AAD Domain Name', 'Admin AAD Domain Name', and 'Notes'.

## Using Device Update Center

As a device manufacturer, you can maintain the security and reliability of your device by regularly updating the device image. This is accomplished by using Device Update Center to create and publish updates specific to your device model which will be offered by Windows Update when the device scans for applicable updates.

1. Before creating updates, start by creating a baseline image which is applied to the device during manufacturing as described in the [Build Your Image](#) section of the Windows IoT Core docs.
2. It's critical that you correctly [populate the SMBIOS fields](#) of your device and that your build environment is set up with the [same corresponding values](#).
3. Ensure that you've registered for a [Windows Hardware Dev Center account](#) to get access to Device Update Center. Make sure you sign in as an administrator with the Azure Active Directory domain you will use for Device Update Center and have your Extended Validation (EV) code signing certificate available.
4. Log into [Device Update Center](#) using the same AAD account you used when registering with Windows Hardware Dev Center. The AAD domain name on this account should also match the Admin AAD domain name specified when creating the resource in the Azure Portal.
5. Follow the instructions in the [Device Update Center User Guide](#) to create and publish updates for your device.

## Frequently Asked Questions (FAQs)

For a list of frequently-asked questions, see the [Pricing page](#) for Windows 10 IoT Core Services on Azure.com.

# IoT Core manufacturing guide

6/24/2019 • 6 minutes to read

## NOTE

For the previous version, see [IoT Core manufacturing guide for v5.x](#).

Thinking about mass-producing devices running Windows 10 IoT Core? Use the [Windows ADK IoT Core Add-ons](#) to create images that you can quickly flash onto new devices.

You can create **test images**, which include tools for quickly accessing and modifying devices. Test images are great for:

- Developers, hardware vendors, and manufacturers (OEMs) who are trying out new device designs.
- Hobbyists and organizations that are creating devices designed to run in non-networked or controlled network environments.

You can create **retail images**, which can be made more secure for public or corporate networks while still receiving updates.

You can add customizations, including apps, settings, hardware configurations, and board support packages (BSPs).

For OEM-style images, you'll wrap your customizations into package (.cab) files. Packages let OEMs, ODMs, developers, and Microsoft work together to help deliver security and feature updates to your devices without stomping on each other's work.

## Scenarios

- [Get the tools needed to customize Windows IoT Core](#)
- [Lab 1a: Create a basic image](#)
- [Lab 1b: Add an app to your image](#)
- [Lab 1c: Add file\(s\) and registry settings to an image](#)
- [Lab 1d: Add networking and other provisioning package settings to an image](#)
- [Lab 1e: Add a driver to an image](#)
- [Lab 1f: Build a retail image](#)
- [Lab 2: Creating your own board support package](#)
- [Lab 3: Update your apps](#)

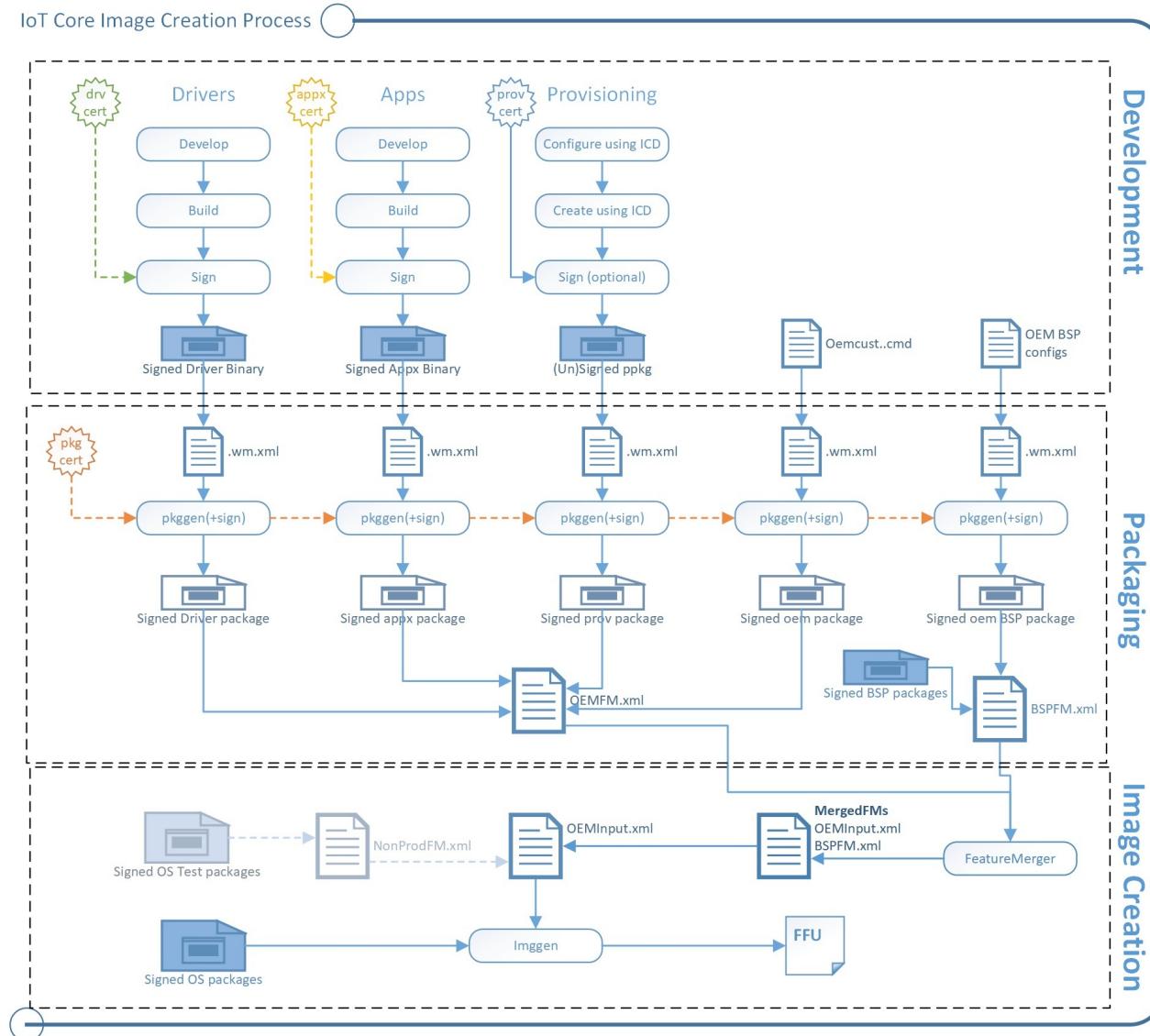
## Concepts

In order to clearly understand the process to create and deploy Windows IoT Core images, a few concepts and terminology must be defined first.

The process to create a Windows IoT Core image involves a number of steps, listed here:

1. Test any customization to include in the image to verify they work properly. These include apps, settings, drivers or Board support Packages (BSPs).
2. Install test certificates on your PC, and package your customizations into .CAB files.
3. Create a test Windows 10 IoT Core image that includes your customizations, along with the IoT Core package

- and any updates from your hardware manufacturer.
4. Flash the image to a device and test that it works. You can use the test tools built into the test image to troubleshoot any issues that may arise.
  5. Once you've verified that everything works properly, obtain a valid retail certificate and sign your customizations with your retail certificate. You then should repackage the customization into new .CAB files.
  6. Create a retail image with your signed .CAB files, and flash the image onto your devices.



## Terminology

### Packages

Packages (.cab files) are the logical building blocks of IoT Core. They contain all the files, libraries, registry settings, executables, and data on the device. From device drivers to system files, every component must be contained in a package. This modular architecture allows for precise control of updates: a package is the smallest serviceable unit on the device.

Each package contains:

- The contents of the package, such as a signed driver binary or a signed appx binary.
- A package definition (.wm.xml) file specifies the contents of the package and where they should be placed in the final image. See %SRC\_DIR%\Packages\ directory from the [Windows ADK IoT Core Add-On](#) kit for various samples of package files. As an example, take a look at [Appx.IoTCoreDefaultApp.wm.xml](#).
- A signature. A package can be signed with a test or retail certificate.

The `pkgen` tool combines these items into signed packages. Our samples include scripts: `createpkg`, and `createprovpkg`, which call pkgen to create packages for our drivers, apps, and settings.

The process is similar to that used by Windows 10 Mobile. To learn more about creating packages, see [Creating mobile packages](#).

## Feature manifests (FMs)

After you've put everything into packages, you'll use Feature Manifest (FM files) to list which of your packages belong in the final image.

You can use as many FMs into an image as you want. In this guide, we refer to the following FMs:

- **OEMFM.xml** includes features an OEM might add to a device, such as the app and a provisioning package.
- **BSPFM.xml** includes features that a hardware manufacturer might use to define a board. For example, `OEM_RPi2FM.xml` includes all of the features used for the Raspberry Pi 2.

The process is similar to that used by Windows 10 Mobile. To learn more, see [Feature manifest file contents](#).

You'll list which of the features to add by using these tags:

- `<BasePackages>`: Packages that you always included in your images, for example, your base app.
- `<Features>\<OEM>`: Other individual packages that might be specific to a particular product design.

The Feature Merger tool generates the required feature identifier packages that are required for servicing the device. Run this tool whenever any changes are made to the FM files. After you change the OEM FM or OEM COMMON FM files, run `buildfm oem`. After you change the BSPFM file, run `buildfm bsp <bspname>`. These commands are run from the IoT Core shell.

## Board Support Packages (BSPs)

Board Support Packages contain a set of software, drivers, and boot configurations for a particular board, typically supplied by a board manufacturer. The board manufacturer may periodically provide updates for the board, which your devices can receive and apply. You can also create your own BSP if the board manufacturer does not provide one, and you have the corresponding set of software and driver files. Supported BSPs are listed [here](#).

## Full Flash Update Image Files

Full Flash Update (FFU) files are image files that can be deployed (aka. "flashed") to a specific hardware device. When you flash a FFU file onto a device, all required software is installed on that device at the same time. An FFU image file bundles boot loaders, the Windows Operating System, drivers, peripheral images and any other required files into a single package.

## Foreground and Background Apps

There are two types of applications that can run on Windows IoT Core.

- Foreground apps - these apps have a UI. Only one app can be running on an IoT device as the foreground apps. If multiple foreground app is included in the image, only one should be set as auto start on boot.
- Background apps - these apps do not have a UI. Multiple apps can be running on an IoT device as background apps. You can configure any number of background apps to auto start.

For more information, please see [Foreground Apps](#) or [Background Apps](#).

## Creating the image: ImgGen and the image configuration file (OEMInput.xml)

To create the final image, you'll use the `imggen` tool with an image configuration file, **OEMInput.xml file**.

These are the same tools used to create Windows 10 Mobile images. To learn more, see [OEMInput file contents](#).

The image configuration file lists:

- The feature manifests (FMs) and the packages that you want to install from each one.
- An **Soc** chip identifier, which is used to help set up the device partitions. The supported values for **soc** are defined in the corresponding bspfm.xml, under <devicelayoutpackages>.
- A **Device** identifier, which is used to select the device layout. The supported values for **device** are defined in the corresponding bspfm.xml, under <oemdeviceplatformpackages>.
- The ReleaseType (either **Production** or **Test**).

**Retail builds:** We recommend creating retail images early on in your development process to verify that everything will work when you are ready to ship.

These builds contain all of the security features enabled.

To use this build type, all of your code must be signed using retail (not test) code signing certificates.

For a sample, see %SRC\_DIR%\Products\SampleA\RetaiLOEMInput.xml.

**Test builds:** Use these to try out new versions of your apps and drivers created by you and your hardware manufacturer partners.

These builds have some security features disabled, which allows you to use either test-signed or production-signed packages.

These builds also include developer tools such as debug transport, SSH, and PowerShell, that you can use to help troubleshoot issues.

For a sample, see %SRC\_DIR%\Products\SampleA\TestOEMInput.xml.

	RETAIL BUILDS	TEST BUILDS
Image release type	ReleaseType: <b>Production</b>	ReleaseType: <b>Test</b>
Package release type	Only Production Type packages are supported	Both Production Type or Test Type are supported
Test-signed packages	Not supported	Supported IOT_ENABLE_TESTSIGNING feature must be included.
Code integrity check	Supported. By default, this is enabled.	Supported. By default, no policy is enforced

## OK, let's try it!

Start here: [Get the tools needed to customize Windows IoT Core.](#)

# Get the tools needed to customize Windows IoT Core

6/24/2019 • 4 minutes to read

Here's the hardware you'll need to create OEM images using the Windows 10 IoT Core (IoT Core) ADK Add-Ons:

## PCs and devices

- **Technician PC:** This is the PC you will be building your images on. A dedicated PC is recommended for this. This PC should have at least 15 GB of free space for installing the software and for modifying IoT Core images.

We recommend either Windows 10 or Windows 8.1 with the latest updates. The minimum requirement is Windows 7 SP1, though this may require additional tools or workarounds for tasks such as mounting .ISO images.

- **IoT device:** A test device or board that represents all of the devices in a single model line.

For our labs, you'll need a Raspberry Pi 3. For a list of support devices, see [SoCs and Custom Boards](#).

- An **HDMI cable**, and a **monitor or TV** with a dedicated HDMI input. We'll use this to verify that the image is loaded and that our sample apps are running.

## Storage

- A **MicroSD card**. Note that this is dependent on the device you are using. Some devices require a microSD card while others have onboard storage. For these labs, we'll use a microSD card. If your technician PC doesn't include a microSD slot, you may also need an adapter. Learn more about existing [supported storage](#) options.

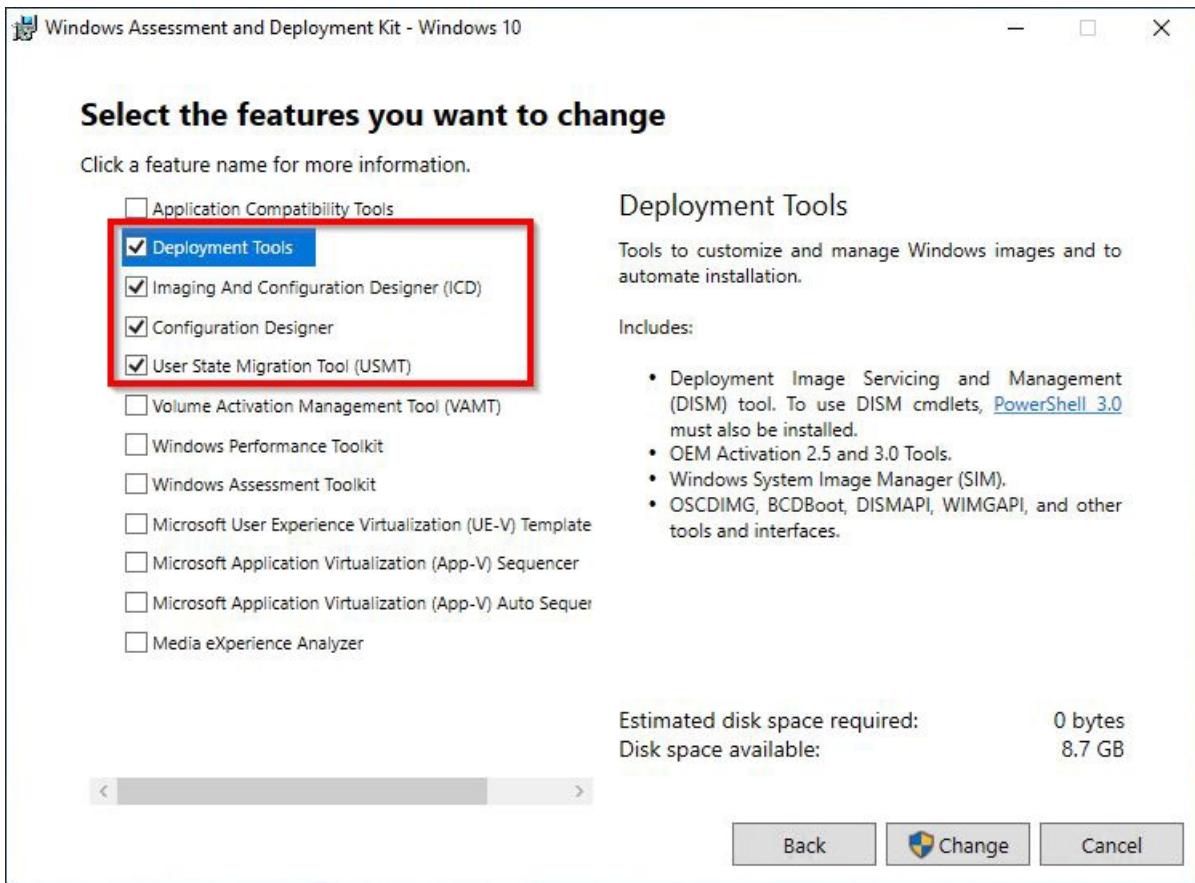
## Software

Here's the software you'll need to create OEM images using the Windows 10 IoT Core (IoT Core) ADK Add-Ons:

### Install the following tools on your technician PC

1. [Windows Assessment and Deployment Kit \(Windows ADK\)](#) - Including at least the **Deployment Tools** and **Imaging and Configuration Designer (ICD)** features, shown below. You'll use these tools to create images and provision packages. For Windows 10, version 1809, you will also need to install **Windows PE add-on for the ADK**.

**NOTE** - The version of ADK and the version of IoT Core Packages used **must** match.



2. **Windows PE add-on for the ADK** - If you need to build image with Recovery capabilities, you will also need to separately download and install the [Windows PE add-on for the ADK](#). In previous releases, this was part of the ADK.
3. **Windows Driver Kit (WDK) 10** - [optional](#), required only if you are compiling drivers from source code.
4. **Windows 10 IoT Core Packages** - The .iso package adds the IoT Core packages and feature manifests used to create IoT Core images. By default, these packages are installed to **C:\Program Files (x86)\Windows Kits\10\MSPackages\Retail**. Install one or more of the IoT Core packages, depending on the architecture you are building an image for (ARM, ARM64, x86, x64).



5. **IoT Core ADK Add-Ons** - Click **Clone or Download > Download ZIP**, and extract it to a folder on the technician PC (for example, **C:\IoT-ADK-AddonKit**). This kit includes the sample scripts and base structures you'll use to create your image. To learn about the contents, see [What's in the Windows ADK IoT Core Add-ons](#). **IoT-ADK-AddonKit version 6.x** is built with Powershell and the older command line scripts are in maintenance mode for supporting older releases.
6. **Windows 10 IoT Core Dashboard**
7. **The Raspberry Pi BSP**. Since this lab uses a Raspberry Pi, you'll need to download the Raspberry Pi BSP. If you're working with a device other than Raspberry Pi, visit the [Windows 10 IoT Core BSP](#) page to download other BSPs.
8. If you are building a Retail image, you will need to get a [code-signing certificate](#). For kernel driver signing, a

Standard Code signing certificate is sufficient. You will require an EV cert to access the [Device Update Center](#) in the Hardware Dev Center portal.

### IMPORTANT

We recommend that you purchase your code signing certificate from a Certificate Authority listed from the link above. Building Windows 10 IoT Core **retail image** or image with Secure Boot enabled requires that all kernel drivers (including all drivers in the BSP that's included in the retail image) are signed with a code signing certificate with Cross Signed root. See [Cross-Certificates for Kernel Model Code Signing](#) and the FAQ section "Cross-Signing and SHA-256 Certificates" in [code-signing certificate](#).

As an example, if you choose to purchase through [DigiCert](#), you must choose the "Microsoft Kernel-Mode Code" option in the Platform section:



## Code Signing Certificates for Sysdevs

DigiCert offers code signing certs to Sysdevs at 50% off our regular pricing. Click the "Start" button to get into the order form with the pricing applied.

[Start](#)

Other DigiCert stuff you might like

- **DigiCert Certificate Inspector Service**

In March 2014 we launched a cool new service to help you monitor your SSL certificates and make sure they're installed securely. You can find out more about it at <https://www.digicert.com/cert-inspector.htm>

- **DigiCertUtil.exe tool**

If you work with certificates on Microsoft Windows Server platforms, our SSL certificate management utility is super useful: <https://www.digicert.com/util/> You can do all kinds of interesting tests and fixes with it. Troubleshoot revocation connectivity issues, import, export, and repair certificates, and fix intermediate certificate problems.



## DigiCert Order Form - Purchase Digital Certificates

1 Select Product

2 Organization Information

3 Payment

Already have a DigiCert account? [Log in](#)

### Step 1: Select a Product

#### Product

##### *Code Signing*

- Code Signing Certificate**  
 EV Code Signing Certificate

Increase trust by verifying the source and integrity of your applications using a DigiCert Code Signing Certificate.

- ✓ Supports Microsoft Authenticode, Office VBA, Java, Adobe AIR, Apple's Mac OS, and Mozilla objects.
- ✓ Avoid unnecessary warning messages and ensure that your applications can be trusted

#### Validity Period

Save money and time by choosing a longer validity period

- 1 Year \$ XXX USD / year**  
 2 Year \$ XXX USD / year  
 3 Year \$ XXX USD / year

#### Platform

Selecting the correct platform helps us issue your Code Signing Certificate in the correct format, but the certificate can be used to sign code for any of the listed platforms

Microsoft Kernel-Mode Code ▾

Other helpful software:

- **A text editor such as Notepad++.** You can also use the Notepad tool, though for some files, you won't see the line breaks unless you open each file as a UTF-8 file.
- **A compression program such as 7-Zip,** which can uncompress Windows app packages.
- [Visual Studio 2017](#), used to create an app in [Lab 1b: Add an app to your image](#).

## Other software

- **An app built for IoT Core.** Our samples use the [IoT Core Default](#) app, though you can use your own.
- **A driver built for IoT Core.** Our samples use the [GPIO KMDF demo](#) driver, though you can use your own.

## Next steps

[Lab 1a: Create a basic image](#)

# Lab 1a: Create a basic image

5/7/2019 • 11 minutes to read

To get started, we'll create a basic Windows 10 IoT Core (IoT Core) image and flash it onto a specific hardware device.

We'll create a product folder that represents our first design. For our first product design, we'll customize just enough for the IoT core device to boot up and run the built-in OOBE app, which we should be able to see on an HDMI-compatible monitor.

To make running these commands easier, we'll install and use the IoT Core shell, which presets several frequently-used paths and variables.

## Prerequisites

make sure your technician PC has the necessary tools installed prior to creating an IoT Core image.

See [Get the tools needed to customize Windows IoT Core](#) to get your technician PC ready.

### IMPORTANT

The version of ADK used must match the version of IoT Core Packages used below.

You will need the following tools installed to complete this section:

- Windows Assessment and Deployment Kit (Windows ADK)
- Windows 10 IoT Core packages
- IoT Core ADK Add-Ons
- IoT Core PowerShell environment
- A text editor like Notepad or VS Code

## Create a Basic Image

### Create a Workspace

1. In Windows Explorer, go to the folder where you installed the IoT Core ADK Add-Ons, for example, **C:\IoT-ADK-AddonKit**, and open **IoTCorePShell.cmd**. It should prompt you to run as an administrator.

This will load the PowerShell module and also check the versions of the ADK, IoT Core kit. This will also check for the test certificates in the certificate store and if not present, install them automatically.

Troubleshooting: Error: "The system cannot find the path specified". If you get this, right-click the icon and modify the path in "Target" to the location you've chosen to install the tools.

