Windows UEFI Firmware Update Platform

April 4, 2013

Abstract

To provide a more consistent, reliable firmware update experience and improve discoverability of important system firmware updates for end-users, Windows 8 supports a platform for installing system and device firmware updates via driver packages. You can read this document to learn how the system firmware update feature of Windows 8 works.

This information applies to the following operating systems:   
 Windows 8

References and resources discussed here are listed at the end of this paper.

The current version of this paper is maintained on the Web at:   
 [Windows UEFI Firmware Update Platform](http://msdn.microsoft.com/en-us/windows/hardware/jj248726)

**Disclaimer**: This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet website references, may change without notice. Some information relates to pre-released product which may be substantially modified before it’s commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2013 Microsoft. All rights reserved.

C:\Users\jenlin\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Outlook\KN5ONHWU\dep_MicrosoftLogotype.png

Contents

[Overview 4](#_Toc352764413)

[Design Summary 4](#_Toc352764414)

[System firmware Updates 4](#_Toc352764415)

[Device Firmware Updates 4](#_Toc352764416)

[System Requirements for Windows Firmware Update 5](#_Toc352764417)

[System and Device Firmware Update via firmware driver package 6](#_Toc352764418)

[Populating the ESRT Table 7](#_Toc352764419)

[Customizing Firmware for Different Geographic Regions and Resale Channels 8](#_Toc352764420)

[Authoring a Firmware Update Package 9](#_Toc352764421)

[Certifying and Signing the Update Package 12](#_Toc352764422)

[Installing the Update 12](#_Toc352764423)

[UEFI Firmware Update Implementation Details 13](#_Toc352764424)

[ESRT Table Definition 13](#_Toc352764425)

[Plug and Play Device 17](#_Toc352764426)

[Authoring an Update Driver Package 17](#_Toc352764427)

[Signing the Firmware Driver Package 23](#_Toc352764428)

[Validating the status of the firmware update 26](#_Toc352764429)

[Processing Updates 26](#_Toc352764430)

[Device I/O From the UEFI Environment 28](#_Toc352764431)

[Seamless Crisis Prevention & Recovery 28](#_Toc352764432)

[Pre-Installation Criteria 29](#_Toc352764433)

[Post-Installation Criteria 30](#_Toc352764434)

[Recovering from install and boot failures 30](#_Toc352764435)

[Firmware update unsuccessful 30](#_Toc352764436)

[Firmware update succeeds, but Windows fails to boot 31](#_Toc352764437)

[Firmware Update Status 31](#_Toc352764438)

[Push-Button Reset 33](#_Toc352764439)

[Rolling Back Firmware Updates 33](#_Toc352764440)

[UEFI Firmware Update User Experience 34](#_Toc352764441)

[User Experience 34](#_Toc352764442)

[Time Frame 35](#_Toc352764443)

[Firmware Update Boot Screen Components 36](#_Toc352764444)

[Firmware Update Boot Screen Component A: OEM Logo 36](#_Toc352764445)

[Firmware Update Boot Screen Component B: Update Text 36](#_Toc352764446)

[Windows Firmware Update Display Capsule 37](#_Toc352764447)

[Links & Appendices 40](#_Toc352764448)

[Appendix A: UEFI System Resource Table (ESRT) Detailed Description 41](#_Toc352764449)

[Identifying the ESRT table pointer 41](#_Toc352764450)

[UEFI System Resource Table Definition 42](#_Toc352764451)

[ESRT Table Header 42](#_Toc352764452)

[Firmware Resource Entry Array 43](#_Toc352764453)

[Appendix B: Firmware update signing process and requirements 44](#_Toc352764454)

[Overview 44](#_Toc352764455)

[Signing the Updated Firmware 45](#_Toc352764456)

[Signing the Capsule 45](#_Toc352764457)

[Signing the Firmware Update Package 46](#_Toc352764458)

# Overview

Windows historically has not provided any facility for servicing (or updating) system or device firmware. If an IHV or OEM wanted to update firmware, it was their responsibility to not only create and test the firmware, but also to design and develop a system for applying the updated firmware to a PC. Without system or device firmware update support in Windows there was no clear, consistent method for performing such updates.

# Design Summary

There are two types of firmware that can be serviced via Windows: system firmware and device firmware. Where system firmware is responsible for providing critical boot and runtime services to the system as a whole, device firmware is associated with a particular device integrated into a system. Such device firmware typically works together with a device driver, allowing the OS to expose the device to OS-level services and applications.

## System firmware Updates

System firmware updates for Windows 8 UEFI-based systems will be deployed as device driver packages (INFs). Windows will use information provided by the platform to ensure that the update package only applies to appropriate systems. A firmware update package contains a binary file containing the system firmware image. Once the firmware update package is on the end-user’s system, Windows will use the UEFI UpdateCapsule() function to hand-off the firmware payload to the platform firmware for processing.

Deploying the update as a driver package allows the firmware update process to align with many existing deployment and servicing tools, and ensures simple update package authoring for hardware vendors.

Note: The fact that the firmware update is delivered as a driver package does not mean that the update is written as an actual driver. The driver package will contain an INF file and a binary file containing the system or device firmware image.

## Device Firmware Updates

For the purposes of updating device firmware, the device firmware can be assigned to one of these two categories:

1. **UEFI-updatable Device Firmware**

This device firmware can be updated using a device driver package leveraging the same mechanism as system firmware. A device firmware update is distributed as a firmware update package. Once the firmware update package is on the end-user’s system, Windows will use the UEFI UpdateCapsule() function to hand-off the device firmware payload to the platform firmware for processing. This process is virtually identical to how Windows hands off system firmware update payload, and is discussed below.

It is recommended that device firmware be updated using a discrete firmware update driver package, but device firmware may also be updated with system firmware as part of a single firmware update driver package.

**Note**UEFI should not be used to update peripheral devices. UEFI requires devices to be present during reboot to apply a firmware update which cannot be guaranteed with (external, removable) peripheral devices.

1. **Driver-updatable Device Firmware**

This device firmware can be updated by the device driver during the normal Windows OS runtime. Updating device firmware using normal Windows OS drivers is not covered by this paper.

### System Requirements for Windows Firmware Update

In order for a system to be compatible with the Windows firmware updating mechanism, it must meet the following requirements:

* **System must have UEFI UpdateCapsule(), QueryCapsuleCapabilities() functionality implemented**
  + UpdateCapsule() will be used as the mechanism for passing firmware update payload between Windows and the platform firmware.
  + Defined in UEFI specification.
* **Platform firmware supports firmware updates initiated by Windows**
  + System firmware, and some classes of device firmware, must be updatable using this process.
  + Firmware code recognizes a firmware update payload passed to UpdateCapsule and initiates the update process.
  + Implementation owned by partner.
* **Must specify a Firmware Resource in the EFI System Resource Table (ESRT)**
  + The Firmware Resource will allow Windows to surface a device instance with a Hardware ID, which will be used to target the system or device firmware update to appropriate systems and devices. It also describes the current firmware version and provides status for previous updates.
  + There exists a single entry for system firmware updates.
  + All devices with updateable firmware must have a resource specified in the ESRT.
    - Except if a device’s firmware is updated as part of a system firmware update.
  + Refer to section 3.1 ESRT Table Definition for additional information.

## System and Device Firmware Update via firmware driver package



Figure 1: System and Device Firmware Update Process

Deploying a firmware update using a firmware driver package follows a relatively simple process that can be divided into three phases:

1. Authoring a firmware update package
2. Certifying and signing the update package
3. Installing the update

This process assumes that the UEFI firmware update payload has already been developed, tested, and signed.

1. The firmware driver package simply contains the payload for a firmware update and allows the firmware update payload to be distributed in the same manner as all Windows drivers.
2. Once the driver package has been deployed to a system, the firmware update payload is passed to platform firmware via the UEFI UpdateCapsule() service.
3. Upon receipt of the firmware update payload, platform firmware recognizes the payload and applies the update.
4. The implementation of the platform firmware update code is proprietary, as is the format of the firmware update payload.

A device driver package contains an INF file describing the devices to which the package applies. A firmware driver package is the same. Devices and system firmware resources supporting this update mechanism must uniquely identify themselves to bind to a firmware driver package. The next section describes the identification mechanism.

### Populating the ESRT Table

The EFI System Resource Table (ESRT) provides a mechanism for identifying integrated device and system firmware resources for the purposes of targeting firmware updates to those resources. Each entry in the ESRT describes a device or system firmware resource that can be targeted by a firmware update driver package. Each firmware resource that can be updated by a firmware update driver package must be described by exactly one entry in the ESRT to enable firmware updates to be deployed and installed. Complete details on the layout and implementation of the ESRT can be found in the UEFI Firmware Update Implementation Details section.



Figure 2: Updatable Firmware on a SoC System

Figure 2 shows a high level block diagram of a typical SoC system. Each system device containing updatable firmware is represented by a single block in this example. Each block is capable of receiving and installing a targeted, independent firmware update for the device. As such, each block has a unique entry in the ESRT representing that device. It is also possible for a device to have its firmware updated as part of a single, monolithic system firmware update driver package. In this case, the device would not have an ESRT entry since it is updated with the system firmware. More generally, a device can only have its firmware update targeted by one entry in the ESRT.