2. In the IoTCorePShell, create a new workspace say `C:\MyWorkspace` with an OEM name `Contoso` for the architecture `arm` using [New-IoTWorkspace](#)

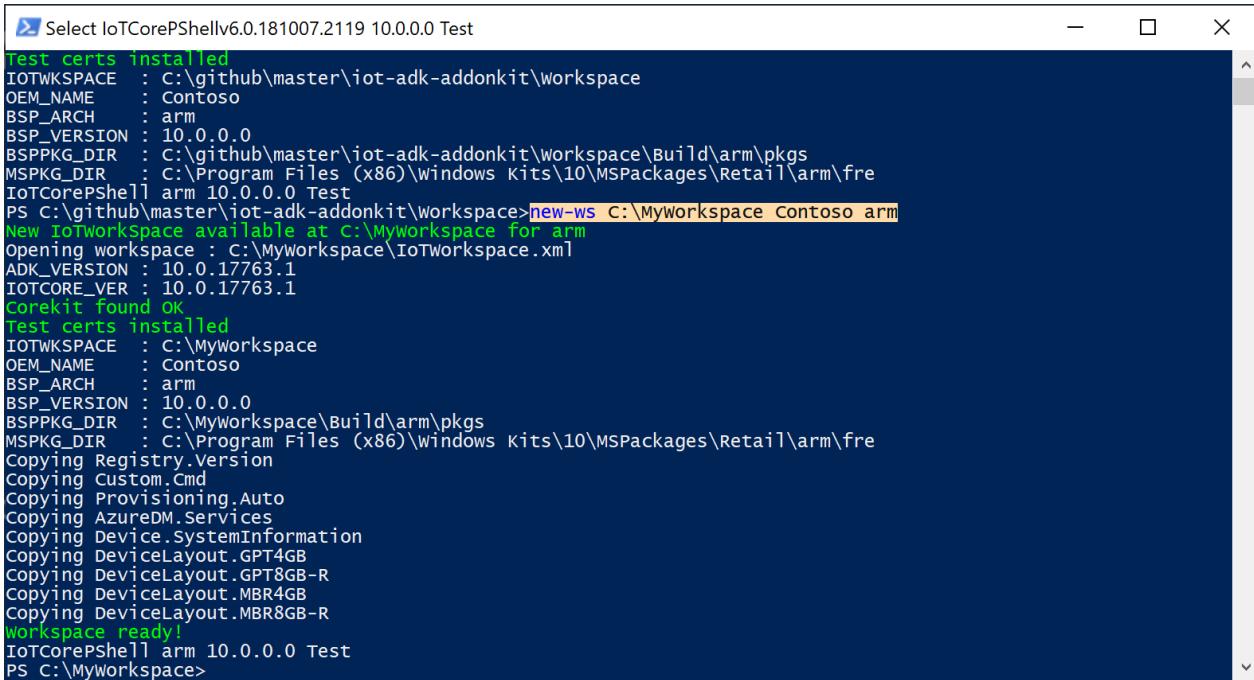
```
New-IoTWorkspace C:\MyWorkspace Contoso arm  
(or) new-ws C:\MyWorkspace Contoso arm
```

IoT Core supports four architectures, x64,x86,arm and arm64.

Only alphanumeric characters are supported in the OEM name as this is used as a prefix for various generated file names.

This generates the IoTWorkspace.xml and sets a version number for the design, which you can use for future updates. The first version number defaults to 10.0.0.0. (Why a four-part version number? Learn about versioning schemes in [Update requirements](#)).

The required packages such as Registry.Version, Custom.Cmd and Provisioning.Auto will be imported into the workspace automatically.



```
Test certs installed
IOTWSPACE  : C:\github\master\iot-adk-addonkit\workspace
OEM_NAME   : Contoso
BSP_ARCH   : arm
BSP_VERSION : 10.0.0.0
BSPPKG_DIR  : C:\github\master\iot-adk-addonkit\workspace\Build\arm\pkgs
MSPKG_DIR   : C:\Program Files (x86)\Windows Kits\10\MSPackages\Retail\arm\fre
IoTCoreshell arm 10.0.0.0 Test
PS C:\github\master\iot-adk-addonkit\workspace>new-ws C:\Myworkspace Contoso arm
New IoTworkspace available at C:\Myworkspace for arm
Opening workspace : C:\Myworkspace\IoTworkspace.xml
ADK_VERSION : 10.0.17763.1
IOTCORE_VER : 10.0.17763.1
Corekit found OK
Test certs installed
IOTWSPACE  : C:\Myworkspace
OEM_NAME   : Contoso
BSP_ARCH   : arm
BSP_VERSION : 10.0.0.0
BSPPKG_DIR  : C:\Myworkspace\Build\arm\pkgs
MSPKG_DIR   : C:\Program Files (x86)\Windows Kits\10\MSPackages\Retail\arm\fre
Copying Registry.Version
Copying Custom.Cmd
Copying Provisioning.Auto
Copying AzureDM.Services
Copying Device.SystemInformation
Copying DeviceLayout.GPT4GB
Copying DeviceLayout.GPT8GB-R
Copying DeviceLayout.MBR4GB
Copying DeviceLayout.MBR8GB-R
workspace ready!
IoTCoreshell arm 10.0.0.0 Test
PS C:\Myworkspace>
```

3. Import sample packages into the workspace. You may choose to import the packages selectively or import all.

```
# Importing Recovery packages from sample workspace
Import-IoTOEMPackage Recovery.*
(or) importpkg Recovery.*

# Below example imports all packages from the sample workspace
Import-IoTOEMPackage *
(or) importpkg *
```

## Import a Raspberry Pi BSP

This next step is to take the Board Support Package files and extra/build their .CAB files to include in the FFU file.

1. Download [RPi\\_BSP.zip](#) to a local directory, say `C:\Downloads\RPi_BSP.zip`.

2. Import the bsp using [Import-IoTBSP](#)

```
Import-IoTBSP RPi2 C:\Downloads\RPi_BSP.zip
(or) importbsp RPi2 C:\Downloads\RPi_BSP.zip
```

For more information on available BSPs, see [Windows 10 IoT Core BSPs](#).

## Create a test project

From the **IoT Core Shell Environment**, create a new product folder that uses the Raspberry Pi 2 BSP. This folder represents a new device we want to build, and contains sample customization files that we can use to start our project. Create a new product using [Add-IoTPProduct](#)

```

``` powershell
Add-IoTProduct ProductA RPi2
(or) newproduct ProductA RPi2
```
You will be prompted to enter the **SMBIOS** information such as Manufacturer name (OEM name), Family, SKU, BaseboardManufacturer and BaseboardProduct. Here are some example values:

```

- **System OEM Name:** Fabricam
- **System Family Name:** FabricamHub
- **System SKU Number:** AI-001
- **Baseboard Manufacturer:** Arrow
- **Baseboard Product:** Raspberry Pi 2

The BSP name is the same as the folder name for the BSP. You can see which BSPs are available by looking in the `C:\MyWorkspace\Source-<arch>\BSP` folders.

This creates the folder: `C:\MyWorkspace\Source-<arch>\Products\\ProductA`.

```

Select IoTCorePShellv6.0.181007.2119 10.0.0.0 Test
Copying AzureDM.services
Copying Device.SystemInformation
Copying DeviceLayout.GPT4GB
Copying DeviceLayout.GPT8GB-R
Copying DeviceLayout.MBR4GB
Copying DeviceLayout.MBR8GB-R
workspace ready!
IoTCorePShell arm 10.0.0.0 Test
PS C:\MyWorkspace>importbsp RPi2 "C:\Temp\RPi_BSP.zip"
Extracting BSP zip into the temporary directory C:\MyWorkspace\BSPDIR
Processing C:\Myworkspace\BSPDIR\RPi2
No .pkg.xml files found in C:\Myworkspace\source-arm\BSP\RPi2.
BSP copy completed
IoTCorePShell arm 10.0.0.0 Test
PS C:\MyWorkspace>newproduct ProductA RPi2

cmdlet Add-IoTProduct at command pipeline position 1
Supply values for the following parameters:
OemName: Contoso
FamilyName: SampleFamily
SkuNumber: P001
BaseboardManufacturer: RaspberryPi
BaseboardProduct: RPi2
Creating ProductA Product with BSP RPi2
Creating C:\Myworkspace\Source-arm\Products\ProductA\prov\customizations.xml...
Creating C:\Myworkspace\Source-arm\Products\ProductA\ProductASettings.xml...
Product Config file : C:\Myworkspace\Source-arm\Products\ProductA\ProductASettings.xml
DeviceInventory file : C:\Myworkspace\Source-arm\Products\ProductA\IoTDeviceModel_ProductA.xml
OS Version       : 10.0.17763.1
BSP Version      : 10.0.0.0
DeviceInventory created
IoTCorePShell arm 10.0.0.0 Test
PS C:\MyWorkspace>

```

### OemCustomization.cmd File

Every image includes a file `oemcustomization.cmd` which will run on every boot up of your device. You have the ability to modify this file to customize what executes on boot up. This file is located under

`C:\MyWorkspace\Source-<arch>\Products\\ProductA` in this example. The contents of the file is as follows:

```

@echo off
REM OEM Customization Script file
REM This script if included in the image, is called everytime the system boots.

reg query HKLM\Software\IoT /v FirstBootDone >nul 2>&1

if %errorlevel% == 1 (
    REM Enable Administrator User
    net user Administrator p@ssw0rd /active:yes
    if exist C:\Data\oobe (
        call folderpermissions.exe 'C:\Data\oobe -e'
    )
REM - Enable the below if you need secure boot/bitlocker
REM Enable Secureboot
REM if exist c:\IoTSec\setup.secureboot.cmd (
REM     call c:\IoTSec\setup.secureboot.cmd
REM )

REM Enable Bitlocker
REM if exist c:\IoTSec\setup.bitlocker.cmd (
REM     call c:\IoTSec\setup.bitlocker.cmd
REM )
    reg add HKLM\Software\IoT /v FirstBootDone /t REG_DWORD /d 1 /f >nul 2>&1
)

REM The below should be called on every boot
if exist C:\RecoveryConfig\Recovery.BcdEdit.cmd (
    call C:\RecoveryConfig\Recovery.BcdEdit.cmd
)

REM Set the crashdump file locations to data partition, set on every boot.
reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CrashControl" /v DedicatedDumpFile /t REG_SZ /d
C:\Data\DedicatedDumpFile.sys /f
reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CrashControl" /v DumpFile /t REG_SZ /d
C:\Data\MEMORY.DMP /f

```

#### **NOTE**

Please be aware that security features like BitLocker and SecureBoot are disabled by default for a custom test image. If you wish to include these features (in a retail image) you can uncomment out the appropriate lines in the file, prior to building your image.

#### **NOTE**

Please be aware that the commands in this file run with local system privilege.

## **Build Packages**

From the IoT Core Shell, get your environment ready to create products by building all of the packages in the working folders using [New-IoTCabPackage](#):

```

``` powershell
New-IoTCabPackage All
(or) buildpkg All
```

```

#### NOTE

If you get SignTool errors when building the packages in Test mode, please run `installoemcerts.cmd` to install the test certificates on your Technician PC.

## Build an image

1. Eject any removable storage drives, including the Micro SD card and any USB flash drives.
2. Build the FFU image file by entering the following command in the IoT Core PowerShell Environment using [New-IoTFFUImage](#):

```
New-IoTFFUImage ProductA Test  
(or) buildimage ProductA Test
```

This builds an FFU file with your basic image at `C:\MyWorkspace\Build\<arch>\ProductA\Test`. This test image will include additional tools that can be used for debugging purposes. Building the final FFU file will take around 10 to 30 minutes to complete.

To direct all output to console instead of log file, add `-Verbose` flag, such as the one below.

```
new-IoTFFUImage -Verbose ProductX Test
```

#### NOTE

If you get either 0x80070005 or 0x800705b4 as an error code, unplug all external drives (including micro SD cards and USB thumb drives), and try again. If this doesn't work, go back to [Set up your PC and download the samples](#) and make sure everything's installed.

## Flash a Windows IoT Core image

Now that your FFU image file is built, you can use this file to flash and deploy the image to your device. Listed below are the steps to flash the FFU image file to specific manufacturer devices.

### Prerequisites/Requirements

You will need the following tools installed to complete this section:

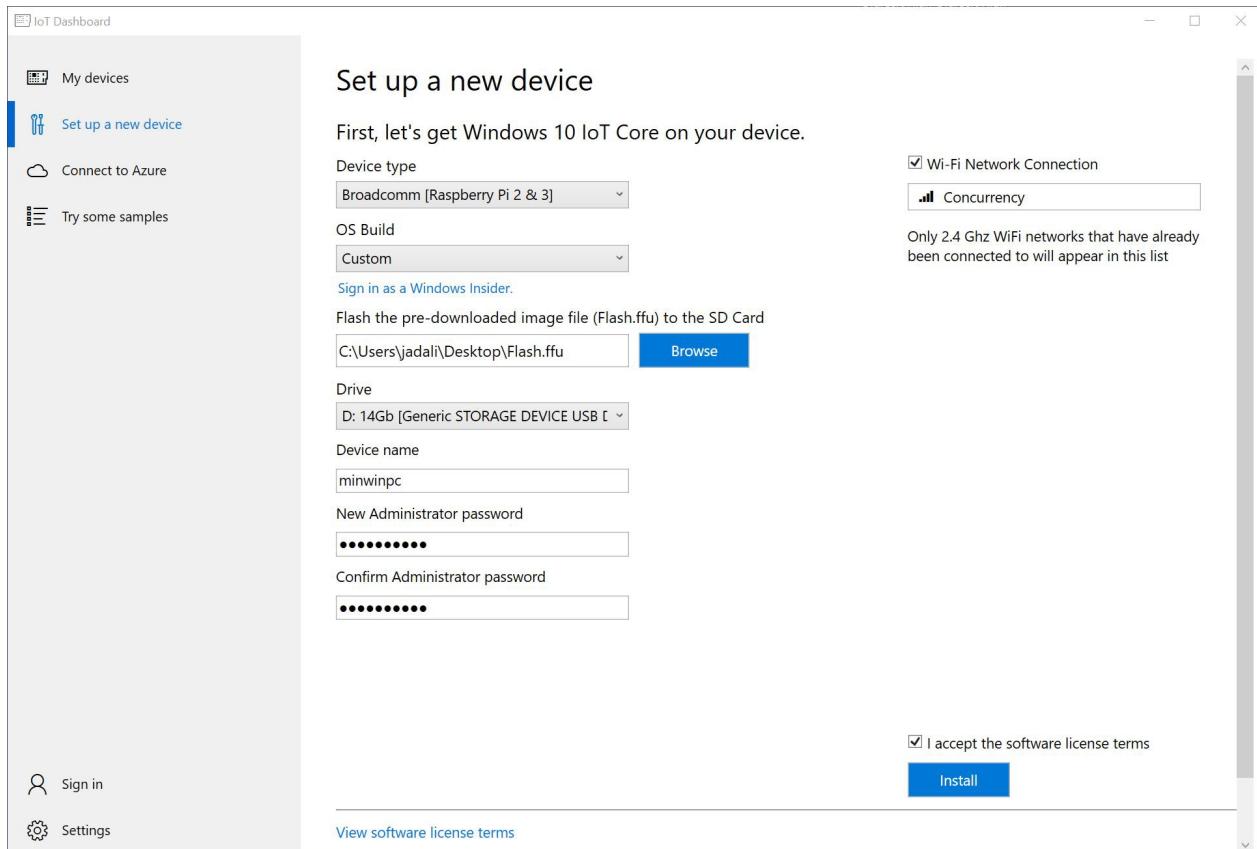
- Windows IoT Core Dashboard (Raspberry Pi only)
- Dragonboard Update Tool (Qualcomm DragonBoard only)
- Deployment Imaging and Servicing and management (DISM) tool (Intel devices only)

Since we're using Raspberry Pi here, we'll start with Raspberry Pi but also explain how to flash images on other devices.

#### Raspberry Pi

1. Start the **Windows IoT Core Dashboard**.
2. Plug your micro SD card into your technician PC, and select it in the tool.
3. From the **Setup a new device** section, select **Broadcom [Raspberry Pi 2 & 3]** under **Device Type**.
4. Select **Custom** under OS Build.
5. Click **Browse** and navigate and select the FFU file you created earlier. In this case, your FFU file is located at `C:\MyWorkspace\Build\<arch>\ProductA\Test\Flash.ffu`. Then click **Next**.

6. Verify that your microSD card is listed under the **Drive** selection.
7. Optionally, you can set the **Device Name** and **Administrator Password** selections for your device.
8. Check the **I accept the software license terms** checkbox (lower right) and click **Install**.



Windows IoT Core Dashboard will now open a command window and use DISM (Deployment Image Servicing and Management Tool) to flash the FFU file to your microSD card.

```
C:\WINDOWS\system32\dism.exe

Deployment Image Servicing and Management tool
Version: 10.0.17134.1

Applying image
[===== 9.0% ]
```

## NOTE

Alternatively, you can also use the **DISM** command to manually flash the image:

```
dism.exe /Apply-Image /ImageFile:"D:\flash.ffa" /ApplyDrive:\.\PhysicalDrive0 /SkipPlatformCheck
```

Once the flashing process is complete, eject the microSD card from the technician PC and insert it into the Raspberry Pi. Reconnect the power to boot Windows IoT Core. You should see the [IoT Core Default app](#), which shows basic information about the image.

## Qualcomm

### DragonBoard 410c

Qualcomm has provided the [Dragonboard Update Tool](#) application, which we will use to flash the FFU image file to the Dragonboard 410c. You can download either the x86 or x64 version (we recommend x64).

[DragonBoard 410c Projects](#)

#### Linux

- [› Linux Board Support Package r1034.2.1](#)
- [› Linux Board Support Package r1032.1.1](#)
- [› Linux Board Support Package r1032.1](#)
- [› Linux Bootloader & Firmware v1030.4 + PSCI](#)
- [› Linux Board Support Package v1.4](#)
- [› Linux Board Support Package v1.3](#)
- [› Linux Board Support Package v1.2](#)
- [› Linux Board Support Package v1.1](#)
- [› Qualcomm Snapdragon 410E r1034.2.1 Linux Embedded Software Release Notes Rev. C](#)

#### Microsoft Azure IoT

- [› Microsoft Azure Internet of Things Suite](#)
- [› DragonBoard 410c Projects](#)

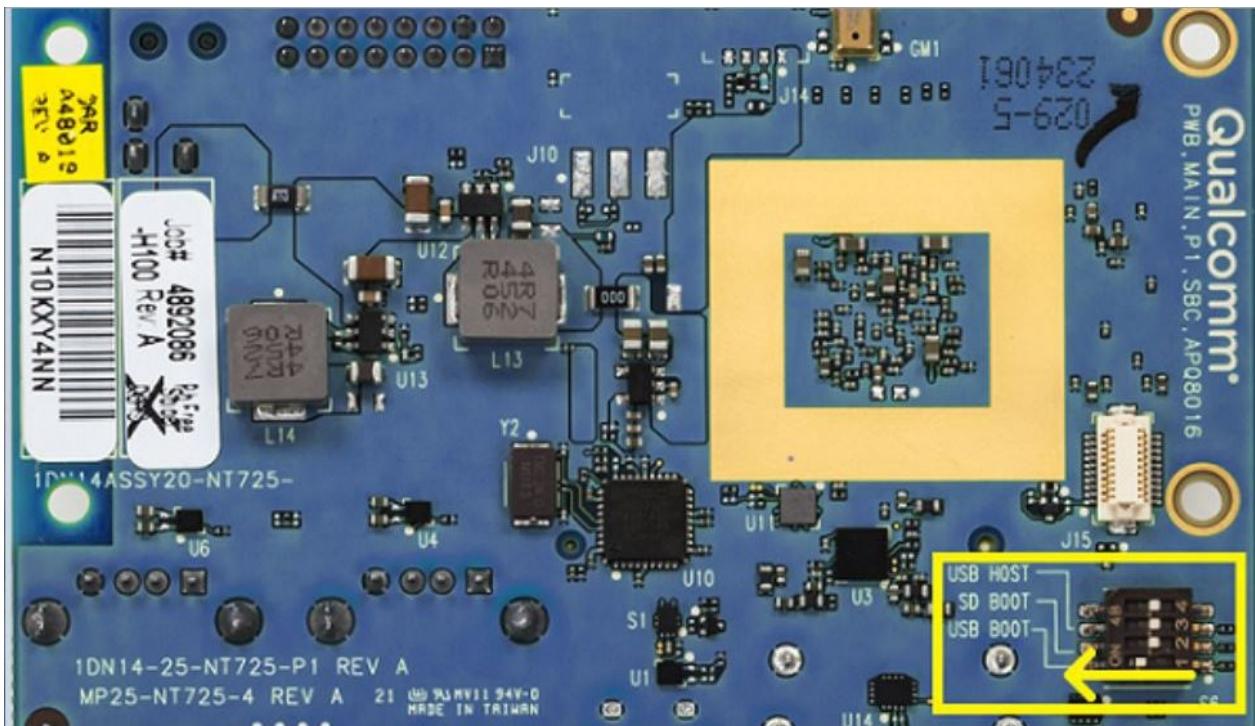
#### ROS

- [› Robotic Operating System \(ROS\) Debian Install](#)
- [› DragonBoard 410c Projects](#)

#### Windows 10 IoT Core

- [› Windows 10 IoT Core Board Support Package](#)
- [› Windows 10 IoT BSP for DragonBoard 410c Customization Guide](#)
- [› Windows 10 IoT Update Tool for DragonBoard 410c x86 \(32 bit\)](#)
- [› Windows 10 IoT Update Tool for DragonBoard 410c x64 \(64 bit\)](#)
- [› Windows 10 - Getting Started](#)

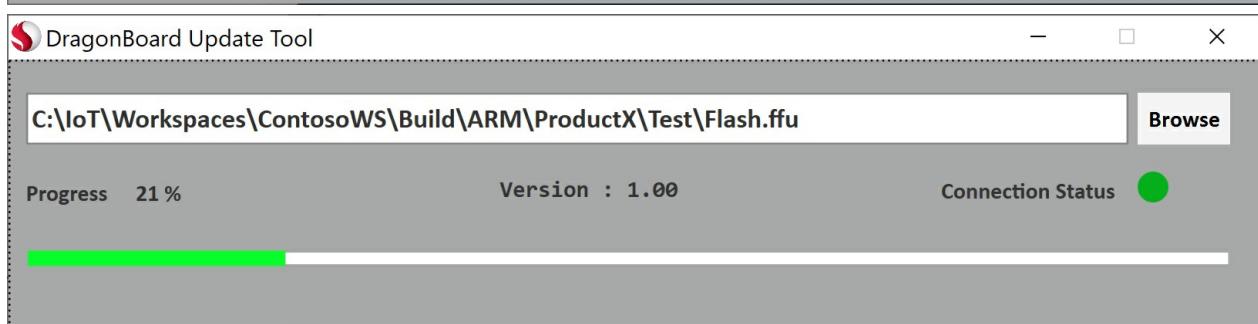
Once you have installed the **DragonBoard Update Tool**, you must first place the DragonBoard device in programming mode. This is done by setting dipswitch #1 (on the backside of the device) to the **ON** position:



Connect the DragonBoard device to your technician PC with a microUSB cable and power on the board.

Run the **DragonBoard Update Tool**, browse for the FFU image file you are deploying, and press the **Program** button. You should see a green **Connection Status**, and a progress bar when the device is flashing the FFU file.

Once the flashing process is complete, disconnect power from the board, as well as the microUSB cable. You then need to flip the dipswitch #1 back to the **OFF** position. Reconnect the power to boot Windows IoT Core.



## Intel

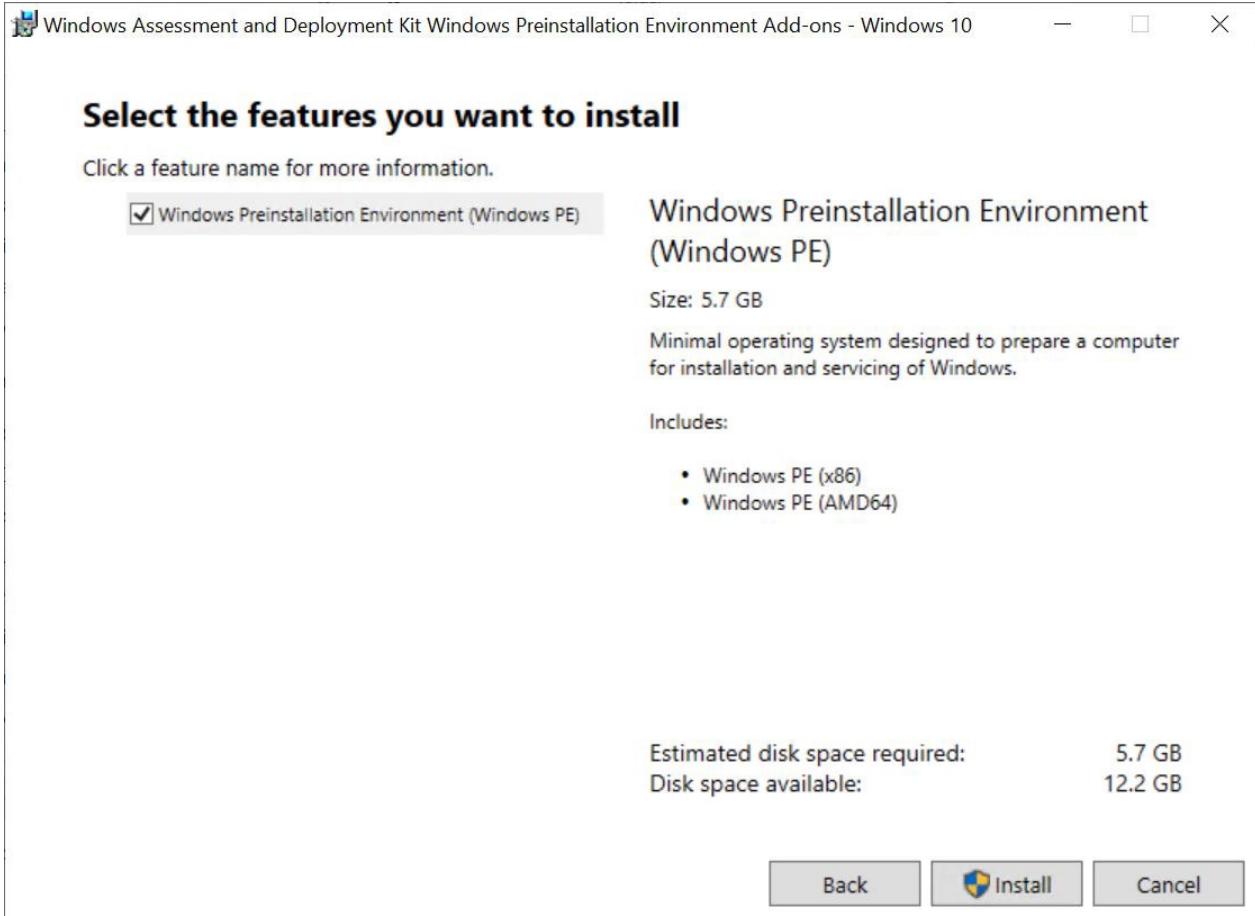
[Apollo Lake/Braswell/Cherry Trail](#)

We will be using the DISM (Deployment Image and Servicing Management Tool) and a bootable USB stick to flash the FFU image file to the specified Intel device (Apollo Lake/Braswell/Cherry Trail). Additional information on DISM can be found [here](#).

### Creating a USB Bootable Drive

We first need to create a bootable USB drive that we can use to boot on the specified Intel hardware device. We can use Window PE (WinPE) for this (additional info on WinPE is [here](#)).

1. Install WinPE from the [Windows ADK Windows Preinstallation Environment Add-ons](#).



## 2. Prepare USB Drive

The USB drive you wish to use must be properly formatted before we can load WinPE on it. Follow these steps to properly format your USB drive:

- Run the `diskpart` tool from an administrative command prompt
- Run `list disk` to see the list of available disks
- Run `select disk X`, where X corresponds to the disk number of your USB drive
- Run `clean` to clean the selected disk

### NOTE

If you encounter the following error, run `convert mbr`:

```
ERROR: Failed to format "F:"; DiskPart errorlevel -2147212244
```

- Run `create partition primary` to create a primary partition on the disk
- Run `format fs=fat32 quick` to format the drive
- Run `assign` to assign the drive
- Exit `diskpart`. Your USB drive is now formatted and ready to have WinPE installed.

## 3. Create WinPE working files

We now need to create a working copy of the Windows PE files on the technician PC. This can be done using the **Deployment and Imaging Tools Environment** command line program. The easiest way to start the

`Deployment and Imaging Tools Environment` is to search for deploy via Start Menu.

Run the **Deployment and Imaging Tools Environment** command line program as an administrator, and run

the following command:

```
copype amd64 C:\WinPE_amd64
```

This will create the working copy of Windows PE files at **C:\WinPE\_amd64**.

Insert your USB drive on he technician PC and run this command to install Windows PE on the USB drive. Make sure you can replace X with the drive letter of your USB drive:

```
MakeWinPEMedia /UFD C:\WinPE_amd64 X:
```

4. Copy your FFU file to the root drive of your USB drive.
5. Insert your USB drive into your Intel hardware device and boot from the USB drive. You may have to enter the BIOS (or Boot Menu) of the hardware device to specify to boot from a USB drive.
6. Once the Windows PE environment boots up, you will see a command window. Change the drive and current directory to the location of your FFU file (depending on drive enumeration, it could be in C: or D: ..etc.) and run the following command to flash the FFU image file:

```
dism.exe /Apply-Image /ImageFile:"C:\IoT\Workspaces\ContosoWS\Build\ARM\ProductX\Test\flash.ffu"  
/ApplyDrive:\\.\PhysicalDrive0 /SkipPlatformCheck
```

#### NOTE

Most of the time, the on-board storage target to be flashed is enumerated as `PhysicalDrive0`, however, if there are multiple storage devices, it can enumerate as other drive number. You can use the `list disk` command in `diskpart` to verify the drive number.

8. Once the flashing process is complete, power down the hardware device and remove the USB drive.  
Reconnect power to the hardware device to boot up Windows IoT Core.

[Diskpart Commands](#)

```
C:\>diskpart
Microsoft DiskPart version 10.0.17134.1
Copyright (C) Microsoft Corporation.
On computer: LWIN-CNCY-BUILD

DISKPART> list disk

Disk ### Status Size Free Dyn Gpt
----- -----
Disk 0 Online 447 GB 1024 KB *
Disk 1 Online 167 GB 1024 KB *
Disk 2 Online 14 GB 0 B *

DISKPART> select disk 2
Disk 2 is now the selected disk.

DISKPART> clean
DiskPart succeeded in cleaning the disk.

DISKPART> convert mbr
DiskPart successfully converted the selected disk to MBR format.

DISKPART> create partition primary
DiskPart succeeded in creating the specified partition.

DISKPART> format fs=fat32 quick
100 percent completed

DiskPart successfully formatted the volume.

DISKPART> assign
DiskPart successfully assigned the drive letter or mount point.

DISKPART> exit
```

#### WinPE Commands

```
copype amd64 C:\WinPE_amd64
MakeWinPEMedia /UFD C:\WinPE_amd64 X:
```

#### DISM Command (via WinPE on the Intel hardware device)

```
X:\WinPE>d:
D:\>dism.exe /Apply-Image /ImageFile:"C:\IoT\Workspaces\ContosoWS\Build\ARM\ProductX\Test\flash.ffa"
/ApplyDrive:\.\PhysicalDrive0 /SkipPlatformCheck
```

## Next steps

Leave the device on for now, and continue to [Lab 1b: Add an app to your image](#).

# Lab 1b: Add an app to your image

5/14/2019 • 7 minutes to read

We are now going to take an app (like the [IoT Core Default](#) app), package it up, and create a new Windows IoT Core image you can load onto your device.

## NOTE

This process is identical for both background and foreground apps. The only difference to be aware of is that only one foreground app can be selected as the startup app, and all other installed apps will run as background apps.

## Prerequisites/Requirements

## NOTE

As you go through this manufacturing guide, ProjectA will start to resemble the SampleA image that's in C:\IoT-ADK-AddonKit\Source-arm\Products\SampleA.

We'll use the ProjectA image we created from [Lab 1a: Create a basic image](#).

You will also need the following tools to complete this section:

- Visual Studio
- Windows Assessment and Deployment Kit (Windows ADK)
- Windows IoT Core Packages
- IoT Core ADK Add-Ons
- IoT Core Powershell Environment
- A text editor like Notepad or VS Code

## Supported Application Types

### Universal Windows Platform (UWP) Apps

IoT Core is a UWP centric OS and UWP apps are its primary app type.

Universal Windows Platform (UWP) is a common app platform across all versions of Windows 10, including Windows 10 IoT Core. UWP is an evolution of Windows Runtime (WinRT). You can find more information and an overview to UWP [here](#).

### Traditional UWP Apps

UWP apps just work on IoT Core, just as they do on other Windows 10 editions. A simple, blank Xaml app in Visual Studio will properly deploy to your IoT Core device just as it would on a phone or Windows 10 PC. All of the standard UWP languages and project templates are fully supported on IoT Core.

There are a few additions to the traditional UWP app-model to support IoT scenarios and any UWP app that takes advantage of them will need the corresponding information added to their manifest. In particular the "iot" namespace needs to be added to the manifest of these standard UWP apps.

Inside the attribute of the manifest, you need to define the iot xmlns and add it to the IgnorableNamespaces list. The final XML should look like this:

```
<Package
  xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10"
  xmlns:mp="http://schemas.microsoft.com/appx/2014/phone/manifest"
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
  xmlns:iot="http://schemas.microsoft.com/appx/manifest/iot/windows10"
  IgnorableNamespaces="uap mp iot">
```

## Background Apps

In addition to the traditional UI apps, IoT Core has added a new UWP app type called "Background Applications". These applications do not have a UI component, but instead have a class that implements the "IBackgroundTask" interface. They then register that class as a "StartupTask" to run at system boot. Since they are still UWP apps, they have access to the same set of APIs and are supported from the same language. The only difference is that there is no UI entry point.

Each type of IBackgroundTask gets its own resource policy. This is usually restrictive to improve battery life and machine resources on devices where these background apps are secondary components of foreground UI apps. On IoT devices, Background Apps are often the primary function of the device and so these StartupTasks get a resource policy that mirrors foreground UI apps on other devices.

You can find in-depth information on Background apps on [MSDN](#).

## Non-UWP (Win32) Apps

IoT Core supports certain traditional Win32 app types such as Win32 Console Apps and NT Services. These apps are built and run the same way as on Windows 10 Desktop. Additionally, there is an IoT Core C++ Console project template to make it easy to build such apps using Visual Studio.

There are two main limitations on these non-UWP applications:

1. No legacy Win32 UI support: IoT Core does not contain APIs to create classic (HWND) Windows. Legacy methods such as CreateWindow() and CreateWindowEx() or any other methods that deal with Windows handles (HWNDs) are not available. Subsequently, frameworks that depend on such APIs including MFC, Windows Forms and WPF, are not supported on IoT Core.
2. C++ Apps Only: Currently, only C++ is supported for developing Win32 apps on IoT Core.

## App Service

App services are UWP apps that provide services to other UWP apps. They are analogous to web services, on a device. An app service runs as a background task in the host app and can provide its service to other apps. For example, an app service might provide a bar code scanner service that other apps could use. App services let you create UI-less services that apps can call on the same device, and starting with Windows 10, version 1607, on remote devices. Starting in Windows 10, version 1607, you can create app services that run in the same process as the host app.

Additional information regarding creating a background app service as well as consuming the service from a uwp apps (as well as background tasks/services) can be found [here](#).

## Extend your app with services, extensions, and packages

There are many technologies in Windows 10 for extending and componentizing your app. This table should help you determine which technology you should use depending on requirements. It is followed by a brief description of the scenarios and technologies.

| SCENARIO                                   | RESOURCE PACKAGE | ASSET PACKAGE | OPTIONAL PACKAGE | FLAT BUNDLE | APP EXTENSION | APP SERVICE | STREAMING INSTALL |
|--|------------------|---------------|------------------|-------------|---------------|-------------|-------------------|
| Third-party code plug-ins                  |                  |               |                  |             | ✓□            |             |                   |
| In-proc code plug-ins                      |                  |               | ✓□               |             |               |             |                   |
| UX Assets (strings/images)                 | ✓□               | ✓□            | ✓□               |             | ✓□            |             | ✓□                |
| On demand content (e.g. additional Levels) |                  |               | ✓□               |             | ✓□            |             | ✓□                |
| Separate licensing and acquisition         |                  |               | ✓□               |             | ✓□            | ✓□          |                   |
| In-app acquisition                         |                  |               | ✓□               |             | ✓□            |             |                   |
| Optimize install time                      | ✓□               |               | ✓□               |             | ✓□            |             | ✓□                |
| Reduce disk footprint                      | ✓□               |               | ✓□               |             |               |             |                   |
| Optimize packaging                         |                  | ✓□            | ✓□               | ✓□          |               |             |                   |
| Reduce publishing time                     | ✓□               | ✓□            | ✓□               | ✓□          |               |             |                   |

## Create an appx package

The first step is to create a Universal Windows Platform (UWP) application that will run on the IoT device. You may skip this section if you've already created and tested your UWP application.

1. Create a UWP app. This can be any app designed for IoT Core, saved as an Appx Package. For our example, we are using the [IoT Core Default](#) app.
2. In Visual Studio, save your application as an Appx package. This is done by clicking **Project > Store > Create App Packages > I want to Create Packages for Sideload > Next**
3. Select **Output location** as C:\DefaultApp (or any other path that doesn't include spaces.)
4. Select **Generate app bundle:** Never
5. Click **Create**.

Create App Packages

## Select and Configure Packages

Output location:  
C:\Dev\OpenSource\ContosoApp\ContosoApp\AppPackages\

Version:  
1 . 0 . 1 . 0  
 Automatically increment  
[More information](#)

Generate app bundle:  
Never   
[What does an app bundle mean?](#)

Select the packages to create and the solution configuration mappings:

|                                     | Architecture | Solution Configuration |
|-------------------------------------|--------------|------------------------|
| <input type="checkbox"/>            | Neutral      | None                   |
| <input checked="" type="checkbox"/> | x86          | Debug (x86)            |
| <input checked="" type="checkbox"/> | x64          | Debug (x64)            |
| <input checked="" type="checkbox"/> | ARM          | Debug (ARM)            |
| <input checked="" type="checkbox"/> | ARM64        | Debug (ARM64)          |

Include full PDB symbol files, if any, to enable crash analytics for the app. [Learn More](#)

Visual Studio creates the Appx files in your specified location for the architecture(s) you selected (ARM, x86, x64). In our example, this file is: C:\DefaultApp\IoTCoreDefaultApp\_1.2.0.0\_ARM\_Debug\_Test

6. Optional: [Test the app](#). Note, you may have already tested the app as part of building the project.

## Package the Appx

The next step is to package the Appx file, which will allow you to customize it and build it using the Windows ADK (when you build the FFU image).

1. Open `IoTCorePShell.cmd`. It should prompt you to run as an administrator.
2. Create the package for your Appx by using [New-IoTAppxPackage](#). Replace the file path location and package name with your Appx package. In our example, the command is as follows:

```
Add-IoTAppxPackage
"C:\DefaultApp\IoTCoreDefaultApp_1.2.0.0_ARM_Debug_Test\IoTCoreDefaultApp_1.2.0.0_ARM_Debug_Test.appx"
fga Appx.MyUWPApp

(or) newappxpkg
"C:\DefaultApp\IoTCoreDefaultApp_1.2.0.0_ARM_Debug_Test\IoTCoreDefaultApp_1.2.0.0_ARM_Debug_Test.appx"
fga Appx.MyUWPApp
```

#### NOTE

The fga parameter indicates the Appx file is a foreground application. If you specify your package as a background application (with the bga parameter) and have no other foreground applications in the image, the system will get stuck when booting up (displays a spinner indefinitely).

This creates a new folder at `C:\MyWorkspace\Source-<arch>\Packages\Appx.MyUWPApp`, copies the appx files and its dependencies and generates a customizations.xml file as well as a package xml file that is used to build the package.

Be aware that if your Appx has dependencies you will need the Dependencies subdirectory to be present in the same location as your Appx when you run this command. Failure to include this will result in errors when you build your FFU image.

This also adds a FeatureID **APPX\_MYUWPAPP** to the `C:\MyWorkspace\Source-<arch>\Packages\OEMFM.xml` file.

- From the IoT Core Shell Environment, you can now build the package into a .CAB file using [New-IoTCabPackage](#).

```
New-IoTCabPackage Appx.MyUWPApp  
(or) buildpkg Appx.MyUWPApp
```

This will build the package into a .CAB file under

```
C:\MyWorkspace\Build\<arch>\pkgs\<oemname>.Appx.MyUWPApp.cab .
```

## Update the project's configuration files

You can now update your project configuration files to include your app in the FFU image build.

- Add the FeatureID for our app package using [Add-IoTProductFeature](#):

```
Add-IoTProductFeature ProductA Test APPX_MYUWPAPP -OEM  
(or) addfid ProductA Test APPX_MYUWPAPP -OEM
```

This adds a FeatureID called **APPX\_HELLOWORLDAPP** to the specified product's Test OEMInput XML file (`C:\IoT\Workspaces\ContosoWS\Source-arm\<product name>\TestOEMInput.xml` file).

- Remove the sample test apps IOT\_BERTHA using [Remove-IoTProductFeature](#)

```
Remove-IoTProductFeature ProductA Test IOT_BERTHA  
(or) removefid ProductA Test IOT_BERTHA
```

## Build and test the image

From the IoT Core PowerShell Environment, get your environment ready to create products by building all of the packages in the working folders (using New-IoTCabPackage):

```
New-IoTCabPackage All  
(or) buildpkg all
```

Build the FFU image again, as specified in [Lab 1a: Create a basic image](#). You can use this command:

```
New-IoTFFUImage ProductX Test  
(or)buildimage ProductX Test
```

Once the FFU file has been build (it should now include your app), you can flash it to your hardware device as specified in [Lab 1a](#).

To learn how to install an appx file on an IoT device, view the [Install your app](#) documentation for more information.

## Related topics

[Update apps on your IoT Core devices](#)

## Next steps

[Lab 1c: Add a file and a registry setting to an image](#)

# Lab 1c: Adding file(s) and registry settings to an image

5/9/2019 • 4 minutes to read

We will create some test files and registry keys to a Windows IoT Core image, and package them up so that they can be serviced after they are distributed to your customers. Since files and registry keys that you add to your image are often not specific to an architecture, we recommend creating a common package that you can use across all of your device architectures.

## Goals

- Create a package that contains registry and file settings for your device
- Package the registry/file settings package so it can be included in an FFU image
- Modify IoT Addon Kit project configuration files to include your registry/file settings package
- Build and deploy an FFU image that contains your registry/file settings package

We'll create some test files and registry keys to the image, and again package them up so that they can be serviced after they reach your customers.

We'll add these to the common feature manifest (OEMCommonFM.xml), which is used in x86, x64, and ARM builds.

For this lab, we'll start an new product, ProductB, so that later we can use the IoT sample app to get the IP address of our device and verify that our files and reg keys have made it.

## Prerequisites

Please make sure that you've created a basic image from [Create a basic image](<https://docs.microsoft.com/en-us/windows-hardware/manufacture/iot/create-a-basic-image>) previously.

You will need the following tools installed to complete this action:

- Windows Assessment and Deployment Kit (Windows ADK)
- IoT Core Powershell Environment
- Windows 10 IoT Core Packages
- IoT Core ADK Add-Ons
- A text editor like Notepad or VS Code.

## Create your test files

Create a few sample text files using Notepad and add some random text so that these files are not empty. For our example, we have created two files titled **TestFile1.txt** and **TestFile2.txt**.

## Build a package for Test Files

1. Open the **IoT Core PowerShell Environment**: run `C:\MyWorkspace\IoTCoresPShell.cmd` as an administrator and create a File package using [Add-IoTFilePackage](#):

```

# Array of files with destinationDir, Source and destinationFilename
$myfiles = @(
    ("`$(runtime.system32)","C:\Temp\TestFile1.txt", ""),
    ("`$(runtime.bootDrive)\OEMInstall","C:\Temp\TestFile2.txt", "TestFile2.txt")
)
Add-IoTFilePackage Files.Configs $myfiles

```

This creates a new folder at `C:\MyWorkspace\Common\Packages\Files.Configs`.

This also adds a FeatureID **FILES\_CONFIGS** to the `C:\MyWorkspace\Common\Packages\OEMCOMMONFM.xml` file.

Variables like `$(runtime.system32)` are defined in

`C:\Program Files (x86)\Windows Kits\10\Tools\bin\i386\pkggen.cfg.xml`.

## 2. Create a **Registry package** using [Add-IoTRegistryPackage](#):

```

# Array of files with destinationDir, Source and destinationFilename
$myregkeys = @(
    ("`$(hklm.software)\`$(OEMNAME)\Test","StringValue", "REG_SZ", "Test string"),
    ("`$(hklm.software)\`$(OEMNAME)\Test","DWordValue", "REG_DWORD", "0x12AB34CD")
)
Add-IoTRegistryPackage Registry.Settings $myregkeys

```

This creates a new folder at `C:\MyWorkspace\Common\Packages\Registry.Settings`.

This also adds a FeatureID **REGISTRY\_SETTINGS** to the `C:\MyWorkspace\Common\Packages\OEMCOMMONFM.xml` file.

## 3. Build the packages using [New-IoTCabPackage](#):

```

New-IoTCabPackage Files.Configs
(or) buildpkg Files.Configs
New-IoTCabPackage Registry.Settings
(or) buildpkg Registry.Settings

```

The package is built and is available at `C:\MyWorkspace\Build\<arch>\pkgs`.

## Create a new product

### 1. Create a new product folder.

```

Add-IoTProduct ProductB RPi2
(or) newproduct ProductB RPi2

```

This will prompt you to enter **SMBIOS** values.

## Update the project's configuration files

Update the product test configuration to include the features using [AddIoTProudctFeature](#):

```

Add-IoTProductFeature ProductX Test FILES_CONFIGS -OEM
(or) addfid ProductX Test FILES_CONFIGS -OEM
Add-IoTProductFeature ProductX Test REGISTRY_SETTINGS -OEM
(or) addfid ProductX Test REGISTRY_SETTINGS -OEM

```

## Build and test the image

Build the FFU image again, as specified in [Create a basic image](#). You should only have to run the `New-IoTFFUImage` command:

```
New-IoTFFUImage ProductX Test  
(or)buildimage ProductX Test
```

Once the FFU file has been built, you can flash it to your hardware device as specified in [Flashing a Windows IoT Core image](#).

## Verify Files and Registry Keys added

In order to verify that the files were added on the device, do the following:

1. Connect both your technician PC and the device to the same ethernet network.

For example, to connect over a wired network, plug in an ethernet cable. To connect directly to the device, plug a network cable directly from your technician PC to the device.

2. On the test app, note the IP address of the device. For example, 10.100.0.100.
3. On the technician PC, open File Explorer and type in the IP address of the device with a \ prefix and \c\$ suffix:

```
\\10.100.0.100\c$
```

Use the devicename, the default Administrator account, and password to log on. (Default is:  
minwinpc\Administrator / p@ssw0rd)

4. Check to see if the files exist on the device. For our example, look for:

```
\\10.100.0.100\c$\Windows\system32\TestFile1.txt
```

```
\\10.100.0.100\c$\OEM\Install\TestFile2.txt
```

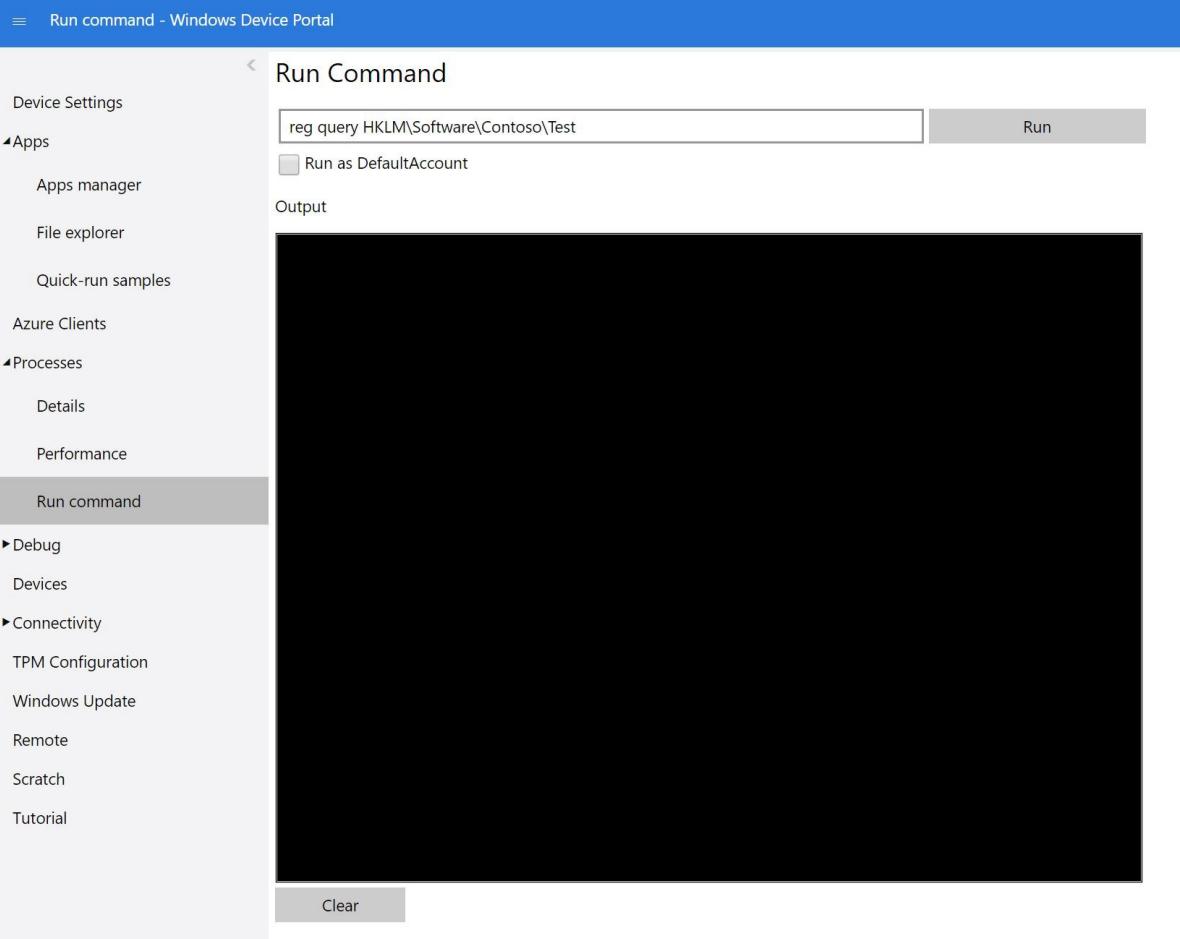
## Verify registry keys

1. On the technician PC, connect your device using an SSH client, such as [PuTTY](#). For example, use the IP address and port 22 to connect to the device. Then log in using the Administrator account and password. (To learn more, see [SSH](#)).
2. From the command line in the SSH client, query the system for the registry key. In our example, this command was executed to check the existence of the registry key:

```
reg query HKLM\Software\Fabrikam\Test
```

The SSH client should display your test values.