Figure 3: SoC System Firmware Resources

For simplicity, the figure describes the model where each device has its firmware update targeted separately with a unique entry. Each GUID in the table identifies an updatable device or the UEFI system firmware within this SoC system. Each GUID in the table is unique (i.e. no two devices/system firmware share the same GUID value) and the table is unique to a single SoC system. Hardware revisions of a SoC system must define new GUID values for devices/system firmware. This ensures that firmware is targetable to each component in the revised hardware, as subtle differences in device hardware across revisions may require different firmware.

### Customizing Firmware for Different Geographic Regions and Resale Channels

Systems will be sold in a variety of markets and geographies worldwide. To enable this, OEMs must define unique GUID values for those devices/system firmware which may require region-specific firmware. For example, region-specific firmware is frequently required for the Mobile Broadband (MBB) device. MBB device firmware is often customized for a specific Mobile Network Operator (MNO) in a particular region, to comply with local MNO and government regulations. To allow targeting of firmware to such devices, the OEM must assign a unique GUID to that device in the ESRT at the time of manufacture.

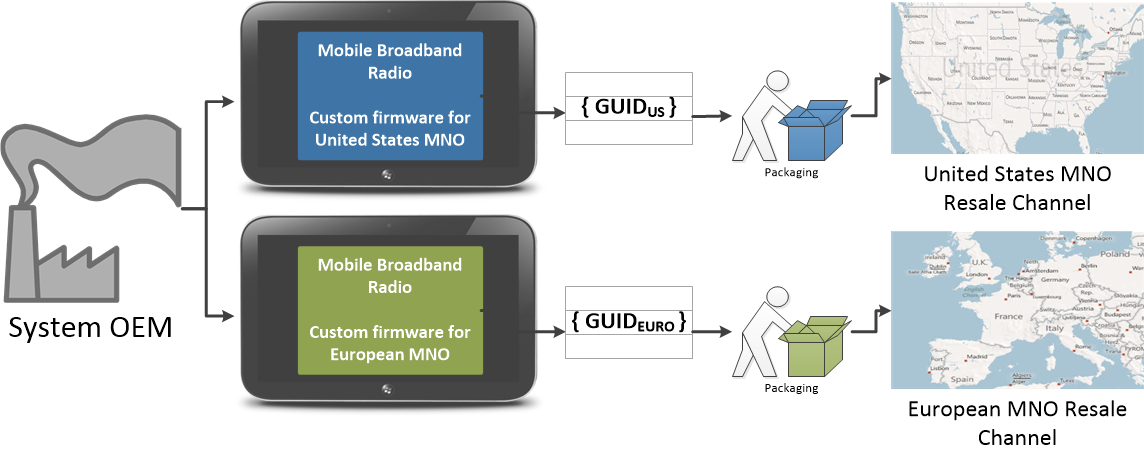


Figure 4: Two Hardware-Identical SoC Systems Destined for Different Geographic Locales

Observe in Figure 4 that the system is identical in all respects, with the exception that the systems are destined for resale in different geographies. Therefore, the MBB device firmware in each system must be independently targetable and assigned a different GUID in the ESRT. This enables the MNO to target firmware updates to the system that is sold by them in their operating area. Similar consideration must be given to any device which may require custom firmware by geography or resale channel.

### Authoring a Firmware Update Package

Each firmware update package will contain a single binary file containing the entire firmware payload (for example *firmware.bin*) and a security catalog Windows will use to validate *firmware.bin* (see the [Links](#_Links) section for more information on security catalogs and drivers).

Firmware update packages must be capable of updating one or more of the following types of firmware:

* The UEFI system firmware.
* The firmware for a single device in the system.

It is recommended that each firmware update package target a single firmware resource (UEFI system firmware or a single device), but there may be circumstances where it is advantageous to have a single firmware update package that updates both system firmware and one or more devices.

**Note**A device cannot be targeted by more than one firmware update package. If a device is targeted by a firmware update package which also includes system firmware, it cannot be targeted by a second firmware update package which only targets the device.

1. To allow a firmware update package to target a firmware update to the appropriate system hardware, Windows surfaces a device instance for each entry in the ESRT, where such a device instance exposes a Hardware ID that identifies it as belonging to the ESRT entry.
2. When a firmware update package is installed, it is processed by Windows as a driver package. Windows will copy the firmware payload of each update package to a safe location under the System directory, prepare the system to perform the firmware updates and trigger the system to restart.

**Note**   Windows does not support dependencies between driver packages. Therefore, the following requirements must be observed when authoring a new firmware update package:

* A firmware update package must be capable of successful installation on its own and without dependency on other device firmware, system firmware, or other firmware update packages.
* It is recommended that each update package is targeted to a single device on the system or to the UEFI system firmware (defined in the ESRT).
* Each update package must contain a single firmware update binary (for example *firmware.bin*).

1. The firmware update payload in each update package needs to be contained in a single binary file. Upon system reboot, the OS loader loads each firmware update binary file for each firmware update package into physical memory and builds an array of pointers to each payload file provisioned for installation (the UEFI 2.3.1 specification refers to this array as the CapsuleHeaderArray).
2. This array is passed in the call to the EFI UpdateCapsule() function. UpdateCapsule() is used as a mailbox, passing each driver package’s firmware update payload to the platform firmware.
3. Each capsule (a firmware update payload) is identified by the firmware ID specified by the ESRT entry for a firmware resource.
4. Upon receipt of each firmware update payload, the firmware update request is processed and applied when applicable.

**Note**   Each entry in the CapsuleHeaderArray is a single, contiguous block of data containing the firmware update payload from a firmware driver package for a single device in the system. For each targeted firmware resource, the firmware update payload must contain the firmware image and all information required by the platform for validation.

* The firmware payload for all firmware update driver packages are passed to platform firmware through the UEFI UpdateCapsule service. Since integrated devices will be sourced from a variety of different IHVs, the system OEM (and possibly the SoC manufacturer) will need to work directly with these IHVs to ensure device firmware updates are authored appropriately for the given system. Additionally, the system OEM needs to ensure that the ESRT entries allow UpdateCapsule packages to be targeted to the appropriate systems.
* For example, several OEMs might choose the same model Mobile Broadband (MBB) device for their systems. Even though the MBB device is identical in each system, each OEM must collaborate with the MBB IHV to author a firmware update package customized for their system. This level of customization of the device firmware update is necessary to address variables across OEM systems:
* Addressing the device may differ based on the SoC chosen by the OEM and how the device is connected to the SoC.
* The system OEM may sell the system to multiple Mobile Network Operators (MNOs) for resale to consumers. The MBB device must be MNO-aware, requiring the firmware to be both customized and certified to a particular MNO’s requirements.
* The system may be sold in multiple markets worldwide, each with different RF regulations and radio frequency assignments. The MBB device firmware may require customization to meet these market requirements.
* Each OEM must carefully consider such device-specific requirements, and take necessary steps to ensure that device firmware can be targeted and updated appropriately. This requires careful management of ESRT entries to ensure device firmware can be properly deployed.

1. After the update package is authored, it needs to be submitted to Microsoft for certification and signing.

### Certifying and Signing the Update Package

Since the firmware update is delivered as a driver package, it will need to go through all of the same verification and signing processes as a regular driver package. The driver package will need to pass Windows Hardware Certification Kit (WHCK) tests, and will need to be submitted to the Windows Dev Center Hardware Dashboard for signing. Once signed, the driver package will be distributed back to the submitter.

**Note**: Signing of the driver package is different from signing the UEFI firmware or device firmware itself. The signature on the driver package, delivered via security catalog, is used by Windows to verify the integrity of firmware.bin before handing it to the UEFI. Windows does not provide the security catalog to the firmware. The signature on the UEFI firmware or device firmware update is validated by the platform firmware, and is not checked by Windows. The IHV/OEM is responsible for ensuring the integrity and security of the firmware through signature verification, encryption or other means. Remember that that UEFI Secure Boot requires properly signed executable images.

### Installing the Update

Firmware update packages can be installed using any tool that installs Windows drivers. The installation process will copy the firmware update payload (*firmware.bin*) to a well-known system directory, and will create the registry keys necessary to tell Windows a new update is available. Once the installation is finished, a reboot will be required in order to trigger the actual firmware update process.

 Figure 5: Update Install Process

During the next boot, and before ExitBootServices() has been called, the OS Loader will check the well-known registry key locations to determine if new firmware update payload is available. If new update payload is available, the OS Loader will verify the hash of *firmware.bin* against the security catalog delivered with the driver package. If the signature is valid, the *firmware.bin* will be handed off to the platform firmware via the UEFI UpdateCapsule() service.

**Important**At this point, the platform firmware is solely responsible for completing the firmware update.

If multiple firmware update packages are installed, the OS Loader will call UEFI UpdateCapsule() with the payload of each available update. Each firmware payload will be a separate capsule, each identified by the GUID of the ESRT entry for the targeted firmware update package.

The EFI System Resource Table provides the current firmware version and the status of the last update attempted. The OS Loader will use this information to assess whether the update was successfully applied. The firmware status information will be persisted into the OS such that it is available to a firmware update application running in Windows. Finally, the OS Loader will continue the boot process, eventually allowing the user to logon.

# UEFI Firmware Update Implementation Details

This section describes the steps necessary for a firmware vendor to implement support for update capsule firmware updates.

## ESRT Table Definition

The following is the table format of the ESRT and Firmware Resource Entries.