Alternatively, you can use the **Run Command** in the Windows Device Portal for your connected device to run the `reg query` command.



The output window should display your test values.

## Next steps

[Lab 1d: Add a provisioning package to an image](#)

# Lab 1d: Add networking and other provisioning package settings to an image

6/24/2019 • 5 minutes to read

A provisioning package allows you to apply customization settings over an existing Windows IoT Core installation image. We will describe the steps required to create a provisioning package that you can apply to your Windows 10 IoT Core FFU images.

## NOTE

To test WiFi, your board will need WiFi support. You can use a WiFi adapter/dongle or a board like the Raspberry Pi 3 that has Wi-Fi built-in.

For this lab, we'll use the ProductB, that includes the default app (Bertha), which shows network status.

## Prerequisites

Make sure you've created an image with your custom app from [Add an app to an image](#) previously.

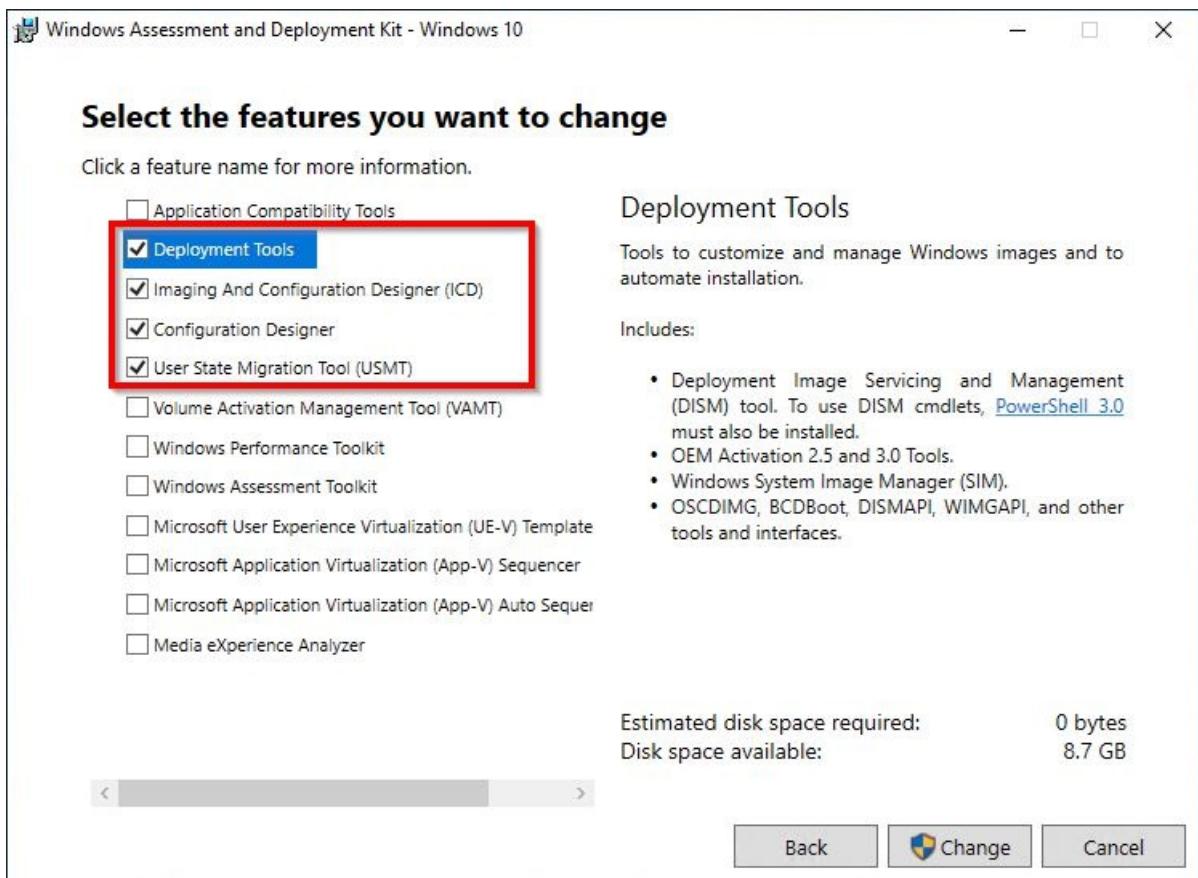
You will need the following tools installed to complete this section:

- Windows Assessment and Deployment Kit (Windows ADK)
- Windows Configuration Designer
- IoT Core PowerShell Environment
- A text editor like Notepad or VS Code

## Install Windows Configuration Designer

We will be using **Windows Configuration Designer (WCD)** to create a provisioning package for our IoT Core image. Windows Configuration Designer allows you to create provisioning packages, which are used to configure devices running Windows 10 IoT Core.

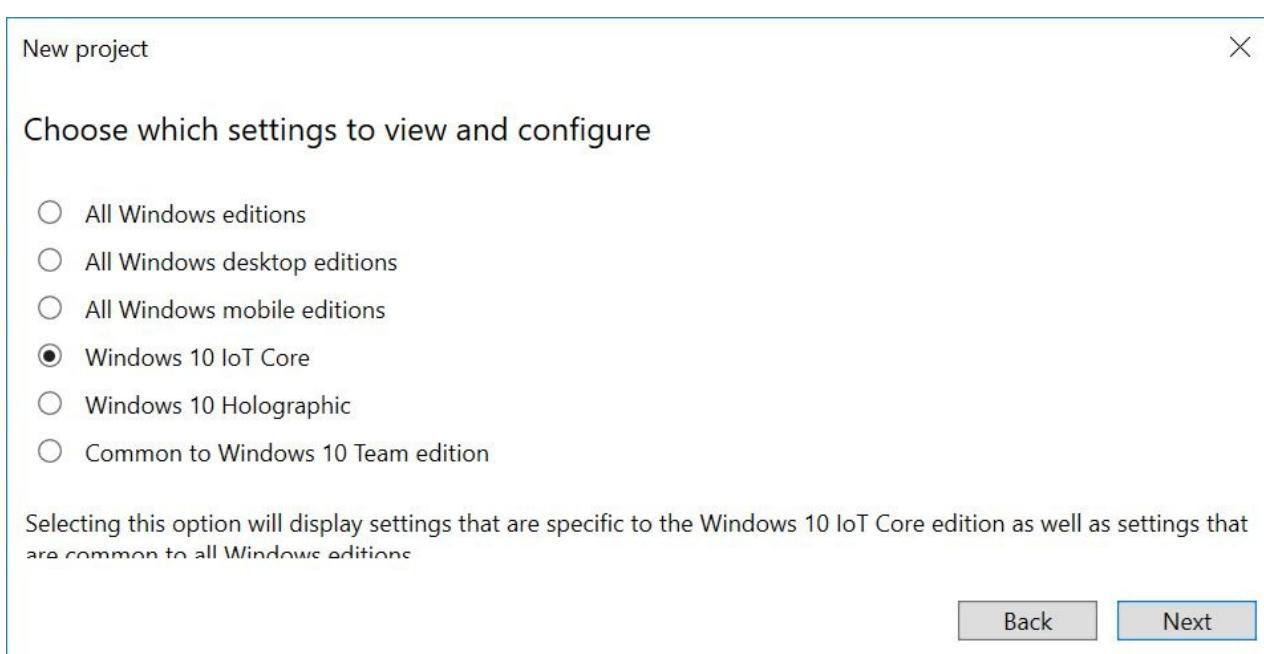
Windows Configuration Designer comes with the **Windows ADK Toolkit** and should have been installed to the Technician PC previously. If not, please run the install for the Windows ADK Toolkit and make sure you have the **Configuration Designer** selection checked for installation.



## Create WCD Project for a Provisioning Package

In order to create a provisioning package for your device, we need to create a project in **Windows Configuration Designer**. Once we have this project, we can specify the configuration customizations we want included in our FFU image.

1. From your IoT Core imaging PC (technician PC), run **Windows Imaging and configuration Designer**.
2. Create a new project by clicking **File > New Project**. For our example, we created a project called *TestProvPackage*.
3. Select **Provisioning Package** and click **Next**.
4. On the **Choose which settings to view and configure** page, select **Windows 10 IoT Core**. Click **Next**.



5. At the **Import a provisioning package (optional)** page, leave the entry blank and click **Finish**.
6. Add a sample setting:
  - a. Expand **Runtime settings > Connectivity Profiles > WLAN > WLANSetting > SSID**.
  - b. Type in the name of a Wi-Fi network name, for example, ContosoWiFi, and click Add.
  - c. Expand the **SSID > WLANXmlSettings > SecurityType** and choose a setting such as Open.
  - d. Expand the **SSID > WLANXmlSettings > AutoConnect** and choose a setting such as **TRUE**.
  - e. Optional: To add more than one WLAN network, go back to WLANSetting, and repeat the process.
7. Optional: Add other apps, drivers, and settings through the UI. To learn more, see [Configure customizations using Windows ICD](#).
8. Export the provisioning package. For example, click **Export > Provisioning Package > Next > (Uncheck the Encrypt Package box) > Next > Build**. (To learn more, see [Export a provisioning package](#).)

#### **IMPORTANT**

When you make any changes to a provisioning package, Windows Configuration Designer increments the version number in the provisioning file (customizations.xml). Starting with **Windows 10 release 1709**, the version number for provisioning package is also a four part number, same as the regular packaging version. In previous releases (prior to release 1709), the version number is not major.minor, it is a number with a decimal point. For example, 1.19 is a lower version than 1.2.

#### **NOTE**

As an alternative, you can update customizations.xml in the `C:\MyWorkspace\Source-<arch>\Products\ProductB\prov` with any desired changes, including the above wifi settings. In such case, you will not need a new package. Make sure you increment the version number if you make changes. See [Windows provisioning answer file](#) for more information.

9. Open **IoTCorePShell.cmd** file from your workspace and execute the following powershell commands.

```
Add-IoTProvisioningPackage Prov.WiFiSettings "C:\IoT\Provisioning\WiFiSettings\WiFiSettings.ppkg"
(or) newprovpkg Prov.WiFiSettings "C:\IoT\Provisioning\WiFiSettings\WiFiSettings.ppkg"
```

This creates a new folder at `C:\MyWorkspace\Common\Packages\Prov.WifiSettings`.

This also adds a FeatureID **PROV\_WIFISETTINGS** to the `C:\MyWorkspace\Common\Packages\OEMCOMMONFM.xml` file.

10. Build a cab file for provisioning.

```
New-IoTCabPackage Prov.WifiSettings
(or) buildpkg Prov.WifiSettings
```

## Update the project's configuration files

1. Update the product test configuration file using [Add-IoTProductFeature](#)

```
Add-IoTProductFeature ProductB Test PROV_WIFISETTINGS -OEM
(or) addfid ProductB Test PROV_WIFISETTINGS -OEM
```

# Build and Test Image

Build the FFU image again, as specified in [Create a basic image](#). You should only have to run the `New-IoTFFUImage` command:

```
New-IoTFFUImage ProductX Test  
(or)buildimage ProductX Test
```

Once the FFU file has been built and you flash it to your hardware device as specified in [Flash a Windows IoT Core Image](#), your provisioning package customizations should be applied when you power up the device. In our example, the default app is the [IoT Core Default app](#), which will show basic info about the image when the device is booted up.

## Test to see if your provisioning settings were applied

1. Unplug any network cables from your IoT device.
2. Select the defaults. At the **Let's get connected** screen, select **Skip this step**.
3. If your wireless network is in range, this screen should show the network successfully connected, and show an IP address for the network.

## Test network connections and upload apps

You can connect to your device's portal page to troubleshoot network connections, upload apps, or see more details about your device.

1. Connect both your technician PC and the device to the same network.  
For example, to connect over a wired network, plug in a ethernet cable. To connect over wireless, make sure both your technician computer and IoT Core device are connected to the same wireless network.
2. On your technician PC, open Internet Explorer, and type in the device's IP address with an http:// prefix and :8080 suffix.

```
http://10.123.45.67:8080
```

3. When prompted, enter your device's default username and password. (Default is: Administrator \ p@ssw0rd)

This opens the [Windows Device Portal](#). From here, you can upload app packages, see what apps are installed, and switch between them.

4. Click **Networking > Profiles**. You should see the Wi-Fi profile you created.

If the device is able to automatically connect to the WiFi network, then under **Available Networks**, you should see a checkmark next to the network you configured.

If your network requires steps such as accepting license terms, the device may not auto-connect.

## Troubleshooting

**Check your Wi-Fi broadcast frequency (2.4GHz vs 5GHz).** Some Wi-Fi adapters, such as the built-in Wi-Fi adapter on the Raspberry Pi 3, only support 2.4GHz Wi-Fi networks. While this is the most common Wi-Fi broadcast frequency, many Wi-Fi networks broadcast at frequencies of 5GHz. Either change the broadcast frequency or use a different adapter.

**Confirm that the provisioning package settings work on your network.** Use a laptop PC to test:

1. Disconnect the laptop from the network: Click on the network icon in the system tray, select the wireless network, and click **Disconnect**.
2. Confirm that the network is no longer connected.
3. Install the provisioning package by double-clicking ProductAProv.pkg. The wireless network should connect automatically.

#### **Check to see if the profile has been added to the device**

1. Connect using an ethernet connection to the device.
2. Connect using an SSH client, such as [PuTTY](#).
3. When connected, check to see what profiles have been installed:

```
netsh wlan show profiles
```

The network should appear in the list of User profiles.

**Use a different device name for each device.** This can help prevent network conflicts. Set this name while creating media for the device.

## Next steps

[Lab 1e: Add a driver to an image](#)

# Lab 1e: Add a driver to an image

5/9/2019 • 2 minutes to read

In this lab, we'll add the sample driver - [Toaster](#) - package it up, and deploy it to our device.

## Prerequisites/Requirements

Make sure you've created a basic image from [Create a basic image](#).

You will need the following tools installed to complete this section:

- Windows Assessment and Deployment Kit (Windows ADK)
- IoT Core PowerShell Environment
- Windows 10 IoT Core Packages
- IoT Core ADK Add-Ons
- A text editor like Notepad or VS Code

## Check for similar drivers

Before adding drivers, you may want to review your pre-built Board Support Package (BSP) to make sure there's not already a similar driver.

For example, review the list of drivers in the file: `\IoT-ADK-AddonKit\Source-arm\BSP\Rpi2\Packages\RPi2FM.xml`

- If there's not an existing driver, you can usually just add one.
- If there is a driver, but it doesn't meet your needs, you'll need to replace the driver by creating a new BSP. We'll cover that in [Lab 2](#).

## Create your driver files

- Complete the steps listed under the [Toaster Driver sample](#) to build this sample. You'll create a file, `wdfsimple.sys`, which you'll use to install the driver.

You can also use your own IoT Core driver, as long as it doesn't conflict with the existing Board Support Package (BSP).

- Copy the files, `wdfsimple.sys` and `wdfsimple.inf`, into a test folder, for example: `C:\wdfsimple\`

## Build a package for your driver

Once the driver files are created, we need to create a package that includes them, and then add that package to our Windows IoT Core image.

1. Run **IoT Core PowerShell Environment** as an administrator. Select your appropriate architecture.
2. Create a **driver package** using [New-IoTDriverPackage](#).

```
Add-IoTDriverPackage C:\wdfsimple\wdfsimple.inf Drivers.Toaster  
(or) newdrvpkg C:\wdfsimple\wdfsimple.inf Drivers.Toaster
```

This creates a new folder at `C:\MyWorkspace\Source-<arch>\Packages\Drivers.Toaster`.

This also adds a FeatureID **DRIVERS\_TOASTER** to the `C:\MyWorkspace\Source-<arch>\Packages\OEMFM.xml` file.

### 3. Build the package using [New-IoTCabPackage](#).

```
New-IoTCabPackage Drivers.Toaster  
(or) buildpkg Drivers.Toaster
```

## Update the project's configuration files

Update the product test configuration file using [Add-IoTProductFeature](#).

```
``` powershell  
Add-IoTProductFeature ProductB Test DRIVERS_TOASTER -OEM  
(or) addfid ProductB Test DRIVERS_TOASTER -OEM  
```
```

## Build and test the image

Build the FFU image again, as specified in [Create a basic image](#). You should only have to run the [New-IoTFFUImage](#) command:

```
New-IoTFFUImage ProductX Test  
(or) buildimage ProductX Test
```

## Verify driver is installed properly

You can verify that the test driver was installed properly by following the steps in the [Toaster Driver sample](#) to test your driver.

Otherwise, if you used another test driver, you can follow these steps:

1. Boot up your Windows 10 IoT Core device and make note of its IP address.
2. On your technician PC, open **File Explorer** and in the address bar type in `\\"<TARGET_DEVICE_IP>\c$` and press **Enter**. **TARGET\_DEVICE\_IP** will correspond to the IP address of your device.

If you are prompted for credentials, please enter these and click OK. If you have not changed the default credentials use the following:

```
User ID: Administrator  
Password: p@ssw0rd
```

3. Once your credentials are accepted and **File Explorer** displays the c\$ directory of your device, navigate to `c:\Windows\System32\Drivers` and look for the **gpiokmdfdemo.sys** file. If present, this validates that your driver has been properly installed on your device.

## Next steps

[Lab 1f: Add Win32 services to an image](#)

# Lab 1f: Add Win32 services to an image

5/9/2019 • 2 minutes to read

Windows 10 IoT Core supports adding a Win32 NT Service to your image.

## Prerequisites/Requirements

Make sure you've created a basic image from [Create a basic image](#).

You will need the following tools installed to complete this section:

- Windows Assessment and Deployment Kit (Windows ADK)

### NOTE

The version of ADK used must match the version of IoT Core Packages used below.

- Windows 10 IoT Core Packages
- IoT Core PowerShell Environment
- IoT Core ADK Add-Ons
- A text editor like Notepad or VS Code

## Add a Win32 Service app to package build

In order to include your Win32 Service App in the FFU image build process, you first must add the .EXE file so that it can be packaged up (using `buildpkg`).

1. Create a subdirectory for your Win32 Service App under

`C:\IoT\Workspaces\ContosoWS\Source-<arch>\Packages`. This will contain the XML and EXE files to include when building the image. For example, refer to the **AzureDM.Services** subdirectory at `C:\IoT\Workspaces\ContosoWS\Source-<arch>\Packages\AzureDM.Services` for a working example.

2. Create an XML file titled `<your Win32 Service App Name>.wm.xml` in the subdirectory you created from Step #1. This file will specify how the package will be built. Here is an example of what that file should look like (you would replace the appropriate entries with your Win32 Service App information):

```

<?xml version="1.0" encoding="utf-8"?>
<identity xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    name="Services"
    namespace="AzureDM"
    owner="${OEMNAME}"
    legacyName="${OEMNAME}.<your Win32 Service App Name>.Services"
    xmlns="urn:Microsoft.CompPlat/ManifestSchema.v1.00">
    <onecorePackageInfo
        targetPartition="MainOS"
        releaseType="Production"
        ownerType="OEM" />
    <files>
        <file
            destinationDir="$(runtime.system32)"
            source="<your Win32 Service App Name executable filename>" />
    </files>
    <service
        name="<your Win32 Service App Name>"
        start="auto"
        type="win32OwnProcess"
        objectName="LocalSystem"
        errorControl="normal"
        displayName="<your Win32 Service App Display Name>"
        description="<your Win32 Service App Description>"
        imagePath="<path and file name of your Win32 Service App>">
        <failureActions
            resetPeriod="86400">
            <actions>
                <action
                    type="restartService"
                    delay="1000" />
                <action
                    type="restartService"
                    delay="1000" />
                <action
                    type="restartService"
                    delay="1000" />
                <action
                    type="none"
                    delay="0" />
            </actions>
        </failureActions>
    </service>
</identity>

```

#### NOTE

The `<service>` area in the XML file specifies Win32 Service-specific information. If you are adding a Win32 application (like a console app), this section can be omitted.

3. Add your EXE file to the subdirectory from Step #1. This is your Win32 Service application executable.

## Package the Win32 Service App

The next step is to package the Win32 Service App file, which will allow you to build it using the Windows ADK (when you build the FFU image).

1. Open `IoTCorePShell.cmd` from your workspace. It should prompt you to run as an administrator.
2. Build the package into a .CAB file (using [New-IoTCabPackage](#))

```
New-IoTCabPackage <your Win32 Service App Name>
(or) buildpkg <your Win32 Service App Name>
```

This will build the package into a .CAB file under the `\\Build\\<arch>\\pkgs` subdirectory in your workspace.

## Update Project Configuration Files

You can now update your product configuration files to include your app in the FFU image build.

1. Add the Feature ID for your app package using [Add-IoTProductFeature](#), replacing `<your Win32 service app name>` with an identifier for your Win32 service app:

```
Add-IoTProductFeature <product name> Test <your Win32 service app name> -OEM
or addfid <product name> Test <your Win32 service app name> -OEM
```

This adds a FeatureID corresponding to the identifier you chose for your Win32 service app.

## Build and Test Image

Build the FFU image again, as specified in [Create a Basic IoT Core Image](#). You should only have to run the [New-IoTFFUImage](#) command:

```
New-IoTFFUImage ProductX Test
(or)buildimage ProductX Test
```

Once the FFU file has been built (it should now include your app), you can flash it to your hardware device as specified in [Flashing a Windows IoT Core Image](#).

## Next Steps

[Lab 1g: Build a retail image](#)

# Lab 1g: Build a retail image

6/24/2019 • 8 minutes to read

We will detail the steps needed to create a retail Windows IoT Core image and flash it onto a specific hardware device.

## Prerequisites/Requirements

Make sure you've created a basic image from [Create a basic image](#), as well as completed all of the previous labs.

You will need the following tools installed to complete this section:

- A retail code-signing certificate
- A cross-signing certificate
- Visual Studio
- Windows Assessment and Deployment Kit (Windows ADK)
- IoT Core PowerShell Environment
- A text editor like Notepad or VS Code

## Modify project configuration files

Follow the steps below to add any custom applications or provisioning packages you want to add to the retail image.

1. To add a custom application, you should follow the instructions listed in [Add an app to an image](#). However, you would specify `Retail` instead of `Test` when executing the `Add-IoTProductFeature` command, as shown here:

```
Add-IoTProductFeature ProductX Retail APPX_HELLOWORLDAPP -OEM  
or addfdid ProductX Retail APPX_HELLOWORLDAPP -OEM
```

This adds a FeatureID called **APPX\_HELLOWORLDAPP** to the specified product's Retail OEMInput XML file.

2. Minimize the included Windows IoT Core features. You also want to remove any test applications that are included (by default) with test images for Windows IoT Core. This includes the IoT Core default application along with any other developer tools or testing features. You can do this by using `Remove-IoTProductFeature`:

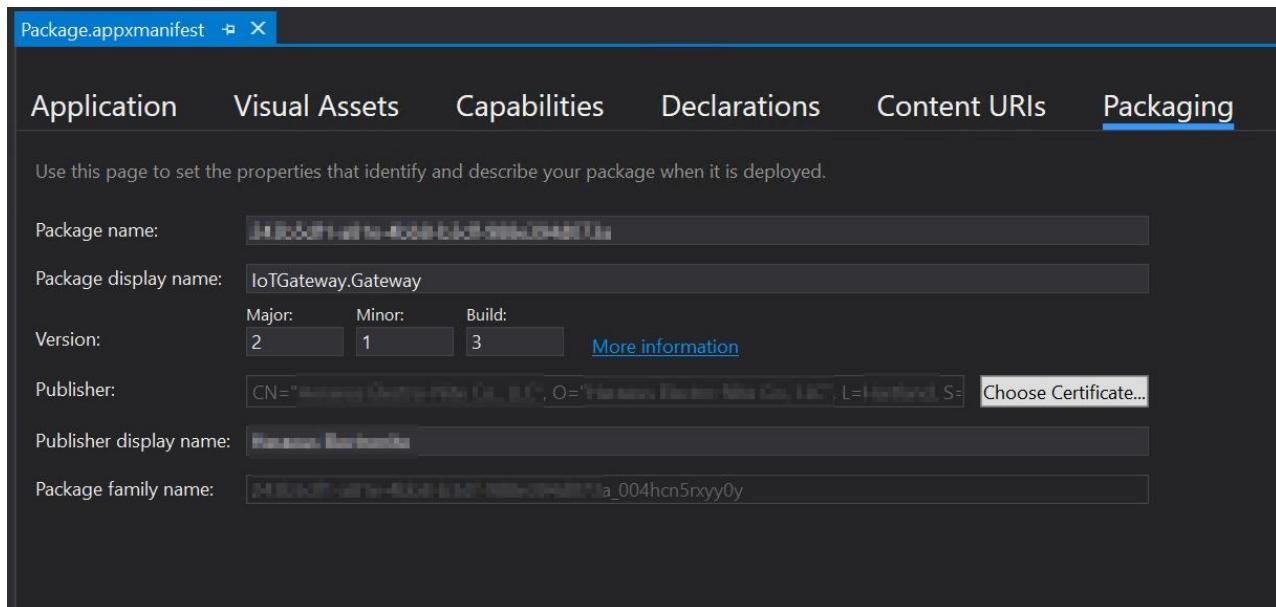
```
Remove-IoTProductFeature ProductX Test IOT_BERTHA  
or removefdid ProductX Test IOT_BERTHA
```

## Properly signing and including your applications

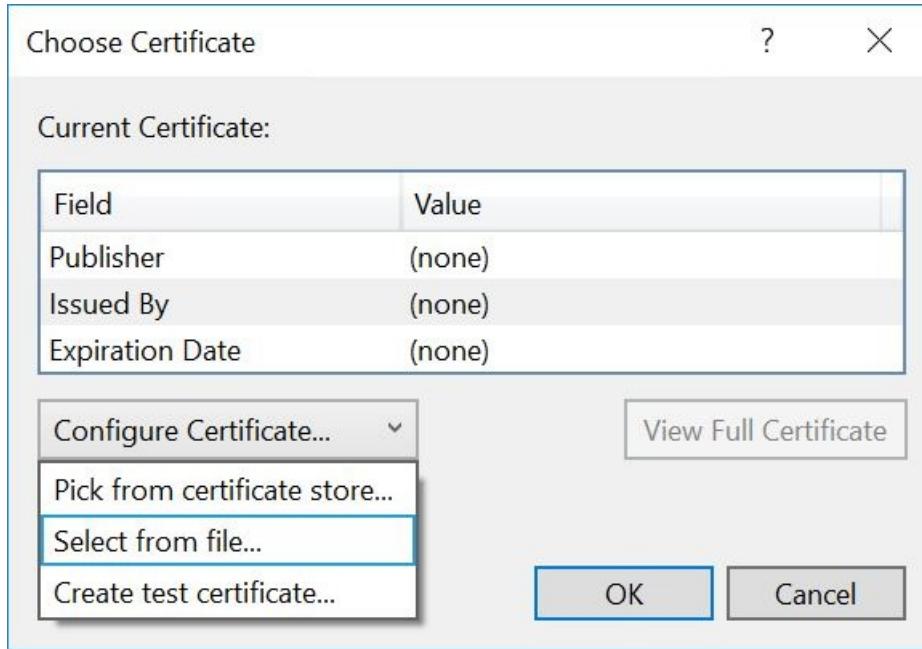
If you have one or more custom applications that you want to include in your Windows IoT Core retail image, you need to verify that these applications are signed properly when including them in your retail image. Follow these steps for each application you want to include in your image. Please note that you can skip Steps 8 and 9 if you only have one application to include.

1. Install your retail code-signing certificate on your technician PC.
2. Open your custom application in **Visual Studio** and open the **Package.appxmanifest** file.

3. Click on the **Packaging** tab and click on the **Choose Certificate...** button.



4. The dialog displayed will show what certificate is being used for code-signing. Click on the **Configure Certificate...** dropdown and select **Pick from certificate store...**:



5. Choose your retail code-signing certificate when prompted and click **OK**.
6. Save your project in **Visual Studio** and then build your Appx package. Please note that you should be prompted for your password for your retail code-signing certificate when building this package.
7. Once the Appx file is built, run the following command in **IoT Core Powershell Environment**:

```
Add-IoTAppxPackage
"C:\Dev\OpenSource\ContosoApp\ContosoApp\AppPackages\ContosoApp_1.0.0.0_ARM_Debug_Test\ContosoApp_1.0.0.0_ARM_Debug.appx" fga Appx.ContosoApp
(or) newAppxPkg
"C:\Dev\OpenSource\ContosoApp\ContosoApp\AppPackages\ContosoApp_1.0.0.0_ARM_Debug_Test\ContosoApp_1.0.0.0_ARM_Debug.appx" fga Appx.ContosoApp
```

**Build the retail image files**

Once we have all the custom application packages signed properly, we can now build the Windows IoT Core retail image. Please verify that you have the retail code-signing certificate installed on your PC prior to following these steps:

1. Set the IoT signature to include details about your certificate and cross-certificate. This is done by modifying the `IoTWorkspace.xml` file, located in your workspace (e.g. C:\MyWorkspace):

```
<!--Specify the retail signing certificate details, Format given below -->
<RetailSignToolParam>/s my /sha1 "thumbprint" /fd SHA256 /ac "c:\DownloadedCrossCert.crt"
</RetailSignToolParam>
```

2. Run **IoT Core PowerShell Environment** as an administrator.
3. Set the environment for retail signing. this is done with [Set-IoTRetailSign](#):

```
Set-IoTRetailSign On
(or) retailsign on
```

4. Build the packages:

```
New-IoTCabPackage All
(or) buildpkg all
```

Once all the package .CAB files are built, you should verify that each of these files is properly signed with the retail certificate. If some are still signed with the test certificates (this usually happens if you use your technician PC for building both test and retail images), you can re-sign these files using the [Redo-IoTCabSignature](#):

```
Redo-IoTCabSignature C:\BSP.IN C:\BSP.OUT
(or) re-sign.cmd C:\BSP.IN C:\BSP.OUT
```

This takes the .CAB files from `C:\BSP.IN`, re-signs them with the retail certificate and copies them to the `C:\BSP.OUT` directory.

6. If you re-signed the .CAB files from Step 5, copy the re-signed .CAB files to the `C:\IoT\Workspaces\ContosoWS\Build\<arch>\pkgs`, overwriting the existing files. In our example, these files are copied to `C:\IoT\Workspaces\ContosoWS\Build\arm\pkgs`.
7. Build your retail image by running the following command:

```
New-IoTFFUIImage ProductX Retail
(or) buildimage ProductX Retail
```

8. You can then flash the retail image as described in [Flash an image](#).

## Commands used

Listed here are the commands (in order) for creating a retail IoT Core image. Please note that your retail code-signing certificate should be installed first, and it may prompt you for the certificate password when re-signing the .CAB files.

```
Set-IoTRetailSign On
New-IoTCabPackage All
Redo-IoTCabSignature C:\BSP.IN C:\BSP.OUT
xcopy C:\BSP.OUT\*.cab C:\IoT\Workspaces\ContosoWS\Build\arm\pkgs\*.cab
New-IoTFFUIImage ProductX Retail
```

## Add your features to the retail configuration

1. Update the product retail configuration file using [Add-IoTProductFeature](#)

```
# Add application features
Add-IoTProductFeature ProductA Test APPX_MYUWPAPP -OEM
Remove-IoTProductFeature ProductA Test IOT_BERTHA
# Add registry and file features
Add-IoTProductFeature ProductA Retail FILES_CONFIGS -OEM
Add-IoTProductFeature ProductA Retail REGISTRY_SETTINGS -OEM
# Add provisioning feature
Add-IoTProductFeature ProductA Retail PROV_WIFISETTINGS -OEM
# Add driver
Add-IoTProductFeature ProductA Retail DRIVERS_HELLOBLINKY -OEM
```

## Verify retail image

A user can easily verify a custom test image for Windows IoT Core just by turning on a device with the image flashed on the device. Once the device is running, you can run through various checks to verify that the device is truly functional. The ease of these tests depends on the levels of security factors baked into the image. Since there are no security protocols built-in on a test image, you can use all available development tools to test the IoT device.

The task of testing becomes harder for a custom retail image for Windows IoT Core, as you can include security protocols as part of the image installed into the device. Due to the nature of these security protocols, you may not be able to use available test tools to verify the device, so you may want to write a test application that can run on the IoT device. This application would then perform the verification tests of the various areas and functions of the IoT device.

Testing of a custom retail image for Windows IoT Core can be done by one of the following ways:

### Clean retail image

If you truly want to have a clean retail image, you will need to create two retail images for your device. These two images will be identical except that one image will include your testing application (configured as a foreground application), while the other "clean" image will not. You would flash the first image (with the test application included) and perform the testing validations on your IoT device. Once verified, you can then re-flash your IoT device with the second "clean" retail image, for distribution.

**Pros:** Final retail image will be totally clean and only the items that are deemed necessary are included in the image.

**Cons:** Including a test application to the retail image could introduce some potential issues with the provisioning package(s), as well as potential user errors in the test application. This would make this retail image different than the final retail image.

### One-Time Passthrough Test

Only one final retail image is created, and would also include the test application. You would configure the image such that once the out-of-box-experience (OOBE) application is launched, your test application is launched (as a foreground application). A conditional statement within the test application would be triggered so that the application is aware that it ran once before (preventing it from running after the first time the device is powered

on).

```
// Declare variable
Windows.Storage.ApplicationDataContainer localSettings =
    Windows.Storage.ApplicationData.Current.LocalSettings;

// Set variable as boolean, numbers, or string values as needed at appropriate location within the test app
localSettings.Values["appRanOnce"] = false;

// Read variable and verify value to check and apply logic
Object value = localSettings.Values["appRanOnce"];
```

#### NOTE

For best results, only use `localSettings` to store the variables to store the settings value. There is a possible chance of undesirable results from using `roamingSettings` features. `localSettings` can only hold 64k of data at the time of this writing. Learn more about Application Settings [here](#).

Using the code block above, you can apply the logic on launch of your test application so that on subsequent launches, the application takes appropriate actions.

So what types of actions can I take?

- Launch another FGA app
- Edit the Registry to modify the boot sequence

#### Launching another FGA application from your test application

If you are launching a Microsoft store app, you can use the following code snippet to launch apps installed and updated through the store. Additional information on URI schemes can be found [here](#).

```
// Following will launch the Microsoft store app and navigate to the Games section
bool result = await Windows.System.Launcher.LaunchUriAsync(new Uri
    ("ms-windows-store://navigatetopage/?Id=Games"));

// Following will launch the One Note app using the package family name (PFN)
bool result = await Windows.System.Launcher.LaunchUriAsync(new Uri
    ("ms-windows-store://pdp/?PFN= Microsoft.Office.OneNote_8wekyb3d8bbwe"));
```

If you are launching a custom (non-Microsoft store) app, you can use `AppServiceConnection` to launch an app using package family name (PFN).

First, you must register the final app (`com.concurrency.lwinsapp`) with app services within the system. You will need to modify the `Package.appxmanifest` file to include the following code block in the `<Applications>` section of the manifest.

```
<Application Id="App" Executable="$targetnametoken$.exe" EntryPoint="AppServiceProvider.App">
    <Extensions>
        <uap:Extension Category="windows.appService" EntryPoint="MyAppService.AppLaunchService">
            <uap3:AppService Name="com.concurrency.lwinsapp" uap4:SupportsMultipleInstances="true" />
        </uap:Extension>
    </Extensions>
    ...
</Application>
```

Following code segment will launch a custom application:

```
private AppServiceConnection appLaunchService;  
...  
this.appLaunchService = new AppServiceConnection();  
this.appLaunchService.AppServiceName = "com.concurrency.lwinsapp";  
this.appLaunchService.PackageFamilyName = "f3a114f7-e099-4773-8c93-77abcba14f62_004hcnsrxyy0y";  
var status = await this.appLaunchService.OpenAsync();
```

By combining logic between `localSettings` and `AppServiceConnection`, you can bypass your test application on every boot of the device. In essence, your test application will run on every boot but "passthrough" to the final application on boot. If needed, you can set your logic in such a way that device will not continue to the final application if tests fails on your test application. This might be helpful if you need to verify that the device is fully tested and functional on every boot.

**Pros:** You can test the device automatically on every boot to ensure that certain conditions are set correctly and the device is fully tested (and secure).

**Cons:** Your test application is included with the retail image. There is the potential of your application having security holes. Please make sure that your test app is locked down as needed. Due to the nature of your test application, you may be able to modify features of the device.

## Next steps

- [Lab 2: Creating your own board support package](#)

# Lab 2: Creating your own board support package (BSP)

6/24/2019 • 2 minutes to read

A BSP includes a set of device drivers that are specific to the components/silicon used in the board. These are provided by the component vendors / silicon vendors, mostly in the form of .inf and associated .sys/.dll files.

Create a new Board Support Package (BSP) when:

- Creating a new hardware design
- Replacing a driver or component on an existing hardware design

Whether you're creating a new BSP or modifying an existing BSP, you become the owner. This lets you decide whether to allow updates to install on your boards.

In our lab, we'll create a new BSP based on the Raspberry Pi 2, removing the existing GPIO driver and replacing it with the sample GPIO driver: [GPIO KMDF demo](#).

## Create a new BSP working folder

1. From the IoT Core Shell, create a BSP working folder that you'd like to modify using [Add-IoTBSP](#).

```
Add-IoTBSP MyRPi2  
(or) newbsp MyRPi2
```

## Add packages into the feature manifest

1. Open the feature manifest file for your new BSP, `C:\MyWorkspace\Source-arm\BSP\MyRPi2\MyRPi2FM.xml`.

In another window, open the Raspberry Pi 2 feature manifest to use as a template.

2. Add your base packages (BasePackages).

- UEFI drivers for the boot partition (RASPBERRYPI.RPi2.BootFirmware.cab)
- Drivers required for UpdateOS (SV.PlatExtensions.UpdateOS.cab)
- Mandatory device drivers (bcm2836sdhc.cab, dwcUsbOtg.cab, rpiq.cab)

When creating your own BSP, it's typical to require a display driver and a storage driver, and sometimes a network driver.

- Device-specific customizations

3. Copy in the device layout and platform packages (DeviceLayoutPackages, OEMDevicePlatformPackages).

Note that both the OEMDevicePlatform.xml and devicelayout.xml can be packaged into one package, for example, DeviceLayout.MBR4GB. The same package can then be specified as input in both the sections (for example, under and ). To learn more, see [Device layout](#).

4. Copy in features (Features).

Copy in features you want. Exclude any that don't apply to your project.

For example, copy in each of the drivers **except** the existing GPIO driver:

```
<PackageFile Path="$(mspackageroot)\Retail\$(cpuytype)\$(buildtype)" Name="RASPBERRYPI.RPi2.GPIO.cab">
<FeatureIDs>
<FeatureID>RPi2_DRIVERS</FeatureID>
</FeatureIDs>
</PackageFile>
```

Note: To make grouping packages easier, you can combine them into one or more Feature IDs. For example, all of the Raspberry Pi 2 optional drivers use the Feature ID: RPi2\_DRIVERS.

## 5. Add the HelloBlinky driver

```
<PackageFile Path="%PKGBLD_DIR%" Name="%OEM_NAME%.Drivers.HelloBlinky.cab">
<FeatureIDs>
<FeatureID>BLINKY_DRIVER</FeatureID>
</FeatureIDs>
</PackageFile>
```

## Create a new product folder

### 1. Create a new working product folder, adding your BSP name to the end.

```
newproduct ProductC MyRpi2
```

This creates the folder: `C:\MyWorkspace\Source-<arch>\Products\ProductC`, which is linked to the new BSP.

## Update the project's configuration files

### 1. Update the product test configuration file using [Add-IoTProductFeature](#)

```
Add-IoTProductFeature ProductC Test BLINKY_DRIVER -OEM
(or) addfid ProductC Test BLINKY_DRIVER -OEM
```

## Build and test the image

### Build the image

#### 1. From the IoT Core Shell, create the image:

```
buildimage ProductC Test
```

This creates the product binaries at `C:\MyWorkspace\Build\<arch>\ProductC\Flash.ffd`.

#### 2. Start **Windows IoT Core Dashboard > Setup a new device > Custom**, and browse to your image.

Put the Micro SD card in the device, select it, accept the license terms, and click **Install**. This replaces the previous image with our new image.

#### 3. Put the card into the IoT device and start it up.

After a short while, the device should start automatically, and you should see your app.

### Check to see if your driver works

#### 1. Use the [testing procedures in the Hello, Blinky! lab](#) to test your driver.

## Related topics

[Device layout](#)

## Next steps

[Lab 3: Update apps](#)

# IoT Core Board Supported Packages (BSP)

8/5/2019 • 2 minutes to read

Board Support Packages (BSP) is a collection of drivers/settings required to run IoT Core on a hardware platform. These are provided by the hardware vendors/silicon vendors. The BSP also includes a set of device drivers that are specific to the components/silicon used in the board, mostly in the form of .inf files and their associated .sys/.dll files.

Listed below are the steps required to extract the BSP files for specific manufacturers. You will need these files extracted properly before you can build a FFU image file. Then, you'll learn how to create your own BSP by following the steps listed in [Lab 2](#).

## Raspberry Pi BSP

1. Download [RPi\\_BSP.zip](#) to a local directory such as `C:\Downloads\RPi_BSP.zip`
2. Launch [IoTCorePShell](#) and create or open a workspace using

```
new-ws C:\MyWorkspace <oemname> arm  
(or) open-ws C:\MyWorkspace
```

3. Import the bsp using [Import-IoTBSP](#) and build using

```
Import-IoTBSP RPi2 C:\Downloads\RPi_BSP.zip  
(or) importbsp RPi2 C:\Downloads\RPi_BSP.zip  
buildpkg RPi2
```

### NOTE

Raspberry Pi BSP driver sources are available at [ms-iot/rpi-iotcore](#)

## Intel BSPs

### BSP Links

CHIPSET	DOWNLOAD LINK
Intel® Atom™ Processor E3800 Product Family and Intel® Celeron® Processor N2807/N2930/J1900	<a href="#">Download</a> Intel® Embedded Drivers for Microsoft Windows® 10 IoT Core (32-bit and 64-bit) MR1
Intel Atom® Processor E3900 Series, and Intel® Pentium® and Celeron® Processor N- and J-Series (Apollo Lake)	<a href="#">Download</a> Software Package: Intel Atom® E3900 SoC Family—Board Support Package (BSP) for Windows® 10 IoT Core 32-bit and 64-bit Platforms
Intel® Pentium® and Celeron® Processor N3000 Product Families, and Intel® Atom™ x5-E8000 Processor	<a href="#">Download</a> Board Support Package for Intel Atom® Processor Windows® 10 IoT Core 32-bit and 64-bit Platforms
Intel® Atom™ x5-E8000 Processor and Intel® Atom™ x5-Z8350 Processor	Contact your Intel representative

## Instructions to use

Follow the steps below to use this BSP with the Windows 10 ADK release 1809 (17763) with iot-adk-addonkit version 6.0.

1. Download the BSP package and install
2. Launch IoTCorePShell and create/open your workspace

```
new-ws C:\MyWorkspace <oemname> arm  
(or) open-ws C:\MyWorkspace
```

3. Set the source location, either the installed directory or the zip file path

```
$Source = "C:\Program Files (x86)\Intel IoT\Source-<arch>"  
(or)  
$Source = "C:\Downloads\IntelBSP.zip"
```

4. Import the bsp using [Import-IoTBSP](#) and build using

```
Import-IoTBSP <bspname> $Source  
(or) importbsp <bspname> $Source  
buildpkg <bspname>
```

## Qualcomm BSPs

### DragonBoard 410C

DragonBoard drivers are available at [DragonBoard 410C Software](#) under the *Windows 10 IoT Core* section.

Steps to import the drivers :

1. Download the [Windows 10 IoT Core Board Support Package](#) to a folder such as `C:\Downloads\*_db410c_BSP.zip`
2. Launch IoTCorePShell, and create/open your workspace

```
new-ws C:\MyWorkspace <oemname> arm  
(or) open-ws C:\MyWorkspace
```

3. Import the bsp using [Import-QCBSP](#) and build using

```
Import-QCBSP "C:\Downloads\*_db410c_BSP.zip" C:\prebuilt\DB410c_BSP -ImportBSP  
buildpkg QCBDB410C
```

Set `<BSPPkgDir>` setting in the `IoTWorkspace.xml` to `C:\prebuilt\DB410c_BSP`

## NXP BSPs

See [Window 10 IoT Core and NXP i.MX SoCs](#) for information on the NXP BSP access and Ecosystem resources.

## Other helpful resources

- [Windows ADK IoT Core Add-Ons Overview](#)
- [IoT Core Add-Ons Powershell Commands](#)

- [IoT Core feature list](#)
- [Channel9 Video on Manufacturing Guide](#)

# IoT Device Layout

1/24/2019 • 2 minutes to read

When modifying an IoT Core board support package (BSP), you can change the drive partitions and layout by modifying the `DeviceLayout` files.

## Partition layout

IoT Core supports UEFI (GPT) and legacy BIOS (MBR) partition layouts. Most IoT Core devices use UEFI and GPT-style partitions, though Raspberry Pi 2 uses MBR-style partitions. To learn more about UEFI, read [Boot and UEFI](#) and the [Windows and GPT FAQ](#).

Sample partition layouts included in the ADK Add-Ons:

- \iot-adk-addonkit\Common\Packages\DeviceLayout.GPT4GB\devicelayout.xml
- \iot-adk-addonkit\Common\Packages\DeviceLayout.GPT4GB-R\devicelayout.xml
- \iot-adk-addonkit\Common\Packages\DeviceLayout.MBR4GB\devicelayout.xml
- \iot-adk-addonkit\Common\Packages\DeviceLayout.MBR4GB-R\devicelayout.xml

These files use three component files:

- **DeviceLayout..pkg.xml**: Package file, creates packages for `DeviceLayout` and `OEMDevicePlatform.xml`.
- **DeviceLayout.xml**: Specifies the device partition layout
- **OEMDevicePlatform.xml**: Specifies the amount of free blocks available in the device and which partitions are compressed.

### Partition layout (`DeviceLayout.xml`)

IoT Core requires 3 mandatory partitions (EFIESP, MainOS and Data). You can optionally include other partitions, for example, a CrashDump partition. Sizes are calculated in sectors, the default sector is 512 bytes.

Supported properties:

**EFI**: Fixed-size partition with the boot manager, boot configuration database. This partition is required for both MBR/GPT-style devices.

- Name: `EFIESP`
- Type: For MBR, use `0x0c`. For GPT, use `{c12a7328-f81f-11d2-ba4b-00a0c93ec93b}`
- FileSystem: `FAT`
- TotalSectors: `65536` (= 32MB)
- Bootable: `true`
- RequiredToFlash: `true`

**MainOS**: OS and OEM-preloaded apps. This partition requires a minimum number of free sectors (MinFreeSectors) for normal operations.

- Name: `MainOS`
- Type: For MBR, use `0x07`. For GPT, use `{ebd0a0a2-b9e5-4433-87c0-68b6b72699c7}`
- FileSystem: `NTFS`

- MinFreeSectors: `1048576` (= 512MB)
- ByteAlignment: `0x800000`
- ClusterSize: `0x1000` (This size is recommended to keep the partition size manageable.)