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Byte Length | Byte Offset | Description |
| Firmware Resource Count | 4 | 0 | The number of firmware resources, must not be zero. |
| Firmware Resource Maximum | 4 | 4 | The maximum number of resource array entries that can be added without reallocating the table, must not be zero. |
| Firmware Resource Version | 8 | 8 | The firmware resource entry version. |
| Firmware Resource Entry Array |  |  | Firmware Resource Entry 0 |
| Firmware Class | 16 | 16 | This GUID defines the class of systems for which an update gets applied. |
| Firmware Type | 4 | 32 | Identifies the type of firmware resource. |
| Firmware Version | 4 | 36 | The current firmware version, where a larger number represents a newer release. The value is not defined but should incorporate version major and minor numbers. Recommended format is first word is major and second word is minor version numbers. |
| Lowest Supported Firmware Version | 4 | 40 | The lowest firmware version that can be rolled back to, i.e. the last firmware version that contained a security fix. |
| Capsule Flags | 4 | 44 | The OS will set the upper capsule flags defined by the UEFI spec, but the low 16-bits are defined per capsule. |
| Last Attempt Version | 4 | 48 | Version of the last firmware update attempt. |
| Last Attempt Status | 4 | 52 | Status of the last firmware update attempt. |
| … |  |  | Firmware Resource Entry 1 |

Table 1: ESRT Table and Firmware Resource Entry Definitions

Core UEFI firmware should allocate and populate an ESRT configuration table containing one system resource entry for itself (system firmware). For illustrative purposes, in this guide core firmware will also create one additional entry representing a device that supports device firmware update using the firmware update package mechanism.

There must always be exactly one entry describing system firmware. This entry is used to target a system firmware update. If an implementation performs system and device firmware updates as a single, monolithic operation, the system firmware entry must be used to target the update. In all other cases, device firmware updates are targeted by an ESRT entry describing device firmware.

The first step then is to generate GUIDs to represent these two firmware resources, i.e., {SYSTEM\_FIRMWARE} and {DEVICE\_FIRMWARE}. Table 2 shows an example of a table definition. This example assumes both firmware versions are currently version 1 (Firmware Version == 1):

|  |  |  |
| --- | --- | --- |
| Field | Value | Comment |
| Firmware Resource Count | 2 | This table contains two firmware resource entries. |
| Firmware Resource Maximum | 2 | This table allocation contains enough space to describe a maximum of two resources. |
| Firmware Resource Version | 1 | The firmware resource entry format version this table uses is one. |
| Firmware Resource Entry Array | Firmware Resource Entry 0 |  |
| Firmware Class | {SYSTEM\_FIRMWARE} | This GUID identifies the system firmware for update via PnP. |
| Firmware Type | 1 | System firmware type is one. |
| Firmware Version | 1 | The current system firmware version is one. |
| Lowest Supported Firmware Version | 1 | The lowest supported firmware version is 1, so firmware cannot be rolled-back to a version earlier than version 1. |
| Capsule Flags | 0 | System firmware doesn’t define any private capsule update flags. |
| Last Attempt Version | 1 | The last system firmware version for which an update was attempted was one. |
| Last Attempt Status | 0 | The last system firmware update attempt was successful. |
|  | Firmware Resource Entry 1 |  |
| Firmware Class | {DEVICE\_FIRMWARE} | This GUID identifies the device firmware for update via PnP. |
| Firmware Type | 2 | Device firmware type is two. |
| Firmware Version | 1 | The current device firmware version is one. |
| Lowest Supported Firmware Version | 1 | The lowest supported firmware version is 1, so firmware cannot be rolled-back to a version earlier than version 1. |
| Capsule Flags | 0x8010 | Device firmware defines private capsule update flags (0x8010). |
| Last Attempt Version | 1 | The last device firmware version for which an update was attempted is one. |
| Last Attempt Status | 0 | The last device firmware update attempt was successful. |

Table 2: ESRT Example Showing Two Firmware Resource Entries

The above ESRT example will be used to walk through the firmware update process and describe Windows support for the update process as well as a supporting firmware implementation.

## Plug and Play Device

The presence of an ESRT configuration table will direct Windows to enumerate a separate PnP device instance for each firmware resource. For driver matching purposes, a firmware resource device is uniquely identified by its hardware IDs, which embed the Firmware ID GUID. Referring to the ESRT example in Table 2, the corresponding device instances are enumerated:

|  |  |
| --- | --- |
| Device Instance ID | Hardware IDs |
| UEFI\RES\_{SYSTEM\_FIRMWARE}\0 | UEFI\RES\_{SYSTEM\_FIRMWARE}&REV\_1  UEFI\RES\_{SYSTEM\_FIRMWARE} |
| UEFI\RES\_{DEVICE\_FIRMWARE}\0 | UEFI\RES\_{DEVICE\_FIRMWARE}&REV\_1  UEFI\RES\_{DEVICE\_FIRMWARE} |

Table 3: Enumerated Device Instances Based on ESRT Entries

Notice that two hardware IDs are reported by each firmware resource device; the first hardware ID includes the current firmware resource version, while the second one does not. Since the firmware resource version is expected to change as a result of applying a firmware update, it is important that a driver be targeted for the second un-versioned hardware ID so that it can be applicable for installation across all firmware resource versions, no matter which version is currently present on a given system.

## Authoring an Update Driver Package

It is required that the update payload for each firmware resource described in the ESRT be bundled and distributed in its own driver package so as to allow it to maintain its own versioning scheme without being tied to other firmware resource updates that may not be updated at the same cadence.

The following example provides a sample driver package INF file definition for a firmware resource update that targets the {SYSTEM\_FIRMWARE} resource from the ESRT example in Table 2, updating it from version 1 to version 2. For reference purposes, let’s assume that the GUID assigned for the SYSTEM\_FIRMWARE resource is 6bd4efb9-23cc-4b4a-ac37-016517413e9a:

[Version]

Signature = "$WINDOWS NT$"

Provider = %Provider%

Class = Firmware

ClassGuid = {f2e7dd72-6468-4e36-b6f1-6488f42c1b52}

DriverVer = 01/01/2012,2.0.0.0

CatalogFile = catalog.cat

PnpLockdown = 1

[Manufacturer]

%MfgName% = Firmware,NTarm

[Firmware.NTarm]

%FirmwareDesc% = Firmware\_Install,UEFI\RES\_{6bd4efb9-23cc-4b4a-ac37-016517413e9a}

[Firmware\_Install.NT]

CopyFiles = Firmware\_CopyFiles

[Firmware\_CopyFiles]

*firmware.bin*

[Firmware\_Install.NT.Hw]

AddReg = Firmware\_AddReg

[Firmware\_AddReg]

HKR,,FirmwareId,,{6bd4efb9-23cc-4b4a-ac37-016517413e9a}

HKR,,FirmwareVersion,%REG\_DWORD%,0x00000002

HKR,,FirmwareFilename,,{6bd4efb9-23cc-4b4a-ac37-016517413e9a}\*firmware.bin*

[SourceDisksNames]

1 = %DiskName%

[SourceDisksFiles]

*firmware.bin* = 1

[DestinationDirs]

DefaultDestDir = %DIRID\_WINDOWS%,Firmware\{6bd4efb9-23cc-4b4a-ac37-016517413e9a}

[Strings]

; localizable

Provider = "Contoso Ltd."

MfgName = "Fabrikam Inc."

FirmwareDesc = "Fabrikam System Firmware 2.0"

DiskName = "Firmware Update"

; non-localizable

DIRID\_WINDOWS = 10

REG\_DWORD = 0x00010001

Figure 6: Sample Driver Package INF File for Updating Firmware

Change the following sections in Figure 6 to customize for your setup

[Version]

DriverVer --> The date on which this driver package was authored; the Driver version of this driver package. Driver version in this driver package must be greater than the current driver version

CatalogFile --> Name of the catalog file

*firmware.bin* --> Change all instances of *firmware.bin* with the name of the firmware image name

[Manufacturer]

%MfgName% = Firmware,NT*arm*

[Firmware.NT*arm*] --> Change the architecture.

For x86, it should be NTx86

For AMD64, it should be NTamd64

[Firmware.NTarm]

%FirmwareDesc% = Firmware\_Install,UEFI\RES\_{*6bd4efb9-23cc-4b4a-ac37-016517413e9a*} --> The GUID of the firmware resource

[Firmware\_AddReg]

HKR,,FirmwareId,,{*6bd4efb9-23cc-4b4a-ac37-016517413e9a*} --> The GUID of the firmware resource

HKR,,FirmwareVersion,%REG\_DWORD%,*0x00000002* --> Version of the firmware for the update

HKR,,FirmwareFilename,,{6bd4efb9-23cc-4b4a-ac37-016517413e9a}\*firmware.bin* --> The subdirectory named after the GUID of the firmware resource and the firmware image name

[DestinationDirs]

DefaultDestDir = %DIRID\_WINDOWS%,Firmware\{6bd4efb9-23cc-4b4a-ac37-016517413e9a} --> The full destination path for the firmware image file based under a subdirectory named after the GUID of the firmware resource within the %SystemRoot%\Firmware directory

[Strings]

; localizable

Modify any strings here [optional]

Table 4 describes the various driver package INF sections and fields with reference to the above sample driver package INF file definition.

|  |  |  |
| --- | --- | --- |
| Section/Field | Value | Comment |
| **[Version]** | | INF [Version] section defines driver package versioning information. |
| Provider | %Provider% = Contoso Inc.  (localized in [Strings] section) | Identifies the provider/vendor of the entire firmware resource update driver package. |
| Class/ClassGuid | Firmware/  {f2e7dd72-6468-4e36-b6f1-6488f42c1b52} | Classifies the driver package as a firmware resource update. This is a new built-in device setup class in Windows 8. |
| DriverVer | MM/DD/YYYY,2.0.0.0 | Specifies the date of the driver package. The date and version should both reflect the date and version of the actual firmware resource update as closely as possible in order to ensure that the PnP device installation system can accurately select the best driver package available on the system. |
| CatalogFile | catalog.cat | Specifies the associated catalog file that signs the driver package INF file and all associated firmware resource update binaries. |
| PnpLockdown | 1 | Enables the PnP driver file lockdown mechanism in order to protect installed driver files from being modified externally by unrelated applications. For firmware resource updates, this setting should always be enabled to ensure that firmware resource image files cannot be tampered with outside of the control of the PnP system. |
| **[Manufacturer]** | | INF [Manufacturer] section lists all distinct driver manufacturers/vendors that define firmware resource updates. Each manufacturer line specifies an INF [<Models>] section and identifies its supported target platform. |
| %MfgName% | Fabrikam Inc.  (localized in [Strings] section) | Identifies the manufacturer/vendor of the firmware resource update. This may be the same as the Provider field. |
|  | Firmware,  NTarm | Identifies the INF [<Models>] section that defines the firmware resource devices supported by this driver package, including their target driver platforms. In this example, the drivers are only targeted for the ARM-based NT platform and the [<Models>] section is [Firmware.NTarm]. |
| **[Firmware.NTarm]** | | INF [<Models>]section for the ARM-based NT platform that lists all firmware resource devices for which updates are defined. Each hardware model line specifies an INF [<DDInstall>] section and its associated hardware ID match. |
| %FirmwareDesc% | Fabrikam System Firmware 2.0  (localized in [Strings] section) | Describes the firmware resource update. This is the primary description string used to present the associated firmware resource device instance in Device Manager and other device related UI. For this reason, the description may include the firmware vendor and version. |
|  | Firmware\_Install,  UEFI\RES\_{RESOURCE\_GUID} | Identifies the INF [<DDInstall>] section containing the installation steps for the firmware resource update that targets the device instance identified by the UEFI\RES\_{RESOURCE\_GUID} hardware ID. Where RESOURCE\_GUID is the GUID of the firmware resource that is being updated. |
| **[Firmware\_Install.NT]**  CopyFiles = Firmware\_CopyFiles  **[Firmware\_CopyFiles]**  ... | | INF [<DDInstall>]section containing the installation steps for the firmware resource update. For firmware resource updates, this only defines the firmware resource image file to copy into place for a firmware resource update. In this example, the [<DDInstall>] section is [Firmware\_Install.NT]. |
| *firmware.bin* |  | Specifies the firmware resource update image file to copy. See section [DestinationDirs] below for details about where this file is copied. |
| **[Firmware\_Install.NT.Hw]**  AddReg = Firmware\_AddReg  **[Firmware\_AddReg]**  ... | | INF [<DDInstall>.Hw]section containing the hardware-specific installation steps for the firmware resource update. For firmware resource updates, this defines the firmware resource update configuration information in the form of registry values that are set under the device hardware key of the target device instance. |
| FirmwareId | {RESOURCE\_GUID} | The firmware GUID of the firmware resource update. Note that this is the same firmware resource GUID that is embedded in the UEFI\RES\_{RESOURCE\_GUID} hardware ID, however it must be specified here as a standalone value since the PnP system treats all hardware IDs as opaque strings that are strictly used for device/driver matching purposes. |
| FirmwareVersion | 0x00000002 | The firmware version of the firmware resource update, specified as a REG\_DWORD value. |
| FirmwareFilename | {RESOURCE\_GUID}\*firmware.bin* | The firmware filename of the firmware resource update’s Update Capsule image filename. This path is relative to the %SystemRoot%\Firmware directory such that {RESOURCE\_GUID} represents a subdirectory used to organize all firmware image files targeted for specific firmware resource. |
| **[SourceDisksNames]** | | INF [SourceDisksNames] section lists all distinct driver package source disk locations where associated driver files, such as firmware update resource image files, are contained. |
| 1 | %DiskName% = Firmware Update  (localized in [Strings] section) | Specifies an arbitrarily numbered driver package source disk ID and its description name. No optional driver package relative subdirectory is specified so any driver files associated to this disk ID, like the firmware resource update image file, are expected to live directly beside the INF file. |
| **[SourceDisksFiles]** | | INF [SourceDisksFiles] section lists all driver files referenced by the driver package and links them to a disk ID from the INF [SourceDisksNames] section. |
| *firmware.bin* | 1 | Establishes the *firmware.bin* firmware resource update image file as being part of the driver package by linking it with the primary disk ID. No optional file-specific subdirectory is specified so this driver file is expected to live relative to its disk ID’s subdirectory, which in this case is right beside the INF file. |
| **[DestinationDirs]** | | INF [DestinationDirs] section lists the target destination directories of all driver files referenced by the driver package. |
| DefaultDestDir | %DIRID\_WINDOWS%,Firmware\  {RESOURCE\_GUID} | Specifies the default destination directory of all driver files copied by this driver package to be %SystemRoot%\Firmware, where DIRID\_WINDOWS (10) represents the base %SystemRoot% directory and {RESOURCE\_GUID} represents a subdirectory names after the firmware resource GUID. |
| **[Strings]** | | INF [Strings] section defines key/value mappings for all indirect string tokens (%token%) in the driver package INF file. Use of string tokens enables a driver package INF file to be easily localized by introducing locale-specific [Strings.<LanguageID>] sections. It can also be useful to use string token substitution for defining constant numeric values, such as REG\_DWORD. |
| Provider | "Contoso Ltd." | An example of a string token key/value mapping. |

Table 4: INF File Sections and Directives

It is important to use a unique name for each firmware resource update image file version in order to avoid any potential collisions with other firmware image files, both your own and those from other firmware vendors. For example, *firmware.bin* from the above should be assigned the following name to satisfy both vendor name and version constraints:

* Fabrikam-System-Firmware-2.0.bin

In order to ensure that variants of a given firmware resource update image, potentially used for OEM/IHV customization purposes, do not collide when deployed into the same Windows system image, it is recommended that each distinct firmware resource update image is maintained under a subdirectory within the %SystemRoot%\Firmware directory. This subdirectory should be named after either the target firmware resource GUID. For example, the following firmware resource update image paths satisfy the deployment constraints:

* %SystemRoot%\Firmware\{6bd4efb9-23cc-4b4a-ac37-016517413e9a}\Fabrikam-System-Firmware-2.0.bin

### Signing the Firmware Driver Package

Once the driver package INF file and firmware payload binary are ready, the entire driver package must be signed in order to produce a catalog file. It is crucial that this catalog file vouch for the validity and authenticity of the INF file and firmware payload binary contained within the driver package in order to enable Windows to securely initiate a firmware resource update.

The steps to self-sign the driver package for test purposes are enumerated below. Please note that these steps are for test purposes only. In production, firmware update driver packages must be submitted to the Windows Dev Center Hardware Dashboard for signing. For the steps to sign a firmware driver package for production see section 5.2.4 Signing the Firmware Update Package.

1. Install the [Windows 8 SDK](http://msdn.microsoft.com/en-US/windows/apps/br229516) and [Windows Driver Kit 8](http://msdn.microsoft.com/en-US/windows/hardware/hh852365) from MSDN. This will install the **makecert**, **pvk2pfx** **inf2cat** and **signtool** tools under the “%systemdir%\Program Files (x86)\Windows Kits\8.0\bin\x86.
2. Make a test certificate

makecert.exe -r -pe -a sha256 -eku 1.3.6.1.5.5.7.3.3 -n CN=Foo -sv fwu.pvk fwu.cer

pvk2pfx.exe -pvk fwu.pvk -spc fwu.cer -pi <Password entered during makecert prompt> -spc fwu.cer -pfx fwu.pfx

1. Create a catalog file

Inf2Cat.exe /driver:"." /os:8\_x64

\*\*\* Change the /OS argument depending on the OS for which the firmware driver package is intended for

\*\*\* /driver argument points to the location where the INF is located

For more information about catalog files, refer to [Catalog Files and Digital Signatures](http://msdn.microsoft.com/en-us/library/windows/hardware/ff537872(v=vs.85).aspx). For details about how to create a catalog file for use with a driver package, refer to [Creating a Catalog File for a PnP Driver Package](http://msdn.microsoft.com/en-us/library/windows/hardware/ff540161(v=vs.85).aspx).

1. Sign the catalog file

signtool sign /fd sha256 /f fwu.pfx /p <Password entered during makecert prompt> delta.cat

1. Install the test certificate on the test system. Double click on the fwu.cer file, it should show an option to “Install Certificate”. Proceed with the installation. Choose the following options during the certificate installation:

For Store location, choose “Local Machine”

For Certificate Store, browse and select “Trusted Root Certification Authorities”.