**Data:** User data partition, user registry hives, apps, apps data. This partition is typically set to use the remainder of the storage space on the device. (UseAllSpace: True)

- Name: `Data`
- Type: For MBR, use `0x07`. For GPT, use `{ebd0a0a2-b9e5-4433-87c0-68b6b72699c7}`
- FileSystem: `NTFS`
- UseAllSpace: `true`
- ByteAlignment: `0x800000`
- ClusterSize: `0x4000` (This partition tends to be larger, so 0x4000 is recommended. 0x1000 is also OK.)

**Crash dump partition:** Optional partition, used to collect data from crash dumps. When used, size is given in total sectors.

- Name: `CrashDump`
- Type: For MBR, use `0x07`. For GPT, use `{ebd0a0a2-b9e5-4433-87c0-68b6b72699c7}`
- FileSystem: `FAT32`
- TotalSectors: `1228800` (= 600 MB)

## Required fields

These fields are required, the following values are supported for IoTCore:

- Version: `IoTUAP`
- SectorSize: `512`
- ChunkSize: `128`
- DefaultPartitionByteAlignment: `0x200000`

## Storage Size Estimations

The following diagrams provide an overview of two configurations.

### 2GB Configuration (2048MB, typically has 1843MB for storage)

#### 2GB partition layout



PARTITION	CONTENTS	MB	SECTORS	REMARKS
EFIESP	EFIESP	32	65536	EFIESP size
Main OS	Main OS	800	1638400	MainOS (estimate)
Main OS	Free space	128	262144	MainOS Headroom

PARTITION	CONTENTS	MB	SECTORS	REMARKS
Data	Data	883	1808384	Expands to fill free space
<b>TOTAL</b>		<b>1843</b>	<b>3774464</b>	

**4GB Configuration:** (4096MB, typically has 3600MB available for storage)

#### 4GB partition layout



PARTITION	CONTENTS	MB	SECTORS	REMARKS
EFIESP	EFIESP	32	65536	EFIESP size
Main OS	Main OS	800	1638400	MainOS (estimate)
Main OS	Free space	512	1048576	MainOS Headroom
CrashDump	Crash Dump	600	1228800	CrashDump Size
Data	Data	1656	3391488	Expands to fill free space
<b>TOTAL</b>		<b>3600</b>	<b>7372800</b>	

#### Device platform layout (OEMDevicePlatform.xml)

OEMDevicePlatform.xml specifies the amount of free blocks available in the device and which partitions are compressed. Example:

```

<?xml version="1.0" encoding="utf-8"?>
<OEMDevicePlatform xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/embedded/2004/10/ImageUpdate">
    <MinSectorCount>7372800</MinSectorCount>
    <DevicePlatformIDs>
        <ID>*</ID>
    </DevicePlatformIDs>
    <CompressedPartitions>
        <Name>MainOS</Name>
    </CompressedPartitions>
</OEMDevicePlatform>
    
```

## Related topics

[Windows 10 IoT Core BSPs](#)

[Creating your own board support package \(BSP\)](#)

[Boot and UEFI Windows and GPT FAQ.](#)

# OS Customizations for OEM

2/28/2019 • 4 minutes to read

OEM can customize various aspects of the OS using the below specified methods.

## OOBE App

IoTCore has an inbox OOBE App that runs when the device boots up for the first time. This is shown until all the provisioning packages are processed in the background and an OEM App is available to be launched as a startup app.

This OOBE app can be customised with a `settings.json` with the following attributes:

- `backgroundColor` : Screen background color
- `background` : Background image (jpg file)
- `progressRingVisible` : Spinning dots can be shown or hidden
- `welcomeText` : Text displayed in large font at the center of the screen
- `pleaseWaitText` : Text displayed below the spinning dots
- `animation` : Animation gif can be specified here
- `animationMargin` : Positioning of the animation gif
  - `left` , `top` , `right`, `bottom`

All files referenced in the `settings.json` should be in the same folder as the `settings.json` file. A sample snippet is given below

```
{  
  "backgroundColor": "#FF0000FF",  
  "progressRingVisible": true,  
  "welcomeText": "Welcome to OOBE customization",  
  "pleaseWaitText": "please wait ..."  
}
```

### NOTE

The `settings.json` file needs to be encoded in Unicode (UCS-2) encoding. UTF-8 will not work.

### Validate settings manually

1. Author the `settings.json` file with your required settings
2. Connect to the IoT device ([using SSH](#) or [using Powershell](#)) and place the `settings.json` file along with all graphical assets in a directory, say `C:\Data\oobe`
3. Configure the device to allow access to this directory from all appx files, using

```
folderpermissions C:\Data\oobe -e
```

4. Launch the OOBE application using

```
iotstartup add headed IoTUAPOOBE
```

## 5. Verify the user interface

### Add settings to IoT Core image

1. Use [Custom.OOBApp](#) package and modify the package xml file to add your graphical assets
2. Copy your settings.json and graphical assets to that package folder.
3. In the oemcustomizations.cmd file, add `folderpermissions C:\Data\oobe -e`, to ensure that this is called at the system boot.
4. In the OEMInput.xml, include the feature id **CUSTOM\_OOBAPP**, note that this is defined in the OEMCOMMONFM.xml.

## Crash Settings

For IoT Core products, it is recommended that you configure your devices to reboot on crash and also hide the crash dump screen (BSOD). This is achieved with setting the following registry keys:

```
HKLM\SYSTEM\CurrentControlSet\Control\CrashControl  
    AutoReboot set to 1  
    DisplayDisabled set to 1
```

### Validate settings manually

1. Connect to your IoT device ([using SSH](#) or [using Powershell](#)) and set the following registry keys

```
reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CrashControl" /v AutoReboot /t  
REG_DWORD /d 1 /f reg add  
"HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CrashControl" /v DisplayDisabled /t  
REG_DWORD /d 1 /f
```

2. See [Forcing a System Crash from the keyboard](#) and configure a key to force the system crash.
3. Force a system crash using the configured key and verify that the device reboots automatically and does not show the crashdump screen.

### Add settings to IoT Core image

1. Use [Custom.Settings](#) package
2. In the OEMInput.xml, include the feature id **CUSTOM\_SETTINGS**, note that this is defined in the OEMCOMMONFM.xml.

#### NOTE

In Windows 10, version 1809, **IOT\_CRASHCONTROL\_SETTINGS** feature is added to address this customization.

## BCD Settings

Boot Configuration Database settings can be used to configure various features. See [BCDEdit Command-LineOptions](#) for the various settings and options available.

A few key features are listed below

### Disable Boot UX animation

1. Manual setting can be done with the below command

```
bcdeedit -set {bootmgr} nobootuxprogress true
```

2. Specify this setting in a `Custom.BCD.xml` file

```
<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<BootConfigurationDatabase
    xmlns="http://schemas.microsoft.com/phone/2011/10/BootConfiguration"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    IncludeDescriptions="true" IncludeRegistryHeader="true">
    <Objects>
        <!-- Windows Boot Manager -->
        <Object SaveKeyToRegistry="false">
            <FriendlyName>Windows Boot Manager</FriendlyName>
            <Elements>
                <Element>
                    <DataType>
                        <WellKnownType>Boot UX Progress Animation Disable</WellKnownType>
                    </DataType>
                    <ValueType>
                        <BooleanValue>true</BooleanValue>
                    </ValueType>
                </Element>
            </Elements>
        </Object>
    </Objects>
</BootConfigurationDatabase>
```

```

3. Include this setting in the image using `Custom.BCD` package and add feature id **CUSTOM\_BCD** to `OEMInput.xml` file

### Replacing the Boot Logo

There are multiple ways to replace the boot logo that is displayed by the BIOS or UEFI. One way is to license the UEFI, or pay a board manufacturer vendor to do so, and make changes directly to the UEFI source code.

Alternatively, on devices whose UEFI implementation supports signed loadable UEFI drivers there is a sample here: [https://github.com/Microsoft/MS\\_UEFI/tree/share/MsIoTSamples](https://github.com/Microsoft/MS_UEFI/tree/share/MsIoTSamples) that shows how to build a driver that replaces the boot logo and supply a BGRT table to bootmgr so that the Windows boot process leaves your logo in place during boot instead of replacing it with the Windows logo.

### Enable Flight Signing

1. Manual setting can be done with the below commands:

```
bcdeedit /set {bootmgr} flightsigning on
bcdeedit /set flightsigning on
```

2. To include this setting in the image, you can add the below fragment to the `Custom.BCD.xml`