1. Disable secure boot in the firmware/BIOS options.
2. Set the testsigning on in the BCD options so that the OS loader can load the firmware image file (*firmware.bin*) during boot even if the catalog is not production signed. As an administrator, run

bcdedit /set testsigning on

Once the driver package is signed, it can be installed using one of the following well-established mechanisms:

1. **Device Manager** – For manual testing, Device Manager provides a friendly interface for locating a firmware resource device and updating its driver in order to initiate a firmware resource update.
   1. Locate the desired firmware resource device under the “Firmware” class while viewing devices by type, or under the “Microsoft UEFI-Compliant System” device while viewing devices by connection.
   2. Right-click on the firmware resource device and select the “Update Driver Software...” option.
   3. Use the “Browse my computer for driver software” option to locate and install a newer firmware resource update driver package onto the firmware resource device. This operation will ensure that the specified firmware resource update driver package is in fact newer than any existing firmware resource update driver package that might already be on the firmware resource device before adding it to the Windows Driver Store and initiating an installation.
2. **PnP Util** – For automated testing, the pnputil command line utility can be used from an Administrator-elevated command prompt to import a firmware resource update driver package into the Windows Driver Store and initiate a device installation on any/all applicable firmware resource devices that are presently using an older firmware resource version, as established by the DriverVer of their currently installed driver package INF file or a lack of a 3rd party supplied driver package INF file altogether. For example, use the following command line to add and install X:\firmware.inf:

pnputil -i -a X:\firmware.inf

If the firmware resource update was successfully installed on a firmware resource device and it supplies a firmware resource update that is a higher version than the current firmware version, then the device will be awaiting a system reboot in order to complete the update operation. A device in this state will indicate its need for the system to be rebooted by maintaining a device problem, which prevents the device from being started and restored to a steady state until the reboot is performed.

### Validating the status of the firmware update

When a firmware driver package is successfully installed, PnP will request a system reboot to apply the updates. Post reboot, the status of the update can be validated. The status of the update is maintained under the registry key

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\FirmwareResources\{RESOURCE\_GUID}\

Where RESOURCE\_GUID is the GUID of the resource (from ESRT) that was updated.

The “LastAttemptStatus” registry value indicates the status of the firmware update, where a value of 0 indicates success and any non-zero value represents a failure. The value for this registry key are NTSTATUS codes populated by OS Loader based upon the value of the LastAttemptStatus from the ESRT. The following table maps the LastAttemptStatus code to its corresponding NTSTATUS code.

|  |  |  |  |
| --- | --- | --- | --- |
| **LastAttemptStatus** | **Code** | **NTSTATUS** | **Code** |
| Success | 0 | STATUS\_SUCCESS | 0x00000000 |
| Error: Unsuccessful | 1 | STATUS\_UNSUCCESSFUL | 0xC0000001 |
| Error: Insufficient Resources | 2 | STATUS\_INSUFFICIENT\_RESOURCES | 0xC000009A |
| Error: Incorrect Version | 3 | STATUS\_REVISION\_MISMATCH | 0xC0000059 |
| Error: Invalid Image Format | 4 | STATUS\_INVALID\_IMAGE\_FORMAT | 0xC000007B |
| Error: Authentication Error | 5 | STATUS\_ACCESS\_DENIED | 0xC0000022 |
| Error: Power Event, AC Not Connected | 6 | STATUS\_POWER\_STATE\_INVALID | 0xC00002D3 |
| Error: Power Event, Insufficient Battery | 7 | STATUS\_INSUFFICIENT\_POWER | 0xC00002DE |

Table 5: UEFI Firmware Status to NTSTATUS Code Mapping

The Hardware ID property of the firmware resource device node should also reflect the change in the firmware version.

* UEFI\RES\_{RESOURCE\_GUID}&REV\_XXX
  + Where XXX is the new firmware version.

If for some reason the firmware update failed, you can retry the failed firmware update by going to the Device Manager, right-click the firmware resource device listed under the Firmware node and click update driver software, browse my computer for driver software, let me pick from a list of device drivers on my computer, and pick the same driver that you installed previously. After clicking on ok, it will ask you for a reboot. On the next reboot, the OS Loader should call into UpdateCapsule with the payload of the firmware driver package.

## Processing Updates

After any firmware update package(s) have been applied and the system subsequently rebooted, the Windows OS loader loads all the firmware payload files (in this example firmware.bin) into physical memory. The Windows OS loader creates capsule headers using the information from each update’s corresponding ESRT entry, which describes the GUID and flags to use when calling UEFI UpdateCapsule. In setting each capsule header’s flags field, the Windows OS loader always sets CAPSULE\_FLAGS\_PERSIST\_ACROSS\_RESET and CAPSULE\_FLAGS\_INITIATE\_RESET. The Windows OS loader may additionally set CAPSULE\_FLAGS\_POPULATE\_SYSTEM\_TABLE for firmware types DEVICE\_FIRMWARE, if the capsule flag was specified in the INF for the driver package. Proprietary capsule flags may also be specified in the INF and when specified will additionally be included when calling UEFI UpdateCapsule.

Referring back to the ESRT example (Table 2) and the firmware resource update driver package (Figure 6), the capsule headers the Windows OS loader creates to pass into UEFI UpdateCapsule would be as follows:

|  |  |  |
| --- | --- | --- |
| **Field** | **Value** | **Comment** |
| CapsuleGuid | {SYSTEM\_FIRMWARE} | From corresponding ESRT resource entry’s FirmwareClass. |
| HeaderSize | … | Padded to page-align *firmware.bin* start. |
| Flags | 0x50000 | Persist across, and initiate, reset. |
| CapsuleImageSize | … | Capsule Header Size + The size of *firmware.bin*. |

Table 6: Capsule Header for the System Firmware Update Capsule

Note that in this example only one of the two devices defined in the ESRT table has installed a new firmware resource update driver package. If a firmware resource update driver package were authored for the second device in Table 2 and then installed on the corresponding firmware resource device, a second capsule header would be created as follows:

|  |  |  |
| --- | --- | --- |
| **Field** | **Value** | **Comment** |
| CapsuleGuid | {DEVICE\_FIRMWARE} | From corresponding ESRT resource entry’s FirmwareClass. |
| HeaderSize | … | Padded to page-align DEVICE.BIN start. |
| Flags | 0x78010 | Persist across, and initiate, reset, and populate system table, OR’d with 0x8010 from corresponding ESRT resource entry’s CapsuleFlags. |
| CapsuleImageSize | … | Capsule Header Size + The size of DEVICE.BIN. |

Table 7: Capsule Header for the Device Firmware Update Capsule

Once the Windows OS loader has loaded all pending firmware updates and created the necessary data structures to describe them, it then calls the UEFI UpdateCapsule run-time service, prior to calling ExitBootServices().

**Note**: UpdateCapsule() is called prior to ExitBootServices() when platform firmware has exclusive control of all devices including the storage device. A platform firmware implementation of UpdateCapsule() can save firmware update payloads to persistent storage to stage an update or to support a recovery rollback.

## Device I/O From the UEFI Environment

When the Windows OS loader calls the UpdateCapsule function, each capsule contained in the CapsuleHeaderArray is executed. The order of capsule execution is dependent on the UEFI firmware implementation, and a capsule cannot make any assumption as to the order of its execution relative to other capsules or take any dependencies on other capsules. Each capsule is a self-contained payload, comprising both the executable UEFI code to manage the update and the firmware image.

When the capsule is called, the executable code contained in the capsule is responsible for opening a communication channel with the target device. The appropriate channel will depend on the system’s device topology, the capabilities of the target device, and the UEFI boot services and drivers provided by the particular UEFI implementation. Capsule implementers may need to consult with the UEFI BIOS vendor regarding the options available in the targeted UEFI environment. Typically, communication is established by utilizing a UEFI device driver for the given device. This driver enables the capsule update code to bind to the device via a well-known device path using the appropriate protocol.

Once communication is established, the update management code writes the firmware image to the targeted device. After completing the update, an appropriate return status code is written to the device’s Firmware Resource Entry in the ESRT. The update management code then returns control to the UpdateCapsule function.

For details on the UpdateCapsule function, the structure of a capsule, and UEFI boot services drivers and protocols, refer to the UEFI specification at <http://www.uefi.org/specs/>.

## Seamless Crisis Prevention & Recovery

If a firmware update fails, the results can be devastating. At best, the update fails, but the system is resilient and recovers without the end-user becoming aware. At worst, it is possible for a failed firmware update to result in a completely unusable system, requiring the end-user to return their system to the retailer or manufacturer for repair. The latter case is what we call a ‘crisis.’

A crisis can result from a failed firmware update, or due to firmware that is incompatible with Windows or other aspects of the system. This section discusses features designed to prevent and recover from crises that resulting from failed firmware updates. Our expectation is that firmware update test coverage by the firmware author will prevent the majority of crises resulting from incompatible firmware.

In order to provide a great experience for Windows 8 end-users, the following crisis prevention and recovery requirements must be met for the firmware driver package update mechanism. These requirements do not preclude additional crisis prevention or recovery solutions.

### Pre-Installation Criteria

When the system firmware is performing the actual update there are a series of pre-installation checks that must be performed. System firmware must perform this check to ensure there is enough power to complete the update. It is also recommended that the checks be made for each of the updates before the update is applied if there are multiple firmware updates. The list of items to check and validate are provided below. All of the checks must be performed where applicable. There is no specific order to the execution of the tests.

* **Power Check**:
  + System must have at least 25% battery charge
  + Tethered power (power via USB cable and/or AC power) is not required[[1]](#footnote-1)
* **Security Check**: This check is composed of 2 sub checks:
  + Validate update capsule payload is properly signed
  + Validate that any PE-based EFI files in the payload are properly signed with a proper EFI cert
* **Integrity Check:** Perform an integrity check on the firmware update payload
* **Version Check:** Verify that the firmware being applied does not downgrade the current, installed firmware beyond the LowestSupportedFirmwareVersion value
* **Storage Check:** This is done as appropriate depending upon the system’s hardware
  + There is sufficient room for backups of the current firmware which will be replaced
  + There is sufficient room in the device to accommodate the new firmware

List 1: Pre-installation Criteria

Any failure must result in an appropriate Last Attempt Status error code; refer to Table 15 (ESRT Last Attempt Status Field Values) for the specific device or system which is being updated.

If multiple updates are being applied and some pass the pre-installation checks and others do not, platform firmware can proceed with updating the firmware for the resources that passed the pre-installation checks. However, any resource that failed the pre-installation check must not be updated.

### Post-Installation Criteria

Once firmware (device or system) has been installed it must be checked to validate that the new updated firmware image is what was intended. This is to minimize any risks of any corruption introduced during the actual updating process (for example sticky bits in flash ROM, noise on a bus during the updating and so on).

The updating process must validate that the updated firmware passes integrity checks. If it fails then it must recover by rolling back to the last known good version of the firmware.

Any failure must result in an appropriate Last Attempt Status error code; refer to Table 14 (ESRT Last Attempt Status Field Values) for the specific device or system which is being updated.

### Recovering from install and boot failures

In order to prevent a system from reaching a non-bootable state, the firmware update mechanism must meet the following requirements in cases where firmware updates fail to install, or in cases where the system fails to boot successfully.

**Note**: Below, the term “committed” is used to describe firmware. Once firmware has been ‘committed,’ the firmware is treated as fully installed, and will not be automatically rolled back by the firmware due to boot failure, etc. “Uncommitted” firmware describes partially updated firmware and can potentially be rolled back to a previous version in cases where the firmware update cannot be completed or a failure is detected by the updating firmware (e.g., invalid CRC check in the update). Whether firmware is committed is something that the firmware should track internally, and is not captured as part of the ESRT.

### Firmware update unsuccessful

If an individual system or device firmware cannot be installed, or has been installed incorrectly (e.g. due to some kind of corruption, or a power loss while installing the update), the update may be retried up to a total of three (3) attempts, including the first attempt. If additional retry attempts will be performed by the firmware, the system must not boot into Windows between any of the attempts. If all attempts fail, the updating firmware must discard the update. If the update was partially applied, the firmware must roll back to the previous version. The firmware must roll back to the previous version without any user interaction. The update failure does not affect other pending updates; pending firmware updates should be attempted.

Once all of the updates have been processed, UEFI will resume booting Windows. The UEFI firmware must verify that any non-committed firmware updates were successfully installed in order to mitigate problems due to power loss (UEFI should never attempt to boot Windows with partially written firmware).

Possible causes for install failure include, but are not limited to:

* Insufficient resources
* Power loss
* Hardware failure

### Firmware update succeeds, but Windows fails to boot

The UEFI firmware is not responsible for rolling back updated firmware once it has been committed. Existing failover logic in Windows will divert the end-user to the Windows Recovery Environment (WinRE) after two unsuccessful boot attempts. WinRE may or may not boot successfully. The end-user will have to take a manual recovery step to recover their system, or will have to return their device to the retailer/manufacturer.

Possible causes for this class of failure include, but are not limited to:

* Firmware incompatible with OS drivers
* Firmware incompatible with OS components

If a hardware vendor decides to implement additional logic to determine whether Windows has successfully booted, that is acceptable. As mentioned previously, the expectation is that firmware update test coverage by the firmware author will prevent the majority of crises resulting from incompatible firmware.

## Firmware Update Status

At this point it is expected that all firmware updates will be applied, and the results of all updates reflected in the ESRT on the subsequent invocation of the Windows OS loader. Considering once again the ESRT example (Table 2) and it associated firmware resource update driver package (Figure 6), if version 2 of firmware.bin was successfully applied by firmware, then the new ESRT table would reflect this. Notice that the only difference in the table is that the Firmware Version and Last Attempt Version fields for the system firmware resource entry have changed to reflect the successfully applied new firmware version:

|  |  |  |
| --- | --- | --- |
| Field | Value | Comment |
| Firmware Resource Count | 2 | This table contains two firmware resource entries. |
| Firmware Resource Maximum | 2 | This table allocation contains enough space to describe a maximum of two resources. |
| Firmware Resource Version | 1 | The firmware resource entry format version this table uses is 1. |
| Firmware Resource Entry Array | Firmware Resource Entry 0 |  |
| Firmware Class | {SYSTEM\_FIRMWARE} | This GUID identifies the system firmware for update via PnP. |
| Firmware Type | 1 | System firmware type is 1. |
| **Firmware Version** | **2** | **The current system firmware version is 2.** |
| Lowest Supported Firmware Version | 2 | Change the lowest supported firmware version to 2, so that the firmware cannot be rolled-back to a version earlier than version 2. This value is typically changed when the firmware update contains security fixes. |
| Capsule Flags | 0 | System firmware doesn’t define any private capsule update flags. |
| **Last Attempt Version** | **2** | **The last system firmware version for which an update was attempted was 2.** |
| Last Attempt Status | 0 | The last system firmware update attempt was successful. |
|  | Firmware Resource Entry 1 |  |
| Firmware Class | {DEVICE\_FIRMWARE} | This GUID identifies the device firmware for update via PnP. |
| Firmware Type | 2 | Device firmware type is 2. |
| Firmware Version | 1 | The current device firmware version is 1. |
| Lowest Supported Firmware Version | 1 | Keep the lowest supported firmware version as 1. The firmware can be rolled-back to version 1 if necessary. |
| Capsule Flags | 0x8010 | Device firmware defines private capsule update flags (0x8010). |
| Last Attempt Version | 1 | The last device firmware version for which an update was attempted is 1. |
| Last Attempt Status | 0 | The last device firmware update attempt was successful. |

Table 8: Updated ESRT After Successful Completion of System Firmware Update to Version 2

If the firmware cannot successfully apply an update, then the Firmware Version, Last Attempt Version and Last Attempt Status entries in the ESRT will reflect the failed update attempt. For example, if the system is attempting to update version 1 of the firmware to version 2, and fails to successfully apply the update, then the Firmware Version = 1, Last Attempt Version = 2, and Last Attempt Status != 0. (I.e. the Last Attempt Status is set to the appropriate non-zero error code indicating the reason failure occurred. The list of valid error codes for this entry can be found in Appendix B: UEFI System Resource Table (ESRT) Detailed Description.)

Although the standard update policy enforces that firmware versions can only increase, this policy can be disabled for test purposes via the Policy setting as described in the authoring an update package section.

### Push-Button Reset

Push-Button Reset allows end-users to revert their systems back to factory settings. It achieves this by re-installing the Windows image pre-loaded on to a system during the manufacturing process. The entire OS, including drivers and applications, will be reinstalled.

**Note**   Due to security requirements which prevent firmware rollback across security boundaries, Push-Button Reset is unable to roll back firmware versions to match the original firmware deployed in the factory. This means that all versions of firmware must be backwards compatible with all driver and operating system versions shipped on that platform. If firmware is not compatible, this could result in a user returning their system to the manufacturer.

### Rolling Back Firmware Updates

In some cases it may be necessary to rollback a firmware update for example during update testing. Each ESRT reported firmware resource has an entry in the following registry key:

* HKLM\SYSTEM\CurrentControlSet\Control\FirmwareResources\

The entry itself is a key with the Name equal to the GUID used to report the resource in the ESRT. In order to allow a firmware rollback, you must create a REG\_DWORD value called Policy and set the value to 1. A given firmware resource can only be rolled back to its respective Lowest Supported Firmware Version, as specified in the ESRT. This is to prevent firmware rollbacks beyond the point at which a critical security fix has been made to the firmware. If the firmware version you are rolling back to meets these conditions the OS loader will update to an older version.

# UEFI Firmware Update User Experience

During the process of updating firmware it is important to provide a visual notice to the end user that an update is being processed. Over time users become accustomed to the typical time it takes for their device to boot into Windows. If a firmware update occurs which extends this boot time then there is a need to notify the user that the extended boot time is expected. Otherwise users may conclude that the device failed to boot or froze during the boot process resulting in the user manually rebooting the system, interrupting the firmware update process.

To avoid such situations the firmware performing an update must manage the user experience by displaying a simple notification that the device is being updated. This will reset the user’s boot time expectations. This user experience must be added to an already existing (and familiar to the user) boot screen such as in Figure 7. The graphic displayed is likely the OEM or motherboard manufacturer’s logo.