```
<!-- Allow Flight Signing Certificate -->
<Object SaveKeyToRegistry="false">
  <FriendlyName>Global Settings Group</FriendlyName>
  <Elements>
    <Element>
      <DataType>
        <WellKnownType>Allow Flight Signatures</WellKnownType>
      </DataType>
      <ValueType>
        <BooleanValue>true</BooleanValue>
      </ValueType>
    </Element>
  </Elements>
</Object>
```

## Runtime customizations

In addition to the static customizations discussed above, you can also customize during the runtime.

### 1. OEMCustomizations.cmd

- This command file is invoked by IoTCore Shell on every boot with system privileges, placed in `c:\windows\system32`
- You can specify any customization actions here in this cmd file, though it is recommended to keep this as a last resort option for customizations
- In the iot-adk-addonkit, this file is created for each product under the product directory. Add feature id **CUSTOM\_CMD** in the OEMInput xml file to include this in the image.
- See [Custom.Cmd](#) package and [sample oemcustomizations.cmd](#)

### 2. Customizations.xml

- This is the settings file used to create the provisioning package
- To automatically process this provisioning package at boot time, this package is placed in `c:\windows\provisioning\packages`
- In the iot-adk-addonkit, this file is created for each product under the product directory. Add feature id **PROV\_AUTO** in the OEMInput xml file to include this in the image.
- See [Provisioning.Auto](#) package and [sample Customizations.xml](#)
- For more details, refer to:
  - [Add a provisioning package](#)
  - [Provisioning](#) for supported Configuration Service Providers (CSPs) in IoT Core

# IoT Core feature list

5/14/2019 • 10 minutes to read

Here's the features you can add to Windows 10 IoT Core (IoT Core) images.

Add features using the **OEMInput** XML file. To learn more, see the [IoT Core manufacturing guide](#).

## Retail features defined by Microsoft

The following table describes the Microsoft-defined features that can be used by OEMs in the Features element in the **OEMInput** file for **Retail** build.

When creating images for your device, determine which features are required for your device.

### Features

FEATURES	DESCRIPTION
<b>IOT_EFIESP</b>	Boots the device using UEFI, required feature in all images.
<b>IOT_UAP_OOBE</b>	Includes the inbox OOBE app that is launched during the first boot and also during installation of apps, required feature in all images.
<b>IOT_CRT140</b>	Adds CRT binaries, required feature in all images.
<b>IOT_UNIFIED_WRITE_FILTER</b>	Adds <a href="#">Unified Write Filter (UWF)</a> to protect physical storage media from data writes.
<b>IOT_USBFN_CLASS_EXTENSION</b>	Adds USB function WDF class extension for USB function mode support.
<b>IOT_POWERSHELL</b>	Adds PowerShell
<b>IOT_ALLJOYN_APP</b>	Adds the AllJoyn application, used for Headless ZwaveAdapterAppx.
<b>IOT_ONBOARDING_APP</b>	Provides a means of setting up the device's WiFi connection if no other WiFi profile was configured. It places the WiFi adapter into a Soft-AP mode so a phone or other device can connect to it.
<b>IOT_FONTS_CHINESE_EXTENDED</b>	Adds additional Chinese fonts.
<b>IOT_APP_TOOLKIT</b>	Adds tools required for Appx installation and management.
<b>IOT_FFU_FLASHMODE</b>	Adds flashing mode support so that the device can be flashed using ffutool. Currently supported for arm only.
<b>IOT_MTP</b>	Adds Media transfer protocol support. See <a href="#">MTP</a> .

FEATURES	DESCRIPTION
<b>IOT_MIRACAST_RX_APP</b>	Adds Connect App that supports Miracast receive feature. Note that the underlying hw/drivers should support Miracast for this app to work. Currently supported for arm only.
<b>IOT_WEBB_EXTN</b>	Adds <a href="#">Windows Device Portal</a> . If you are building an open retail device for commercial deployment to a "specific/limited installation" (i.e. factory or retail store) where the end-user does the final configuration and you document your customers that they must <b>obtain a certificate for WDP and install it on both WDP and connecting browsers and passwords are changed on WDP</b> , then using WDP in this narrow commercial instance is acceptable.
<b>IOT_BLUETOOTH_A2DP_SINK</b>	Allows an audio device to play to the device. This is new in Windows 10, version 1809
<b>IOT_BLUETOOTH_A2DP_SOURCE</b>	Allows the device to play to an external device (e.g. a Bluetooth speaker). This is available as a separate feature in Windows 10, version 1809. In Windows 10, version 1803, this was included in the image by default.
<b>IOT_BLUETOOTH_HFP_AUDIOGATEWAY</b>	HFP AudioGateway allows the device to serve as a gateway device for phone calls. An application can interact with the calling functions of a paired Bluetooth headset by using APIs under Windows.ApplicationModel.Calls, specifically the VoipPhoneCall class. .
<b>IOT_HEADLESS_CONFIGURATION</b>	Configures device to boot into Headless mode, where the UI stack is disabled and foreground apps will not launch
<b>IOT_NARRATOR</b>	Adds support for the Windows 10 screen-reading functionality, Narrator.
<b>IOT_OCR_ALL_LANGS</b>	Adds Optical Character Recognition (OCR) for all supported languages, including English
<b>IOT_OCR_EN_US</b>	Adds Optical Character Recognition (OCR) support for English. Do not use with <b>IOT_OCR_ALL_LANGS</b>
<b>IOT_HWN_CLASS_EXTENSION</b> (Deprecated)	Adds hardware notification WDF class extension for vibration API support. Deprecated in Windows 10, version 1709, as this feature is added by default
<b>IOT_NETCMD</b> (Deprecated)	Adds the command-line tool: netcmd.exe, used for configuring network connectivity. Deprecated in Windows 10, version 1803. The netcmd.exe will be removed when updating to version 1803. Use <a href="#">Windows.Devices.WiFi.WiFiAdapter</a> for managing Wifi. See <a href="#">WiFi Connector</a> example.
<b>IOT_APPLICATIONS</b> (Deprecated)	Deprecated in Windows 10, 1809 release along with IOT_CORTANA feature. Adds Account Management host application, enables MSA sign-in. Required for Cortana.

## Settings

FEATURES	DESCRIPTION
<b>IOT_POWER_SETTINGS</b>	Prevents the device from going to sleep due to inactivity. Required for x86/amd64 platforms. This feature supports ARM starting with Windows 10, Version 1703.
<b>IOT_EFIESP_BCD</b>	Sets boot configuration data (BCD) for GPT-based drives. Required for x86/amd64. MBR devices should use <b>IOT_EFIESP_BCD_MBR</b> .
<b>IOT_EFIESP_BCD_MBR</b>	Sets boot configuration data (BCD) for MBR-based drives.
<b>IOT_SHELL_HOTKEY_SUPPORT</b>	Adds support to launch default app using a hotkey: <a href="#">VK_LWIN</a> ( <a href="#">Left Windows key</a> ).
<b>IOT_SHELL_ONSCREEN_KEYBOARD</b>	Adds available on-screen keyboard.
<b>IOT_SHELL_ONSCREEN_KEYBOARD_FOLLOWFOCUS</b>	Enables on-screen keyboard to automatically appear when input field is focused. Requires <b>IOT_SHELL_ONSCREEN_KEYBOARD</b> .
<b>IOT_DISABLEBASICDISPLAYFallback</b>	Disables the inbox basic render driver. This feature should only be used with the Qualcomm DragonBoard (DB).
<b>IOT_CRASHCONTROL_SETTINGS</b>	Configures the device to auto reboot without showing blue screen (BSOD) when the device crashes. This also disables crashdump. [AutoReboot = 1 ; DisplayDisabled = 1 and CrashDumpEnabled = 0]. See <a href="#">Crash settings</a>
<b>IOT_SSH</b>	Enables Secure Shell (SSH) connectivity
<b>IOT_GENERIC_POP</b> (Deprecated)	In Windows 10 1809, this is deprecated and the device will get OS only updates by default. Adds the Generic device targeting info for OS only Updates.

## Developer Tools

### IMPORTANT

The following developer features should not be used in **Retail** builds and in images for commercial devices. If you would still like to disable IOT\_SIREP as a developer tool, however, please follow the instructions [here](#).

FEATURES	DESCRIPTION
<b>IOT_SIREP</b>	Enables SIREP service for TShell connectivity.
<b>IOT_TOOLKIT</b>	Includes developer tools such as: Kernel Debug components, FTP, Network Diagnostics, basic device portal, and XPerf. This also relaxes the firewall rules and enables various ports.
<b>IOT_NANORDPSERVER</b>	Adds <a href="#">Remote Display packages</a> . Note: Remote Display is prerelease software intended for development and training purposes only.

FEATURES	DESCRIPTION
<b>IOT_BERTHA</b>	Adds a sample app: "Bertha". This app provides basic version info and connectivity status.
<b>IOT_UAP_DEFAULTAPP</b>	Adds a sample app, "Chucky". This app is similar to "Bertha".
<b>IOT_FTSER2K_MAKERDRIVER</b>	Adds the FTDI USB-to-Serial driver.
<b>IOT_CP210x_MAKERDRIVER</b>	Adds drivers for SiliconLabs CP210x-based USB to Serial adapters.
<b>IOT_DMAP_DRIVER</b>	Adds DMAP drivers.
<b>IOT_CONTAINERS</b>	Adds support for native <a href="#">Nano Server Containers</a> . These are supported on Intel 64-bit platforms (since Windows 10, version 1709) and ARM32 platform (since Windows 10, version 1809).
<b>IOT_CORTANA</b> (Deprecated)	Deprecated in Windows 10, 1809 release. See Cortana SDK for including Cortana to your device. Adds Cortana feature. Requires <b>IOT_APPLICATIONS</b> feature.
<b>IOT_CORTANA_OBSCURELAUNCH</b> (Deprecated)	Deprecated in Windows 10, 1809 release along with IOT_CORTANA feature. Enables running Cortana application on boot. This add-on causes Cortana to run in the background resulting in better response time for Cortana.

## Speech Data

FEATURES	DESCRIPTION
<b>IOT_SPEECHDATA_AR_SA</b>	Adds speech data for Arabic (Saudi Arabia).
<b>IOT_SPEECHDATA_DE_DE</b>	Adds speech data for German (Germany).
<b>IOT_SPEECHDATA_EL_GR</b>	Adds speech data for Greek.
<b>IOT_SPEECHDATA_EN_CA</b>	Adds speech data for English (Canada).
<b>IOT_SPEECHDATA_EN_GB</b>	Adds speech data for English (UK).
<b>IOT_SPEECHDATA_ES_ES</b>	Adds speech data for Spanish (Spain).
<b>IOT_SPEECHDATA_ES_MX</b>	Adds speech data for Spanish (Mexico).
<b>IOT_SPEECHDATA_FR_CA</b>	Adds speech data for French (Canada).
<b>IOT_SPEECHDATA_FR_FR</b>	Adds speech data for French (France).
<b>IOT_SPEECHDATA_IT_IT</b>	Adds speech data for Italian.
<b>IOT_SPEECHDATA_JA_JP</b>	Adds speech data for Japanese.

FEATURES	DESCRIPTION
<b>IOT_SPEECHDATA_KO_KR</b>	Adds speech data for Korean.
<b>IOT_SPEECHDATA_NL_NL</b>	Adds speech data for Dutch.
<b>IOT_SPEECHDATA_PL_PL</b>	Adds speech data for Polish.
<b>IOT_SPEECHDATA_PT_BR</b>	Adds speech data for Portuguese (Brazil).
<b>IOT_SPEECHDATA_PT_PT</b>	Adds speech data for Portuguese (Portugal).
<b>IOT_SPEECHDATA_RO_RO</b>	Adds speech data for Romanian.
<b>IOT_SPEECHDATA_RU_RU</b>	Adds speech data for Russian.
<b>IOT_SPEECHDATA_ZH_CN</b>	Adds speech data for Chinese (Mainland).
<b>IOT_SPEECHDATA_ZH_HK</b>	Adds speech data for Chinese (Hong Kong). Do not include <b>IOT_SPEECHDATA_ZH_TW</b> .
<b>IOT_SPEECHDATA_ZH_TW</b>	Adds speech data for Chinese (Taiwan). Do not include <b>IOT_SPEECHDATA_ZH_HK</b> .
<b>IOT_SPEECHDATA_EN_US</b> (Deprecated)	Deprecated in Windows 10, version 1607. Do not add this feature. The default image includes speech data for English (US).

## Features in the IoT Core Add-Ons

### NOTE

The packages corresponding to these features are available in source in the `iot-adk-addonkit`. You can modify them to suit your requirements.

FEATURES	DESCRIPTION
<b>CUSTOM_CMD</b>	Feature to include the <code>oemcustomization.cmd</code> . This is product-specific and picks up the input file from product directory. <code>OEM_CustomCmd</code> is the deprecated feature ID, can still be used for legacy builds. See <a href="#">Runtime Customizations</a>
<b>CUSTOM_BCD</b>	Includes <a href="#">BCD settings</a> to suppress boot UX progress display and also enables flight signing. Modify <a href="#">Custom.BCD.xml</a> to remove flight signing.
<b>CUSTOM_OOBAPP</b>	Includes customizations for the <a href="#">OOBE App</a>
<b>PROV_AUTO</b>	Feature to <a href="#">add a provisioning package to an image</a> . This is product specific and picks up the input <code>ppkg</code> file from the product directory. <code>OEM_ProvAuto</code> is the deprecated feature ID, can still be used for legacy builds. See <a href="#">Runtime customizations</a>

FEATURES	DESCRIPTION
<b>RECOVERY_BCD</b>	Includes recovery BCD settings for GPT devices. See <a href="#">Add a recovery mechanism</a>
<b>RECOVERY_BCD_MBR</b>	Includes recovery BCD settings for MBR devices. See <a href="#">Add a recovery mechanism</a>
<b>SEC_BITLOCKER</b>	Includes the configuration for Bitlocker
<b>SEC_SECUREBOOT</b>	Includes the retail configuration for Secure Boot
<b>SEC_SECUREBOOT_TEST</b>	Includes the test configuration for Secure Boot
<b>SEC_DEVICEGUARD</b>	Includes the retail configuration for DeviceGuard
<b>SEC_DEVICEGUARD_TEST</b>	Includes the test configuration for DeviceGuard
<b>SETTINGS_HOTKEY</b>	Feature to demonstrate how to <a href="#">add a registry setting to an image</a> . Read <a href="#">Switching between apps</a> for more details.

## Test features

The following table describes the Microsoft-defined test features that can be used by OEMs in the Features element in the **OEMInput** file for **Test** builds ONLY.

FEATURES	DESCRIPTION
<b>IOT_BCD_FLASHMODE_SETTING</b>	Enables FFU flashing setting.
<b>IOT_DISABLE_TESTSIGNING</b>	Disables runtime-installation of test-signed packages.
<b>IOT_EFIESP_TEST</b>	UEFI packages required for booting test images. Should not be used with <b>IOT_EFIESP</b> .
<b>IOT_ENABLE_ADMIN</b>	Enables the Administrator account with default password 'p@ssw0rd'.
<b>IOT_ENABLE_TESTSIGNING</b>	Enables run-time installation of test-signed packages. Allows test-signed drivers and (.appx) apps to run.
<b>IOT_KD_ON</b>	Enables Kernel Debugger
<b>IOT_KDNETUSB_SETTINGS</b>	Includes all kernel debugger transports and enables KDNET over USB. The default debug transport settings for this feature are an IP address of "1.2.3.4", a port address of "50000", and a debugger key of "4.3.2.1". To use the default IP address of 1.2.3.4, run VirtEth.exe with the /autodebug flag. For example, to establish a kernel debugger connection to the phone, use the command: <code>Windbg -k net:port=50000,key=4.3.2.1</code>
	<b>Note</b> Do not include either <b>IOT_KDUSB_SETTINGS</b> or <b>IOT_KDNETUSB_SETTINGS</b> if you need to enable MTP or IP over USB in the image. If the kernel debugger is enabled in the image and the debug transports are used to connect to the device, the kernel debugger has exclusive use of the USB port and prevents MTP and IP over USB from working.

FEATURES	DESCRIPTION
<b>IOT_KDSERIAL_SETTINGS</b>	Includes all kernel debugger transports and enables KDSERIAL with the following settings: 115200 Baud, 8 bit, no parity. These settings apply to x86 and amd64 platforms. ARM platforms use UEFI-defined serial transport settings.
<b>IOT_KDUSB_SETTINGS</b>	Includes all kernel debugger transports and enables KDUSB. The default debug transport target name for this feature is <b>WOATARGET</b> . To establish a kernel debugger connection to the phone, use the command: <code>Windbg -k usb:targetname=WOATARGET</code> . <b>Note</b> Do not include either <b>IOT_KDUSB_SETTINGS</b> or <b>IOT_KDNETUSB_SETTINGS</b> if you need to enable MTP or IP over USB in the image. If the kernel debugger is enabled in the image and the debug transports are used to connect to the device, the kernel debugger has exclusive use of the USB port and prevents MTP and IP over USB from working.
<b>IOT_WDTF</b>	Includes components for Windows Driver Test Framework, required for HLK validation.
<b>IOT_DIRECTX_TOOLS</b>	Adds DirectX tools.
<b>IOT_UMDFDBG_SETTINGS</b>	Includes user-mode driver framework debug settings.
<b>IOT_DISABLE_UMCI</b> (Deprecated)	Disables the code integrity check. Deprecated in Windows 10, version 1709.

## Features per release

The following table provides an overview of supported features per IoT Core OS release, listed in alphabetical order.

FEATURES	1809 (17731.X)	1803 (17134.X)	1709 (16299.X)	1703 (15063.X)	1607 (14393.X)
<b>IOT_ALLJOYN_APP</b>	x	x	x	x	x
<b>IOT_APPLICATIONS</b> (Deprecated)	N/A	x	x	x	
<b>IOT_APP_TOOLKIT</b>	x	x	x	x	x
<b>IOT_BCD_FLASH_MODE_SETTING</b>	x				
<b>IOT_BERTHA</b>	x	x	x	x	x
<b>IOT_BLUETOOTH_A2DP_SINK</b>	x				
<b>IOT_BLUETOOTH_A2DP_SOURCE</b>	x				

FEATURES	1809 (17731.X)	1803 (17134.X)	1709 (16299.X)	1703 (15063.X)	1607 (14393.X)
<b>IOT_BLUETOOTH_HFP_AUDIO_GATEWAY</b>	x				
<b>IOT_CONTAINERS</b>	x (x64,arm32)	x(x64)	x(x64)		
<b>IOT_CORTANA</b> (Deprecated)	N/A	x	x	x	
<b>IOT_CORTANA_OBSCURELAUNCH</b> (Deprecated)	N/A	x	x	x	
<b>IOT_CP210x_MAKERDRIVER</b>	x	x	x	x	
<b>IOT_CRASHCONTROL_SETTINGS</b>	x	x			
<b>IOT_CRT140</b>	x	x	x	x	x
<b>IOT_DIRECTXT_TOOLS</b>	x	x	x	x	x
<b>IOT_DISABLE_FIRMWARESIGNING</b>	x	x	x	x	x
<b>IOT_DISABLE_TESTSIGNING</b>	x	x	x	x	x
<b>IOT_DISABLE_UHCI</b> (Deprecated)	N/A	N/A	N/A	x	x
<b>IOT_DISABLEBACKDISPLAY</b>	x	x	x	x	x
<b>IOT_DMAP_DRIVER</b>	x	x	x	x	x
<b>IOT_EFIESP</b>	x	x	x	x	x
<b>IOT_EFIESP_BCD</b>	x	x	x	x	x
<b>IOT_EFIESP_BCD_MBR</b>	x	x	x	x	
<b>IOT_EFIESP_TEST</b>	x	x	x	x	x
<b>IOT_ENABLE_ADDRESSIN</b>	x	x	x	x	x

FEATURES	1809 (17731.X)	1803 (17134.X)	1709 (16299.X)	1703 (15063.X)	1607 (14393.X)
<b>IOT_ENABLE_FLI GHTSIGNING</b>	X	X	X	X	X
<b>IOT_ENABLE_TE STSIGNING</b>	X	X	X	X	X
<b>IOT_FFU_FLASH MODE</b>	x(arm)	x(arm)			
<b>IOT_FONTS_CHI NESE_EXTENDED</b>	X	X	X	X	
<b>IOT_FTSER2K_M AKERDRIVER</b>	X	X	X	X	X
<b>IOT_GENERIC_P0 P (Deprecated)</b>	N/A	X	X	X	X
<b>IOT_HWN_CLASS _EXTENSION (Deprecated)</b>	N/A	N/A	N/A	X	
<b>IOT_HEADLESS_ CONFIGURATIO N</b>	X				
<b>IOT_KD_ON</b>	X	X	X	X	X
<b>IOT_KDNETUSB_ SETTINGS</b>	X	X	X	X	X
<b>IOT_KDSERIAL_S ETTINGS</b>	X	X	X	X	X
<b>IOT_KDUSB_SET TINGS</b>	X	X	X	X	X
<b>IOT_MIRACAST_ RX_APP</b>	X	X			
<b>IOT_MTP</b>	X	X			
<b>IOT_NANORDPS ERVER</b>	X	X	X	X	X
<b>IOT_NETCMD (Deprecated)</b>	N/A	N/A	N/A	X	X
<b>IOT_NARRATOR</b>	X				
<b>IOT_OCR_ALL_L ANGS</b>	X				

FEATURES	1809 (17731.X)	1803 (17134.X)	1709 (16299.X)	1703 (15063.X)	1607 (14393.X)
IOT_OCR_EN_US	X				
IOT_ONBOARDING_APP	X	X	X		
IOT_POWER_SETTINGS	X	X	X	X	x (x86/x64)
IOT_POWERSHELL	X	X	X	X	X
IOT_SHELL_HOTKEY_SUPPORT	X	X	X	X	X
IOT_SHELL_ONSCREEN_KEYBOARD	X	X	X	X	
IOT_SHELL_ONSCREEN_KEYBOARD_FOLLOWFOCUS	X	X	X	X	
IOT_SIREP	X	X	X	X	X
IOT_SPEECHDATA_AR_SA	X				
IOT_SPEECHDATA_DE_DE	X	X	X	X	X
IOT_SPEECHDATA_EL_GR	X				
IOT_SPEECHDATA_EN_CA	X	X	X	X	
IOT_SPEECHDATA_EN_GB	X	X	X	X	X
IOT_SPEECHDATA_EN_US (Deprecated)	N/A	N/A	N/A	N/A	X
IOT_SPEECHDATA_ES_ES	X	X	X	X	X
IOT_SPEECHDATA_ES_MX	X	X	X	X	
IOT_SPEECHDATA_FR_CA	X	X	X	X	

FEATURES	1809 (17731.X)	1803 (17134.X)	1709 (16299.X)	1703 (15063.X)	1607 (14393.X)
<b>IOT_SPEECHDAT A_FR_FR</b>	x	x	x	x	x
<b>IOT_SPEECHDAT A_IT_IT</b>	x	x	x	x	x
<b>IOT_SPEECHDAT A_JA_JP</b>	x	x	x	x	x
<b>IOT_SPEECHDAT A_KO_KR</b>	x				
<b>IOT_SPEECHDAT A_NL_NL</b>	x				
<b>IOT_SPEECHDAT A_PL_PL</b>	x				
<b>IOT_SPEECHDAT A_PT_BR</b>	x				
<b>IOT_SPEECHDAT A_PT_PT</b>	x				
<b>IOT_SPEECHDAT A_RO_RO</b>	x				
<b>IOT_SPEECHDAT A_RU_RU</b>	x				
<b>IOT_SPEECHDAT A_ZH_CN</b>	x	x	x	x	x
<b>IOT_SPEECHDAT A_ZH_HK</b>	x	x	x	x	x
<b>IOT_SPEECHDAT A_ZH_TW</b>	x	x	x	x	x
<b>IOT_SSH</b>	x	x	x	x	x
<b>IOT_TOOLKIT</b>	x	x	x	x	x
<b>IOT_UAP_DEFAU LTAPP</b>	x	x	x	x	x
<b>IOT_UAP_OOBE</b>	x	x	x	x	x
<b>IOT_UMDFDBG_ SETTINGS</b>	x	x	x	x	x
<b>IOT_UNIFIED_W RITE_FILTER</b>	x	x	x	x	x

FEATURES	1809 (17731.X)	1803 (17134.X)	1709 (16299.X)	1703 (15063.X)	1607 (14393.X)
<b>IOT_USBFN_CLASSES_EXTENSION</b>	X	X	X	X	X
<b>IOT_WDTF</b>	X	X	X	X	X
<b>IOT_WEBB_EXTN</b>	X	X	X	X	X

## Related topics

[What's in the Windows ADK IoT Core Add-ons](#)

[IoT Core manufacturing guides](#)

# Windows ADK IoT Core Add-ons: contents

6/24/2019 • 3 minutes to read

The [Windows 10 IoT Core ADK Add-Ons](#) include OEM-specific tools to create images for your IoT Core devices with your apps, board support packages (BSPs), settings, drivers, and features.

This kit

- makes IoT Core image creation process easy and simple
- enables creation of multiple images/image variants easily
- provides automation support for nightly builds

The [IoT Core manufacturing guide](#) walks you through building images with these tools.

## Key XML definitions

- Package definitions (\*.wm.xml) : defines a component package
- Provisioning definitions (customizations.xml) : source file for provisioning settings
- Feature manifests (\*.FM.xml) : defines feature composition and feature IDs
- Feature manifest List (\*.FMList.xml) : enumerates the FM files
- Product definitions (\*.OEMInputFile.xml) : specifies the product composition with the Microsoft features and OEM features included in the product

NAME	FILENAME.EXT	ADK TOOL	BUILD COMMAND	OUTPUT
Package	*.wm.xml	pkggen.exe	buildpkg.cmd	*.cab
Provisioning	customizations.xml	icd.exe	buildprovpkg.cmd	*.ppkg
Feature manifest	*.FM.xml	featuremerger.exe imageapp.exe	-	-
Feature manifest list	*.FMList.xml	featuremerger.exe	buildfm.cmd	MergedFM/*FM.xml , *FIP.cab
Product	*.OEMInputFile.xml	imageapp.exe	buildimage.cmd	*.ffu

## Code Architecture

- Root folder
  - IoTCorePShell.cmd: Launches the IoT Core Powershell
  - IoTCoreShell.cmd: Launches the IoT Core commandline shell
  - README.md: Version info, links to documentation
- Scripts
  - This contains helper powershell scripts and sample build scripts.
- Tools
  - CmdTools, containing commandline wrapper tools to invoke powershell commands
  - IoTCoreImaging, containing the powershell module and scripts. See [IoT Core Add-ons Powershell tools](#)

- README.md : Documentation on the powershell tools
- Workspace
  - IoTWorkspace.xml
    - XML file containing the workspace configuration information such as supported architecture, security settings etc.
  - Build
    - This is the output directory where the build contents are stored. It starts as empty.
  - Common/Packages
    - Architecture *independent*, platform *independent* packages
    - OEMCommonFM.xml - feature manifest file that enumerates common packages and defines common features.
  - Source-<arch>
    - Packages
      - Architecture *specific*, platform *independent* packages
      - OEMFM.xml - the feature manifest file that enumerates arch specific packages and defines arch specific features.
      - OEMFMList.xml - enumeration of OEM FM files.
    - BSP
      - <bspname>/Packages
        - Architecture *specific*, platform *specific* packages
        - <bspname>FM.xml - feature manifest that enumerates the bsp packages and defines supported device layouts and features
        - <bspname>FMList.xml - enumeration of BSP FM files.
      - <bspname>/OemInputSamples
        - sample oeminput files demonstrating how to use the bsp, these files are used as templates in `newproduct.cmd`
    - Products
      - architecture specific named products

## Sample packages

Sample packages are provided in the iot-adk-addonkit that can be used as a reference or as is in your image, if it meets your needs. Few of such packages are listed here.

### Common Packages

PACKAGE NAME	DESCRIPTION
Registry.Version	Package containing registry settings with product and version information.
DeviceLayout.GPT4GB	Package with <a href="#">GPT drive/partition layout</a> for UEFI-based devices with 4GB drives.
DeviceLayout.GPT8GB-R	Package with GPT drive/partition layout for UEFI-based devices with 8GB drives with recovery partition.
DeviceLayout.MBR4GB	Package with MBR drive/partition layout for legacy BIOS-based devices with 4GB drives.

PACKAGE NAME	DESCRIPTION
DeviceLayout.MBR8GB-R	Package with MBR drive/partition layout for legacy BIOS-based devices with 8GB drives with recovery partition.

## Applications and Services packages

PACKAGE NAME	DESCRIPTION
Appx.IoTCoreDefaultApp	Foreground apps package containing <a href="#">IoTCoreDefaultApp</a> , see <a href="#">description</a> .
Appx.DigitalSign	Foreground apps package containing <a href="#">DigitalSign</a> , see <a href="#">description</a> .
Appx.IOTOnboardingTask	Background apps package containing <a href="#">IOTOnboardingTask</a> , see <a href="#">description</a> .
AzureDM.Services	Service package containing Azure Device Management

## BSP

Source files to create board support packages (BSPs).

Some BSPs are included in each folder as a start. You can [create your own BSPs](#) based on these packages.

## Driver packages

PACKAGE NAME	DESCRIPTION
Drivers.GPIO	Sample package for adding a driver.

## Products

Source file for product configurations. Use our samples (SampleA, SampleB) or [create your own](#).

PRODUCT	DESCRIPTION
SampleA	Product with Microsoft provided features / apps
SampleB	Product using OEM Apps and OEM drivers
SingleLang	Product with single non english language support
MultiLang	Product with multiple language support
SecureSample	Product using security features
RecoverySample	Product using recovery mechanism
DigitalSign	Sample real life product using various features Cortana, recovery mechanism, security features, multi languages support

## Related topics

[IoT Core manufacturing guides](#)

[IoT Core feature list](#)

[IoT Core Image Wizard](#)

# IoT Core Add-ons Powershell Commands

1/18/2019 • 2 minutes to read

## NOTE

IoT Core Add-ons command-line is deprecated. Refer to [IoT Core Add-ons command-line options](#) for the old list of commands.

The Powershell version of the [Windows 10 IoT Core \(IoT Core\) ADK Add-Ons](#) supports the following commands. These are part of the powershell module `IoTCoreImaging`. To learn more about these tools, see [What's in the Windows ADK IoT Core Add-ons](#).

## Powershell Commands with Alias

### **Add-IoTAppxPackage (newappxpkg)**

Creates Appx OEM package and adds featureID to OEMFM.xml

### **Add-IoTBSP (newbsp)**

Adds new bsp based on a template

### **Add-IoTBitLocker**

Adds bitlocker package for the product

### **Add-IoTCommonPackage (newcommonpkg)**

Creates common (file/reg) OEM package and adds featureID to OEMCOMMONFM.xml

### **Add-IoTDeviceGuard**

Adds device guard package

### **Add-IoTDriverPackage (newdrvpkg)**

Creates Driver OEM package and adds featureID to OEMFM.xml

### **Add-IoTFilePackage**

Adds a file package and adds the featureID to OEMCOMMONFM.xml

### **Add-IoTProductFeature (addfid)**

Adds feature id to the product's oeminput xml file

### **Add-IoTProduct (newproduct)**

Adds new product based on the OEMInputSamples from BSP

### **Add-IoTProvisioningPackage (newprovpkg)**

Adds provisioning oem package and adds the featureID to OEMCOMMONFM.xml

### **Add-IoTRegistryPackage**

Adds a registry package and adds the featureID to OEMCOMMONFM.xml

### **Add-IoTSecureBoot**

Adds secureboot package for the product

### **Add-IoTSecurityPackages**

Adds security packages for the product

### **Add-IoTSignature (signbinaries)**

Signs files with the cert set via Set-IoTSignature

### **Convert-IoTPkg2Wm (convertpkg)**

Converts pkg.xml files to wm.xml files

### **Copy-IoTBSP (copybsp)**

Copies BSP between workspaces

### **Copy-IoTOEMPackage (copypkg)**

Copies OEM package between workspaces

### **Copy-IoTProduct (copyproduct)**

Copies product between workspaces

### **Dismount-IoTFFUImage (ffud)**

Dismounts the FFU image

### **Export-IoTDUCCab (exportpkgs)**

Exports the update cab for DUC upload

### **Export-IoTDeviceModel (exportidm)**

Exports the IoT Device Model for DUC registration

### **Export-IoTFFUAsWims (ffue)**

Exports the EFIESP/MainOS/Data partitions as Wims

### **Get-IoTFFUDrives (ffugd)**

Returns a hashtable of the drive letters for the mounted partitions

### **Get-IoTProductFeatureIDs (gpfids)**

Gets features IDs supported in the IoTCore OS

### **Get-IoTProductPackagesForFeature (gpfidpkgs)**

Gets OS packages corresponding to features ID

### **Get-IoTWorkspaceBSPs (gwsbps)**

Gets the list of BSP names in the workspace

### **Get-IoTWorkspaceProducts (gwsproducts)**

Gets the list of product names in the workspace

### **Import-IoTBSP (importbsp)**

Imports BSP from the given folder / zip file or sample workspace

### **Import-IoTCertificate**

Imports the certificate for security functions

### **Import-IoTDUCCConfig (importcfg)**

Imports the CUSConfig.zip into the product directory

### **Import-IoTOEM Package (importpkg)**

Imports OEM package from Sample workspace

### **Import-IoTProduct (importproduct)**

Imports Product from Sample workspace

### **Install-IoTOEMCerts**

Installs oem pfx files in the certs\private folder

### **Mount-IoTFFUImage (ffum)**

Mounts the FFU image

### **New-IoTCabPackage (buildpkg)**

Creates `.cab` files

### **New-IoTFFUCIPolicy (ffus)**

Scans the MainOS partition and generates CI policy (`initialpolicy.xml`)

### **New-IoTFFUImage (buildimage)**

Creates regular FFU

### **New-IoTFIPPackage (buildfm)**

Creates FIP packages and merged FM files

### **New-IoTInf2Cab (inf2cab)**

Creates cab file for the given inf file

### **New-IoTOEMCerts**

Creates new OEM specific certificates

### **New-IoTProvisioningPackage (buildppkg)**

Creates `.ppkg` files

### **New-IoTRecoveryImage (buildrecovery)**

Creates recovery FFU

### **New-IoTWindowsImage (newwinpe)**

Creates custom winpe with bsp drivers / recovery scripts

### **New-IoTWorkspace (new-ws)**

Creates new workspace

### **Open-IoTWorkspace (open-ws)**

Opens existing workspace

### **Redo-IoTCabSignature (re-signcabs)**

Resigns cab and its contents using Add-IoTSignature

### **Redo-IoTWorkspace (migrate)**

Converts legacy iot-adk-addonkit directory into a workspace

### **Remove-IoTProductFeature (removefid)**

Removes feature id from the product's oeminput xml file

### **Set-IoTCabVersion (setversion)**

Stores the version in the IoTWorkspace.xml

### **Set-IoTEnvironment (setenv)**

Sets environment settings based on the config values in IoTWorkspace.xml

### **Set-IoTRetailSign (retailsign)**

Sets/resets use of the retail code signing certificate

#### **Set-IoTSignature (setsignature)**

Sets the Certificate info used for signing

#### **Test-IoTCabSignature**

Tests if the Cab package and its contents are signed for the given config

#### **Test-IoTFeatures**

Tests if all feature ids are defined, for the given product / config

#### **Test-IoTPackages**

Tests all packages and its contents are signed, for the given product / config

#### **Test-IoTRecoveryImage (verifyrecovery)**

Verifies if the wim files in the recovery ffu are proper

#### **Test-IoTSignature**

Tests if the file is signed for the given config

## Related topics

[IoT Core Add-ons](#)

[IoT Core Image Wizard](#)

[IoT Core manufacturing guide](#)

# IoT Core Image Wizard (Preview release)

1/25/2019 • 2 minutes to read

The IoT Core Image Wizard is a GUI tool that uses the [IoT Core Add-ons Powershell Commands](#). Using the IoT Core Image Wizard simplifies the process of creating your first image for your device.

## Download link

Download the [IoT Core Image Wizard](#).

## Setup instructions

1. Download the zip file
2. Extract to a folder
3. See the readme.md in the extracted files for the prerequisites
4. Starting with the Windows 10 October 2018 Update, you use the RSAT: Server Manager optional feature.  
Older versions of Windows 10 use the matching [package](#)
5. Once prerequisites are installed, go to the extracted files' folder and run IoTCoreImageWizard.exe

## November 2018 Preview release notes

- Only supports creating a new workspace. Cannot open/edit a workspace. Use the [Powershell Commands](#) to open/edit a workspace.
- Just debug configurations of image builds. If you need to build retail images, use the [Powershell Commands](#).
- Errors with creating a recovery image. Make sure the OS version matches or is newer than the kits' version.  
Also, all the kits need to be the same version.

## Related topics

[IoT Core Add-ons](#)

[IoT Core Add-ons Powershell Commands](#)

[IoT Core manufacturing guide](#)

# Update the time server

1/24/2019 • 2 minutes to read

By default, IoT Core devices are setup to synchronize time from time.windows.com. If you don't have internet connectivity or behind a firewall, then you'll need to synchronize the system time for your IoT Core devices to a time server reachable in your network. You can change the time server or add multiple time servers using the information below.

## Update the server from a command line (for example, using a tool like PuTTY):

1. Identify the required NTP server(s) and make sure you can reach them from your network. For example, if time.windows.com, NTPServer1, NTPServer2 are the three desired NTP servers, make sure the following commands succeed when run on a Windows computer on the network before using in an IoT device:

```
W32tm.exe /stripchart /computer:time.windows.com /samples:5  
W32tm.exe /stripchart /computer:NtpServer1 /samples:5  
W32tm.exe /stripchart /computer:NtpServer2 /samples:5
```

2. Modify the W32Time service configuration on the IoT device to use your NTP time server(s).

```
reg add HKLM\SYSTEM\CurrentControlSet\Services\w32time\Parameters /v NtpServer /t REG_SZ /d  
"time.windows.com,0x9 NtpServer1,0x9 NtpServer2,0x9" /f >nul 2>&1
```

3. Restart the time service

```
net stop w32time  
net start w32time
```

4. Verify the time servers from which the device is currently receiving time. If you restarted the time service, allow a minute or so before verifying the time service.

```
W32tm.exe /query /peers
```

## Update the server in an IoT Core image

1. Create a package definition file, and add it to the image. To learn more, see [Lab 1c: Add a file and a registry setting to an image](#). Sample script:

```
<regKeys>  
  <regKey  
    keyValue="$(hk1m.system)\CurrentControlSet\Services\w32time\Parameters">  
    <regValue name="NtpServer" type="REG_SZ"  
      value="time.windows.com,0x9 NtpServer1,0x9 NtpServer2,0x9" />  
  </regKey>  
</regKeys>
```

# Add a recovery mechanism to your image

1/18/2019 • 2 minutes to read

You can add a recovery mechanism to your image with **WinPE** as a Safe OS and **WIM files** as Recovery SW from recovery partition, using the steps provided below.

See [Windows 10 IoT Core Recovery](#) for the details on the possible mechanisms.

## Step 1 : Update device layout with recovery partition

In the devicelayout.xml file, you add a new partition **MMOS** with the following attributes

- FAT32 filesystem
- Atleast 2GB size ( to accommodate WinPE wim and recovery wims)
- Partition type
  - GPT :{ebd0a0a2-b9e5-4433-87c0-68b6b72699c7}
  - MBR :0x07

Sample xml snippet given below for a GPT device (assumes a sector size of 512)

```
<Partition>
  <Name>MMOS</Name>
  <FileSystem>FAT32</FileSystem>
  <TotalSectors>4096000</TotalSectors>
  <Type>{ebd0a0a2-b9e5-4433-87c0-68b6b72699c7}</Type>
</Partition>
```

See also [QCDB410C device layout](#)

Sample xml snippet given below for an MBR device

```
<Partition>
  <Name>MMOS</Name>
  <FileSystem>FAT32</FileSystem>
  <TotalSectors>4096000</TotalSectors>
  <Type>0x07</Type>
</Partition>
```

See also [MBR 8GB Recovery device layout](#)

## Step 2 : Configure BCD settings

In this step, the newly added MMOS partition is defined as a bootable partition in the BCD settings and the recovery sequence is enabled and configured to boot into this partition. These settings are available in the below given packages that you can readily use. Select GPT or MBR packages based on your device.

- [Recovery.GPT-BCD package](#)
- [Recovery.MBR-BCD package](#)
  - Recovery.BCD.xml declares the MMOS partition availability.
- [Recovery.GPT-BcdEdit package](#)
- [Recovery.MBR-BcdEdit package](#)
  - Recovery.BcdEdit.cmd enables recovery sequence and configures to boot into the MMOS partition.

## Step 3 : Prepare WinPE image

Windows 10 ADK Release 1709 contains the Windows 10 Preinstall Environment for all architectures (x86/amd64 and arm). For Windows 10 ADK Release 1809, you will need to install the **Windows PE add-on for ADK**. In this WinPE, you add the following

- Recovery scripts used for recovery process on device
  - `startnet.cmd` , `startnet_recovery.cmd` : predefined scripts from the templates directory (see `templates\recovery`).
  - config files : generated files based on the device layout, placed at `Build\<arch>\<bspname>\recovery`.
- Recovery customizations files (optional)
  - `RecoveryGUI.exe` : Optional simple UI to hide the recovery shell prompt on the device. The `recoveryGUI.exe` can be a C++ application built for the target CPU or a .NET Framework 4 Windows from application. `Newwinpe.cmd` will have to be modified to add .NET Framework 4 capabilities to the WinPE image.
  - `pre_recovery_hook.cmd` and `post_recovery_hook.cmd` : optional hooks to add additional actions before and after recovery process.
  - Place these files in `Source-<arch>\bsp\<bspname>\WinPEExt\recovery` folder.
- BSP drivers (optional)
  - You may need to add bsp drivers to winpe image to boot/write to storage, on your device platform.
  - Place the required drivers in `Source-<arch>\bsp\<bspname>\WinPEExt\drivers` folder.

You can create the WinPE image for the bsp with the above contents using [New-IoTWindowsImage](#) command in IoTCorePShell

```
New-IoTWindowsImage <product> <config>
(or) newwinpe <product> <config>
```

This script will output the winpe at `Build\<arch>\<product>\<config>\winpe.wim`.

## Step 4 : Update Feature manifest file and OEMInputFile

- Update the `<bspname>FM.xml` with the following changes (see [QCDB410CFM.xml sample](#))
  - Include the new device layout package, specifying new SOC name, QC8016-R in the example below .