Figure 7: Standard OEM Boot Screen

## User Experience

During the firmware update process the display must show the user that an update is in progress. The goals for this User Experience (UX) are as follows:

* Display must be very short and easy to understand
* Must have the Windows 8 look and feel
* Must convey the messages:

1. Do not disturb the system during this process (don’t unplug from power, etc.)
2. Boot time may take longer than expected
3. The process is still in progress

The image in Figure 8 shows the expected look and feel for this UX. An OEM image is displayed (in this example the fictitious Contoso logo) as it would be displayed during any other time the system boots The text “Please wait while we install a system update” indicates that a critical system component update is in progress. Users have learned that this means to let the device do what it must do and to not disturb it as it may take some time.  
Since this message does not normally appear during the boot sequence users will expect some extra time is warranted.



Figure 8: Firmware Update Boot Screen

## Time Frame

During the initial boot process the system will need to present the user with the standard boot screen as would be expected from the device. However once the Windows bootloader detects that new firmware capsule files are present it will transition from the normal boot screen (Figure 7) to the firmware update boot screen (Figure 8). The transition will include the Windows bootloader displaying localized text on the screen indicating an update is in progress before it makes a call into the UpdateCapsule(). The entire firmware update process must continue to display this firmware update boot screen (Figure 8) for the duration of the update process; until Windows boots after all firmware updates have been completed.

The firmware update UX needs to display from the time UpdateCapsule() is called until the time that all firmware updates have been successfully applied and the system has handed off the boot process to Windows. If, during this time, additional reboots are needed then every attempt must be made to continue displaying the Firmware Update Boot Screen (Figure 8) without interruption. If it is not possible to do so (e.g. graphic chip firmware has been updated, a cold reboot was required, etc.) then every attempt must be made to re-display the Firmware Update Boot Screen (Figure 8) screen as soon as possible. To facilitate this requirement, the Windows bootloader will provide the firmware a bitmap copy of the localized text (details to follow).

## Firmware Update Boot Screen Components

There are two components to the firmware update boot screen as illustrated in (Figure 9). Each component is described below:

### Firmware Update Boot Screen Component A: OEM Logo

The OEM logo (component A in Figure 9) must be the same logo which is displayed during the normal boot process. For firmware update boot screens the logo must be the same size, position and quality expected as is displayed during the normal boot process.

### Firmware Update Boot Screen Component B: Update Text

The update text (component B in Figure 9) is a simple text which is designed to be quick to read and easy to understand. The text is rendered by the Windows bootloader. Once it determines that firmware updates are pending then the bootloader determines the locale of Windows and displays the localized text on the screen.

During the call into UpdateCapsule() the bootloader will pass all firmware update capsules. In addition it will also pass in a special Microsoft defined “Firmware Update Display Image” capsule containing a bitmap of the text which is displayed and the location of the bitmap on the screen. It is the responsibility of the system firmware’s UpdateCapsule() method to persist the capsule so that any time the screen is cleared or modified it can re-display the bitmap on the screen.



Figure 9: Firmware Update Boot Screen Components

### Windows Firmware Update Display Capsule

When the Windows bootloader calls into the system firmware’s UpdateCapsule() it passes in all firmware update capsules. Additionally it will pass in a Windows UX capsule. This capsule contains the bitmap of rendered, localized text which must be displayed on the screen.

The GUID used to identify this capsule is: {3b8c8162-188c-46a4-aec9-be43f1d65697}

There is not guarantee of order the UX Capsule will appear in the array of capsules. Do not rely on a specific index position to find the UX Capsule. A best practice includes scanning the array looking for the UX Capsule and processing it before processing remaining firmware capsules in the array.

It is important to note that there may be some scenarios where there will be no UX Capsule for example if a “headless” Windows Server does not have a display adapter. In such cases the firmware UpdateCapsule call can ignore the UX Capsule requirement. However if the UX Capsule is present then UpdateCapsule must process it according to this specification document.

The Windows UX Capsule is described in Table 7 and Table 8:

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Byte Length | Byte Offset | Description |
| CapsuleGuid | 16 | 0 | FIRMWARE\_UPDATE\_DISPLAY\_CAPSULE. |
| HeaderSize | 4 | 16 | sizeof(EFI\_CAPSULE\_HEADER). |
| Flags | 4 | 20 | CAPSULE\_FLAGS\_PERSIST\_ACROSS\_RESET. |
| CapsuleImageSize | 4 | 24 | 4-byte unsigned integer describing the length of the firmware update display capsule (defined below). The size includes the header and capsule, which includes the display image. |

Table 9: Firmware Update Display Header

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Byte Length | Byte Offset | Description |
| Version | 1 | 28 | 1 |
| Checksum | 1 | 29 | The entire capsule (header and payload), including the display image, must sum to zero. |
| Image Type | 1 | 30 | 1-byte enumerated type field indicating the format of the image.  0 = Bitmap  1 – 255 = *Reserved* (for future use) |
| Reserved | 1 | 31 | Reserved for future use. Must be zero. |
| Mode | 4 | 32 | 4-byte unsigned long describing the graphics output protocol video mode. The value represents the video mode that is capable of displaying the bitmap. The value equals the *Mode* field of the *EFI\_GRAPHICS\_OUTPUT\_PROTOCOL\_MODE* structure when the image is rendered*.* |
| Image Offset X | 4 | 36 | A 4-byte (32-bit) unsigned long describing the X-offset of the bitmap image. (X, Y) display offset of the top left corner of the image. The top left corner of the display is at offset (0, 0). |
| Image Offset Y | 4 | 40 | A 4-byte (32-bit) unsigned long describing the Y-offset of the bitmap image. (X, Y) display offset of the top left corner of the image. The top left corner of the display is at offset (0, 0). |
| Image | N | 44 | Contains the bitmap to display during the firmware update process. |

Table 10: Windows Firmware Update Display Capsule Payload

Note that unlike a capsule generated for the firmware update payload, the display capsule payload is not padded to be page-aligned. The display payload immediately follows the capsule header.

The firmware update display capsule describes a graphic that must be rendered during the duration of a firmware update. The graphic is initially rendered and display by Windows and handed off to the firmware as part of the same UpdateCapsule call containing the update payload(s) to the firmware. If the firmware resets the system or the video device, the firmware must redisplay the bitmap provided in the display capsule. If physical memory is not persisted across the reset, the firmware may have to save the bitmap to persistent storage to redisplay the bitmap after the reset. The details on how to save and restore the bitmap across a reset are implementation specific and are not discussed in this paper.

The firmware update display capsule is modeled off of the Boot Graphics Resource Table (BGRT) defined in ACPI 5.0. The BGRT defines a mechanism for system firmware to provide a graphic to an OS boot loader. While the two tables are similar, there are a couple of notable differences.

|  |  |  |
| --- | --- | --- |
| BGRT | FW Update Display Capsule | Reason |
| Pointer to Bitmap | Embedded Bitmap | Embedding the bitmap allows the capsule to be saved and restored in a single operation. |
| Does not contain video mode | Contains video mode | Done to avoid requiring the firmware to query video mode during UpdateCapsule call. |
| Contain a Status field | Does not contain a Status field | The Status field of the BGRT describes whether the image is currently displayed on the screen. This is not applicable to the firmware update display capsule. |

Table 11: Differences between Firmware Update Display and BGRT

Each of the capsule fields are defined in greater detail below.

#### Version

The version field identifies which revision of the display capsule is implemented. The version field will be set to 1.

#### Checksum

The capsule contains a checksum to enable simple validation. The sum of the entire capsule must equal zero. If the sum does not equal zero, the capsule should be ignored.

#### Image Type

The Image Type field contains information about the format of the embedded image. If the value is 0, the Image Type is Bitmap. All other values not defined in the table are reserved for future use.

#### Mode

The Mode describes the Graphics Output Protocol video mode compatible with displaying the embedded image. The video mode is queried prior to calling *UpdateCapsule* and describes the current video mode and the video mode of the local display when the embedded image is displayed by the boot loader.

#### Image Offset

The Image Offset contains two consecutive 4 byte unsigned longs describing the (X, Y) display offset of the top left corner of the boot image. The top left corner of the display is at offset (0, 0).

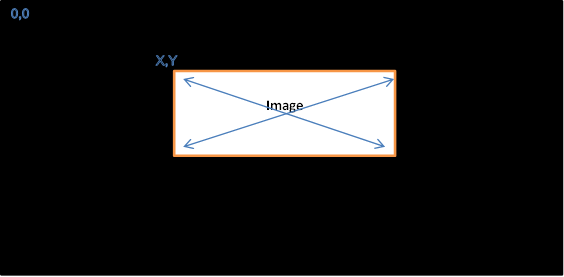


Figure 10: Image Offset Value Relative to Display

#### Image

The Image field is a byte-array containing an embedded Bitmap. The format for a Bitmap is defined by the “Types of Bitmaps”[[2]](#footnote-2) reference page. The bitmap can be either a 24-bit bitmap with the pixel format 0xRRGGBB or a 32-bit bitmap with the pixel format 0xrrRRGGBB, where ‘rr’ is reserved.

# Links & Appendices

* Catalog Files and Digital Signatures:  
  <http://msdn.microsoft.com/en-us/library/windows/hardware/ff537872(v=vs.85).aspx>
* Creating a Catalog File for a PnP Driver Package:  
  <http://msdn.microsoft.com/en-us/library/windows/hardware/ff540161(v=vs.85).aspx>

## Appendix A: UEFI System Resource Table (ESRT) Detailed Description

### Identifying the ESRT table pointer

The pointer to the ESRT table is identified via its corresponding GUID in the EFI configuration table:

|  |  |
| --- | --- |
| UEFI System Resource Table GUID | b122a263-3661-4f68-9929-78f8b0d62180 |

### UEFI System Resource Table Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Byte Length | Byte Offset | Description |
| Firmware Resource Count | 4 | 0 | The number of firmware resources, must not be zero. |
| Firmware Resource Maximum | 4 | 4 | The maximum number of resource array entries that can be added without reallocating the table, must not be zero. |
| Firmware Resource Version | 8 | 8 | The firmware resource entry version. |
| Firmware Resource Entry Array |  |  | Firmware Resource Entry 0 |
| Firmware Class | 16 | 16 | This GUID defines the class of systems for which an update gets applied. |
| Firmware Type | 4 | 32 | Identifies the type of firmware resource. |
| Firmware Version | 4 | 36 | The current firmware version, where a larger number represents a newer release. |
| Lowest Supported Firmware Version | 4 | 40 | The lowest firmware version that can be rolled back to, i.e. the last firmware version that contained a security fix. |
| Capsule Flags | 4 | 44 | The OS will set the upper capsule flags defined by the UEFI spec, but the low 16-bits are defined per capsule. |
| Last Attempt Version | 4 | 48 | Version of the last firmware update attempt. |
| Last Attempt Status | 4 | 52 | Status of the last firmware update attempt. |
| … |  |  | Firmware Resource Entry 1 |

Table 12: ESRT Table and Firmware Resource Entry Definitions

### ESRT Table Header

**Firmware Resource Count**

This firmware resource count field describes the number of firmware resource entries in the ESRT (must contain at least one entry).

**Firmware Resource Maximum**

The firmware resource maximum field describes the maximum number of resource entries that could be described by the current table without needing to re-allocate it (must be greater than or equal to Firmware Resource Count.)

**Firmware Resource Version**

The firmware resource version field represents the firmware resource entry version (this value should be set to 1).

### Firmware Resource Entry Array

**Firmware Class**

The firmware class field contains a GUID that identifies a firmware component that can be updated via update capsule. This GUID will be passed into the UEFI update capsule run-time service as the update capsule header’s CapsuleGuid parameter during update.

**Firmware Type**

The firmware type field describes the firmware resource type.

|  |  |
| --- | --- |
| Value | Definition |
| 0 | Unknown |
| 1 | System firmware |
| 2 | Device firmware |
| 3 | UEFI driver |

Table 13: ESRT Firmware Type Field Values

**Firmware Version**

The firmware version field represents the current version of the firmware resource, where a larger number represents a newer version.

**Lowest Supported Firmware Version**

The lowest firmware resource version to which a firmware resource can be rolled back for the given system/device. If a security related fix is available in this firmware version, then the least compatible version should be equal to the current firmware version.

**Capsule Flags**

The capsule flags field contains flags that will be passed into the UEFI update capsule run-time service in bits 0 – 15 of the update capsule header’s Flags field (the OS is responsible to configure bits 16 – 31 of Flags as defined by UEFI spec.)

| Offset | Field Name |
| --- | --- |
| Bits 0 - 15 | Capsule Flags |
| Bits 16 - 63 | Not used |

Table 14: ESRT Capsule Flags Field

**Last Attempt Version**

The last attempt version field describes the last firmware version for which an update was attempted (uses the same format as Firmware Version).

**Last Attempt Status**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastAttemptStatus** | **Code** | **NTSTATUS** | **Code** |
| Success | 0 | STATUS\_SUCCESS | 0x00000000 |
| Error: Unsuccessful | 1 | STATUS\_UNSUCCESSFUL | 0xC0000001 |
| Error: Insufficient Resources | 2 | STATUS\_INSUFFICIENT\_RESOURCES | 0xC000009A |
| Error: Incorrect Version | 3 | STATUS\_REVISION\_MISMATCH | 0xC0000059 |
| Error: Invalid Image Format | 4 | STATUS\_INVALID\_IMAGE\_FORMAT | 0xC000007B |
| Error: Authentication Error | 5 | STATUS\_ACCESS\_DENIED | 0xC0000022 |
| Error: Power Event, AC Not Connected | 6 | STATUS\_POWER\_STATE\_INVALID | 0xC00002D3 |
| Error: Power Event, Insufficient Battery | 7 | STATUS\_INSUFFICIENT\_POWER | 0xC00002DE |

Table 15: ESRT Last Attempt Status Field Values

## Appendix B: Firmware update signing process and requirements

This section describes policies that need to be met when signing UEFI firmware updates intended for consumption by Windows 8 devices with UEFI Secure Boot active.

### Overview

The signing process entails some combination of the following components:

1. Signing the updated firmware image(s)
2. Signing the capsule carrying the updated firmware
3. Signing the firmware update package provided to the operating system; this package will contain the capsule

Of these, only c) is always required. On a Secure Boot system, all UEFI firmware must be signed, which implies that a) is required when the update concerns UEFI drivers or applications. For Connected Standby systems, a) is also required for all system firmware. If the firmware update is not protected by other means, the capsule must be signed in order to protect the firmware update itself and to ensure authenticity of the update package before being installation.

Figure 11 below indicates the signer for the various components discussed herein.

Updated System Firmware

Updated System Firmware

Windows Firmware Update Package

INF file

Catalog file

Capsule

Capsule

Updated System Firmware

Updated Device Firmware

Microsoft-signed

Color marking:

OEM-signed

OEM-signed or IHV-signed (device firmware)

UEFI Drivers

Signed with key in UEFI “db” (not applicable to ARM)

Figure 12: Firmware Update Components and Signers

### Signing the Updated Firmware

When signed, the signature of the updated firmware must be possible to validate by the systems’ firmware loader during boot. At a minimum, this will occur automatically on reboot, but pre-validation is strongly recommended for reliability and user experience reasons.

On ARM systems, no UEFI drivers or applications separate from the firmware image itself can be installed since the only allowed UEFI PE/COFF image is the Microsoft Windows OS loader (BootMgfw.efi), and it will be verified using the sole UEFI Allowed Database entry containing the Microsoft Windows Production CA 2011. Therefore, only system and device firmware can be added. On non-ARM systems, UEFI drivers and applications can be signed with any key chaining back to a key in the UEFI Allowed Database.

System or device firmware can either be signed with a key chaining back to a key bound to system boot ROM or be protected by other means (e.g. a signed capsule and then protected installation).

### Signing the Capsule

The capsule contents are determined by the OEM. The capsule may just contain a catalog of firmware images to update in whatever format the OEM chooses, or it may be delivered in the form of an EFI Application image (PE/COFF file format). If the capsule is a PE/COFF file then it must be signed by the OEM before submitting to Microsoft for Windows Firmware Update Package signing.

On ARM-based systems, since no keys other than the Microsoft Production CA 2011 are allowed in the UEFI Allowed Database (“db”) and Microsoft won’t use a signer under this CA to sign 3rd-party UEFI code, load of such a capsule cannot leverage the regular UEFI LoadImage() service. The capsule application may, however, be loaded using a platform-specific verification against the boot ROM public key or the UEFI PK[[3]](#footnote-3). More generally, when capsule signing is deemed necessary (e.g. to ensure integrity and authenticity of the complete update package), and the capsule may comprise firmware updates for firmware outside of UEFI, the capsule should be signed in such a way that it can be verified using platform-held, non-UEFI keys (e.g. signed using a key chaining back to a public key bound to boot ROM or the UEFI PK).

On non-ARM systems, the capsule can be an EFI application as long as it is signed with a key chaining back to an entry in the UEFI Allowed Database. UEFI Secure Boot can then automatically be leveraged to verify the integrity of the capsule.

### Signing the Firmware Update Package

The firmware update package needs to be submitted to the Windows Dev Center Hardware Dashboard to be signed. This step will create a catalog-signature of the package contents. The catalog-signature is used by the Microsoft OS loader to verify that the package is authentic and has not been tampered with before the actual update is provided to the firmware through UpdateCapsule().

Submitting the firmware update package to the Microsoft Windows Dev Center Hardware Dashboard for signing:

1. Sign the contents of the capsule as per instructions in section 5.2.3 Signing the Capsule.
2. Create a firmware update package that includes the capsule.
3. Test sign the firmware update package as per instructions in section 3.3.1 Signing the Firmware Driver Package.

**Note**   Windows 8 does not allow OEM verisign signed firmware update packages, even under test environment.

1. Update the firmware by installing the firmware update package.
2. Install the Windows Hardware Certification Kit (HCK) on the test system and run all the tests applicable to the firmware device.
3. Submit the HCK logs and the driver to the Windows Dev Center Hardware Dashboard for signature.

**Note**   While submitting the firmware update driver package, make sure to select only Windows 8 as the applicable OS. If you choose any down-level OS, then the Windows Dev Center Hardware Dashboard will sign the catalog in the driver package with SHA1 algorithm. Windows 8 requires all firmware update driver packages to be SHA256 signed.

1. In a test/laboratory environment it is acceptable to have no battery present yet still allow firmware updates as long as tethered power is provided. However a differentiation must be made between a dead/not charging battery and no battery present. [↑](#footnote-ref-1)
2. Types of Bitmaps - <http://msdn.microsoft.com/en-us/library/at62haz6.aspx> [↑](#footnote-ref-2)
3. Note that this load must still be measured into TPM PCR[7] as for any other image. [↑](#footnote-ref-3)