```
<DeviceLayoutPackages>
    <PackageFile SOC="QC8016-R" Path="%PKGBLD_DIR%" Name="%OEM_NAME%.QCDB410C.DeviceLayout-
R.cab" />
    <PackageFile SOC="QC8016" Path="%BSPPKG_DIR%" Name="Qualcomm.QC8916.DeviceLayout.cab" />
</DeviceLayoutPackages>
```

- Update the `<productname>/TestOEMInput.xml` (and `RetailOEMInput.xml`) with the following changes (see [Recovery sample](#))
  - Specify the SOC name as defined in the `<bspname>FM.xml`

```
<SOC>QC8016-R</SOC>
```

- Include the RECOVERY\_BCD feature in the OEM section

```
<OEM>
...
<Feature>RECOVERY_BCD</Feature>
...
</OEM>
```

- Update the `oemcustomization.cmd` to invoke the `Recovery.BcdEdit.cmd`

```
REM The below should be called on every boot
if exist C:\RecoveryConfig\Recovery.BcdEdit.cmd (
    call C:\RecoveryConfig\Recovery.BcdEdit.cmd
)
```

## Step 5 : Build the recovery image using [New-IoTRecoveryImage](#)

```
# Build all packages
New-IoTCabPackage All
(or) buildpkg All
# Build the product image
New-IoTFFUIImage <product> <config>
(or) buildimage <product> <config>
# Build the recovery image
New-IoTRecoveryImage <product> <config>
(or) buildrecovery <product> <config>
```

This will generate the recovery file as `Flash_Recovery.ffd`

# Windows 10 IoT Core Commercialization Frequently Asked Questions (FAQ)

7/4/2019 • 2 minutes to read

If you're looking to get a few questions answered regarding the commercialization process, you've come to the right place. Not finding your question? Submit a suggested question below under "Feedback".

## **Has Windows 10 IoT replaced embedded?**

Windows 10 IoT is the replacement for the Windows Embedded product lines.

## **How do you support Windows 10 IoT Core? Do you have LTSB and LTSC support for 10 years, similarly to Windows 10 IoT Enterprise?**

Yes, we recently announced LTSC for Windows 10 IoT Core and Enterprise. You can read more about this announcement [here](#).

## **What will happen six months after I've signed the agreement?**

You will need to come back and sign the agreement again. If you'd prefer not to manage this, you can sign an agreement with a distributor, where the agreement renews differently and automatically. You can learn more about distributors [here](#).

## **Do my devices have to update automatically?**

There is not a requirement that a device be connected to the internet. The license does require that you provide a mechanism to provide an update to the device if needed, but there are non-internet connected methods of delivering updates. You can learn more [here](#).

## **How can I use Cortana for my Windows 10 IoT Core device?**

You can leverage the Cortana Devices SDK, which can be found [here](#).

## **Do I have to use a specific version of Visual Studio when developing with Windows 10 IoT Core?**

No, you can use any version of Visual Studio you'd like.

## **Where can I get extra in-person help for prototyping and commercializing?**

Microsoft's IoT & AI Insider Labs are facilities to help you develop, prototype and test IoT and AI solutions for market commercialization. Located in Redmond, USA; Munich, Germany; and Shenzhen, China; they are located at the heart of Microsoft's product group, industrial and manufacturing solutions. You can learn more about these labs [here](#).

**TIP**

For troubleshooting issues related to development, please read the [Troubleshooting doc](#) in our developer documentation.

# Troubleshooting

Having worked with a number of people, teams, and companies interested in commercializing, the Windows 10 IoT team decided to publish learnings from troubleshooting different issues. To find something specific, use Ctrl+F to find a word or phrase. Have any insight you want to add? Create a PR for this documentation or provide content feedback below.

## Identifying SKU

`GetProductInfo` API identifies the IoT Core SKU/Edition information. Edition/Product ID will be baked into the image.

`PRODUCT_IOTUAP: 0x0000007B` = Windows 10 IoT Core

You can find more information about edition IDs [here](#).

## Mapping hardware vkeys on Windows 10 IoT Core

Switch to reporting as a HID keyboard and sending VKs instead of consumer controls.

## Minimizing memory allocation

If you want to minimize OS controlled memory and processing for display, make sure that the OS is clear and that no display is attached. Make sure the Intel INF settings are not faking a display when none is attached and configure any outputs to be external DisplayPort. This should result in DWM running but acting as if the monitor is idle.

## Retrieving lots and getting crash dumps

For UWP apps specifically:

1. Get your app store-signed and you'll receive telemetry through the app portal on Dev Center. You will receive call stacks, but not full dumps.
2. You can use [`WerRegisterAppLocalDump`](#) to get your app to dump logs and then you can have the app upload them to wherever you like.
3. Additionally, you can instrument even further with VSAppCenter or [HockeyApp](#).

For system issues (NTServices, OS stability or driver):

1. We are working on a Partner Telemetry Insights portal to enable customers to receive call stacks and info on crashing services, OS components, and drivers.
2. You can have a custom script or exe to harvest dmp files from the device.
3. You can use Azure IoT DM to configure ETW tracing on the device as required and [capture logs](#).
4. You can build a custom exe to call into wevtapi.dll and record the last shutdown reason.

## Running SLEEPSTUDY

If you run into the error 0x080004005 while trying to run SLEEPSTUDY in Windows 10 IoT Core, you will need to do the following to generate a SLEEPSTUDY report:

1. From SSH/PowerShell, run this command on IoT COrer; `powercfg / sleepstudy / xml`
2. This will generate a "sleepstudy-report.xml".
3. Copy the generated sleepstudy-report.xml report to a desktop machine and run  
`"powercfg/sleepstudy/transformxml sleepstudy-report.xml"`
4. The final sleepstudy-report.html will be generated.

### Servicing apps and dealing with NTServices

Fast app iterations can be done with the App Store or using Azure Blob. With the app store, you do not have to pay for the CDN and egrees. There is also worldwide CDN coverage for free with the store. It is a good rule of thumb to get a Store ID and Store Update ready when shipping for a fast app update.

### Setting Bluetooth Class of Device to "not a PC"

For good interop and Bluetooth compatibility, it's important that the Bluetooth stack indicates a proper class of device (COD). The various values are defined by the standard are [here](#). By default, Windows reads the form factor data from the SmBios enclosure type value (See table 17 on page 38) and derives the Major Device Class and Minor Device Class fields of the CoD from that. If a platform wants to override the default COD assigned via the enclosure type, it can do so by setting the "COD Major" and "COD Type" values documented [here](#).

### Setting the computer name

IoT has a built-in Win32 tool called "SetComputerName". This can be called to change the computername. A reboot is required to change the name. Unfortunately, this utility must be called from the administrator context and UWP apps run in the DefaultAccount user context. One way to workaround this is to run a batch file from a manually triggered scheduled task that calls SetComputerName. The batch file, will need to read the desired computername from a text file that your main UWP app will save the name into. For example, the UWP app could could save a file to the public documents folder and the batch file would read it from there. The UWP app would then need to use the process launcher (to trigger the scheduled task to run (e.g. schtasks /run /TN "SetMyComputerNameTask") Read processlauncher docs [here](#) Read schTasks docs [here](#)

# Create Windows Universal OEM Packages

1/18/2019 • 5 minutes to read

The Windows Universal OEM packaging standard is supported on Windows IoT Core, version 1709.

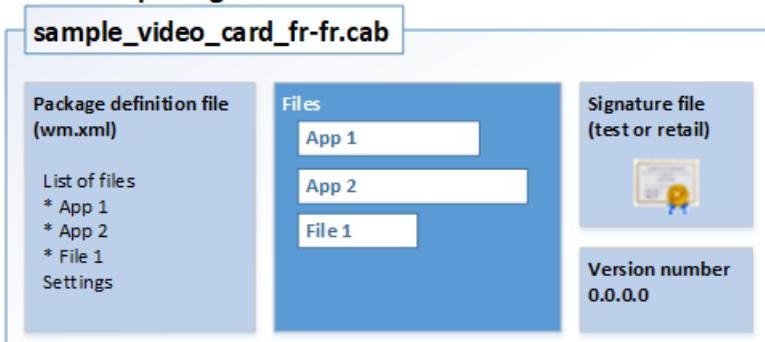
This new packaging schema is built to be compatible with more types of devices in the future. If you've built packages for IoT Core devices using the legacy packaging standard (pkg.xml), and you'd like to use them on IoT devices, you can [convert them to the new packaging standard](#).

## Packages

Packages are the logical building blocks used to create IoT Core images.

- **Everything you add is packaged.** Every driver, library, registry setting, system file, and customization that you add to the device is included in a package. The contents and location of each item are listed in a package definition file (\*.wm.xml).
- **Packages can be updated by trusted partners.** Every package on your device is signed by you or a trusted partner. This allows OEMs, ODMs, developers, and Microsoft work together to help deliver security and feature updates to your devices without stomping on each other's work.
- **Packages are versioned.** This helps make updates easier and makes system restores more reliable.

### Windows package



Packages fall into three main categories:

- **OS kit packages** contain the core Windows operating system
- **SoC vendor prebuilt packages** contain drivers and firmware that support the chipset
- **OEM packages** contain device-specific drivers and customizations

[Learn about how to combine these packages into images for devices.](#)

## Start by creating a new empty package

1. Install Windows ADK for Windows 10, version 1709, as well as the other tools and test certificates described in [Get the tools needed to customize Windows IoT Core](#) and [Lab 1a: Create a basic image](#).
2. Use a text editor to create a new package definition file (also called a Windows Manifest file) based on the following template. Save the file using the **wm.xml** extension.

```

<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<identity
  xmlns="urn:Microsoft.CompPlat/ManifestSchema.v1.00"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  name="MediaService"
  namespace="Media"
  owner="OEM"
  >
</identity>

```

3. Create the empty package file (\*.cab). The filename created is based on the owner, namespace, and name from the file.

```

c:\oemsample>pkggen myPackage.wm.xml /universalbsp

Directory of c:\oemsample

04/03/2017  05:56 PM    <DIR>      .
04/03/2017  05:56 PM    <DIR>      ..
04/03/2017  05:43 PM          333 myPackage.wm.xml
04/03/2017  05:56 PM        8,239 OEM-Media-MediaService.cab

```

## Add content to a package

The contents of a package are organized as a list of XML elements in the package definition file.

The following example demonstrates how to add some files and registry settings to a package. This example defines a variable (\_RELEASEDIR) that can be updated each time you generate the package.

```

<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<identity
  xmlns="urn:Microsoft.CompPlat/ManifestSchema.v1.00"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  name="MediaService"
  namespace="Media"
  owner="OEM"
  >
<files>
  <file source="$( _RELEASEDIR )\MediaService.dll"/>
</files>
<regKeys>
  <regKey keyName="$( hk1m.software )\OEMName\MediaService">
    <regValue
      name="StringValue"
      type="REG_SZ"
      value="MediaService"
      />
    <regValue
      name="DWordValue"
      type="REG_DWORD"
      value="0x00000020"
      />
  </regKey>
</regKeys>
</identity>

```

## Run the pkggen.exe tool

PkgGen.exe [project] /universalbsp ...

```
[project]..... Full path to input file : .wm.xml, .pkg.xml, .man  
Values:<Free Text> Default=NULL

[universalbsp]..... Convert wm.xml BSP package to cab  
Values:<true | false> Default=False

[variables]..... Additional variables used in the project file,syntax:<name>=<value>;<name>=<value>;....  
Values:<Free Text> Default=NULL

[cpu]..... CPU type. Values: (x86|arm|arm64|amd64)  
Values:<Free Text> Default="arm"

[languages]..... Supported language identifier list, separated by ';'   
Values:<Free Text> Default=NULL

[version]..... Version string in the form of <major>.<minor>.<qfe>.<build>  
Values:<Free Text> Default="1.0.0.0"

[output]..... Output directory for the CAB(s).  
Values:<Free Text> Default="CurrentDir"
```

Example:

```
c:\oemsample>pkggen myPackage.wm.xml /universalbsp /variables:"_RELEASEDIR=c:\release"
```

## Add a driver component

In the package definition file, use the **driver** element to inject drivers. We recommend using relative paths, as it's typically the simplest way to describe the INF source path.

```
<drivers>  
  <driver>  
    <inf source="$_RELEASEDIR\Media.inf"/>  
  </driver>  
</drivers>
```

If the default file import path is not equal to the INF source path, you can use the defaultImportPath attribute. In the following example, the INF is in the current directory, but the files to be imported are relative to `$_RELEASEDIR`.

```
<drivers>  
  <driver defaultImportPath="$_RELEASEDIR">  
    <inf source="Media.inf"/>  
  </driver>  
</drivers>
```

If files to be imported are not relative to how they are defined in the INF, file overrides can be applied. This is not recommended, but is available for special cases.

```

<drivers>
  <driver>
    <inf source="Media.inf"/>
    <files>
      <file name="mdr.sys" source="$_RELEASEDIR\path1\mdr.sys" />
      <file name="mdr.dll" source="$_RELEASEDIR\path2\mdr.dll" />
    </files>
  </driver>
</drivers>

```

## Add a service component

In the package definition file, use the **service** element (and its child elements and attributes) to define and package a system service.

```

<service
  dependOnService="AudioSrv;AccountProvSvc"
  description="@%SystemRoot%\system32\MediaService.dll,-201"
  displayName="@%SystemRoot%\system32\MediaService.dll,-200"
  errorControl="normal"
  imagePath="%SystemRoot%\system32\svchost.exe -k netsvcs"
  name="MediaService"
  objectName="LocalSystem"
  requiredPrivileges="SeChangeNotifyPrivilege,SeCreateGlobalPrivilege"
  sidType="unrestricted"
  start="delayedAuto"
  startAfterInstall="none"
  type="win32UserShareProcess"
>

```

## Build and Filter WOW Packages

To build Guest or WOW packages (32 bit packages to run on 64 bit devices) add the **buildWow="true"** attribute to myPackage.wm.wml

```

<identity
  xmlns="urn:Microsoft.CompPlat/ManifestSchema.v1.00"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  name="MediaService"
  namespace="Media"
  owner="OEM"
  buildWow="true"
>

```

Running PkgGen.exe with now generate one WOW package for each host package.

04/05/2017 07:59 AM	11,870 OEM-Media-MediaService.cab
04/05/2017 07:59 AM	10,021 OEM-Media-MediaService_Wow_arm64.arm.cab

Typically, the 64 bit device will get its Host 64 bit package and either its Guest 32 bit or WOW package, both generated from myPackage.wm.xml. To avoid resource conflicts between the two packages, use **build filters**:

```
<regKeys buildFilter="not build.isWow and build.arch = arm" >
  <regKey keyName="$(hk1m.software)\OEMName\MediaService">
    <regValue
      name="StringValue"
      type="REG_SZ"
      value="MediaService"
    />
  </regKey>
```

In this case, the registry keys are exclusive to the Host 32 bit ARM package. The CPU switch is used to set build.arch, and build.isWow is set by PkgGen to false when building the 32 bit Host Package, then true when building the 32 bit Guest or WOW package.

```
[cpu]..... CPU type. Values: (x86|arm|arm64|amd64)
Values:<Free Text> Default="arm"
```

## Converting Windows Universal OEM Packages

If you've created packages using the pkg.xml packaging model, and you want to use them on Windows IoT Core, version 1709, you'll need to either recreate your packages, or convert them using the pkggen.exe tool.

After you convert the packages, you may need to modify the wm.xml file to make sure that it follows the [schema](#).

IoT Core Add-ons v4.x support the new [Windows Universal OEM Packages standard \(wm.xml\)](#). This new packaging schema is built to be compatible with more types of devices in the future.

### Convert your package files

To convert your existing packages created in the legacy phone packaging format (pkg.xml) to the new wm.xml format:

```
pkggen.exe "filename.pkg.xml" /convert:pkg2wm
```

Or, from the IoTCoreShell prompt, convert using either convertpkg or buildpkg. The output wm.xml files are saved to the same folder.

```
convertpkg.cmd MyPackage.pkg.xml
buildpkg.cmd MyPackage.pkg.xml
```

Review and test the wm.xml packages with buildpkg.

```
buildpkg.cmd MyPackage.wm.xml
```

After you've converted the files to wm.xml format, it's safe to delete the pkg.xml files.

### Regenerate your app packages

Use the newAppxPkg with the same component name. This regenerates the customizations.xml file. The version number of the appx is retained as the version number for ppkg.

```
newAppxPkg
"C:\DefaultApp\IoTCoreDefaultApp_1.2.0.0_ARM_Debug_Test\IoTCoreDefaultApp_1.2.0.0_ARM_Debug_Test.appx" fga
Appx.MyUWPApp
```

Learn more: [Add apps](#).

## **Adding files: watch out for zero-sized files, relative paths**

Zero-sized files are not supported in `wm.xml`. To work around this, add an empty space in the file, making it non-zero size file.

Paths: When you're adding files that are in the current directory, you'll need to explicitly add the `.\` prefix to the file name.

```
<BinaryPartition ImageSource=".\\uefi.mbn" />
```

Learn more: [Add files](#)

## **Update your provisioning package `customization.xml` file**

In ADK version 1709, you'll need to update the `customizations.xml` file:

In your `product\\prov` folder, manually move `Common/ApplicationManagement` to `Common/Policies/ApplicationManagement`

```
<Customizations>
  <Common>
    <Policies>
      <ApplicationManagement>
        <AllowAppStoreAutoUpdate>Allowed</AllowAppStoreAutoUpdate>
        <AllowAllTrustedApps>Yes</AllowAllTrustedApps>
      </ApplicationManagement>
    </Policies>
  </Common>
</Customizations>
```

Provisioning packages (PPKG) now support four-part versioning similar to the package versioning. So with this change, version 1.19 > 1.2. Previous versions used character-based sorting, so version 1.19 was considered earlier than 1.2.

Learn more: [Add provisioning files](#)

# Windows Universal OEM Package Schema

1/18/2019 • 2 minutes to read

You can manually edit your packages using the Universal OEM Package Schema.

## [Creating Windows Universal OEM Packages](#)

## Schema

Only the common elements and attributes are documented here.

To get the full schema run "pkggen /universalbsp /wmxsd:", then open **WM0.XSD** with Visual Studio.

### **identity**

ATTRIBUTE	TYPE	REQUIRED	MACRO	NOTES
owner	string	*		
name	string	*	*	
namespace	string		*	
buildWow	boolean			Default = false, set to true to generate WOW packages
legacyName	string		*	Uses the specified name as the package name overriding the default name (owner-namespace-name.cab).

```
<identity name="FeatureName" namespace="FeatureArea" owner="OEM" buildWow="false"/>
```

### **onecorePackageInfo**

ATTRIBUTE	TYPE	REQUIRED	MACRO	NOTES
targetPartition	MainOS Data UpdateOS EFIESP PLAT	*		If onecorePackageInfo is not specified, Default = MainOS
releaseType	Production Test			If onecorePackageInfo is not specified, Default = Production

```
<onecorePackageInfo targetPartition="MainOS" releaseType="Production"/>
```

### **file**

ATTRIBUTE	TYPE	REQUIRED	MACRO	NOTES
source	string	*	*	
destinationDir	string		*	destinationDir must start with one of the following built in runtime macros below.
name	string			used to rename the source file
buildFilter	string			

destinationDir must start with:

- \$(runtime.bootDrive)
- \$(runtime.systemDrive)
- \$(runtime.systemRoot)
- \$(runtime.windows)
- \$(runtime.system32)
- \$(runtime.system)
- \$(runtime.drivers)
- \$(runtime.help)
- \$(runtime.inf)
- \$(runtime.fonts)
- \$(runtime.wbem)
- \$(runtime.appPatch)
- \$(runtime.sysWow64)
- \$(runtime.mui)
- \$(runtime.commonFiles)
- \$(runtime.commonFilesX86)
- \$(runtime.programFiles)
- \$(runtime.programFilesX86)
- \$(runtime.programData)
- \$(runtime.userProfile)
- \$(runtime.startMenu)
- \$(runtime.documentSettings)
- \$(runtime.sharedData)
- \$(runtime.apps)
- \$(runtime.clipAppLicenseInstall)
- If not specified, the default is \$(runtime.system32)

To see the directories that map to these locations, see C:\Program Files (x86)\Windows Kits\10\tools\bin\i386\pkggen.cfg.xml.

```
<file buildFilter="(not build.isWow) and (build.arch = arm)" name="output.dll"
source="$_RELEASEDIR\input.dll" destinationDir="$(runtime.system32)"/>
```

## regKey

ATTRIBUTE	TYPE	REQUIRED	MACRO	NOTES
keyName	string	*	*	keyName must start with \$(hkLM.system), \$(hkLM.software), \$(hkLM.hardware), \$(hkLM.sam), \$(hkLM.security), \$(hkLM.bcd), \$(hkLM.drivers), \$(hkLM.svchost), \$(hkLM.policies), \$(hkLM.microsoft), \$(hkLM.windows), \$(hkLM.windowsnt), \$(hkLM.currentcontrolset), \$(hkLM.services), \$(hkLM.control), \$(hkLM.autologger), \$(hkLM.enum), \$(hkcr.root), \$(hkcr.classes), \$(hkcu.root), \$(hkuser.default)
buildFilter	string			

To see the registry keys that map to these locations, see C:\Program Files (x86)\Windows Kits\10\tools\bin\i386\pkgen.cfg.xml.

```
<regKey buildFilter="buildFilter1" keyName="keyName1">
  <regValue buildFilter="buildFilter1" name="name1" value="value1" type="REG_SZ" />
</regKey>
```

## regValue

ATTRIBUTE	TYPE	REQUIRED	MACRO	NOTES
name	string			Name of the value you are specifying. If not specified, the Default value in the key will be overwritten
type	string	*		type must be one of these: REG_SZ, REG_MULTI_SZ, REG_DWORD, REG_QWORD, REG_BINARY, REG_EXPAND_SZ
value	string			
buildFilter	string			

```
<regKey buildFilter="buildFilter1" keyName="keyName1">
<regValue buildFilter="buildFilter1" name="name1" value="value1" type="REG_SZ" />
<regValue buildFilter="buildFilter2" name="name2" value="value1,value2" type="REG_MULTI_SZ" />
<regValue buildFilter="buildFilter3" name="name3" value="00000000FFFFFFFF" type="REG_QWORD" />
<regValue buildFilter="buildFilter4" name="name4" value="FFFFFFFF" type="REG_DWORD" />
<regValue buildFilter="buildFilter5" name="name5" value="0AFB2" type="REG_BINARY" />
<regValue buildFilter="buildFilter6" name="name6"
value=""%ProgramFiles%\MediaPlayer\wmplayer.exe"" type="REG_EXPAND_SZ" />
</regKey>
```