UEFI on Dell BizClient Platforms

Authors:

Anand Joshi

Kurt Gillespie

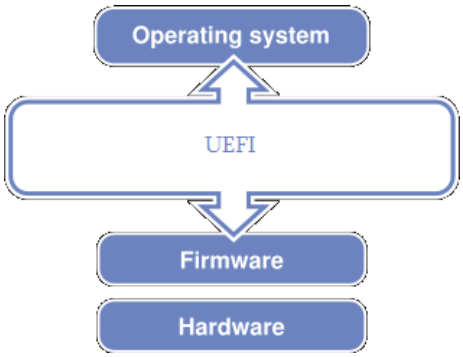
January 2013 | Rev 1.0

Brief History

In 2005 several leading tech companies came together to create the UEFI Forum. Based on Intel's earlier EFI architecture, the UEFI Forum's mission was to create the first ever industry standard firmware interface specification - UEFI (Unified Extensible Firmware Interface). Dell is represented on the UEFI board of directors of the UEFI Forum.

What is UEFI?

The UEFI (Unified Extensible Firmware Interface) specification defines an interface between operating systems and platform firmware. The interface consists of data tables that contain platform-related information, plus boot and runtime service calls that are available to the operating system and its loader. Together, these provide a standard, modern environment for securely booting an operating system and running pre-boot applications.



In addition to the services UEFI defines, there are various protocols/APIs to access various hardware and the boot devices in the system. The UEFI spec also defines a generic framework and can be adapted to any type of bus or device, knowing that computer hardware is constantly evolving. Visit *http://www.uefi.org* for more information on the UEFI Forum that controls the UEFI specifications or to download the UEFI specifications.

There are 3 types of entities that can execute under UEFI environment:

- **Applications**: Some examples of common UEFI applications include the UEFI shell, UEFI shell commands, flash utilities, and diagnostic utilities. It is perfectly acceptable to invoke UEFI applications from inside other UEFI applications. Applications can reside inside firmware shipped on a system, or can be located/installed on storage media, such as a SSD/HDD, PCI card internal memory, or USB Key.
- **OS Loader:** A special type of UEFI application, called an OS boot loader, provides the necessary initialization routines until the OS loader has set up enough of the OS infrastructure to be ready to assume complete ownership of the platform resources. OS Loaders, usually installed as part of the Operating System and usually located on the same storage media the Operating System is stored on, have the vital role of transitioning the system into Runtime mode, including coordinating virtual memory mapping to firmware code/data that will continue to be utilized during/after the OS has booted.
- **Drivers:** UEFI drivers differ from UEFI applications in that the driver stays resident in memory unless an error is returned from the driver's entry point. The UEFI core firmware, the boot manager, or other UEFI applications may load drivers. The PCI spec refers to code on an external card or add-in device as an "Option ROM" or "oprom", and therefore a UEFI driver may also be referred to as a "UEFI Option ROM" or "UEFI oprom" by some in the industry. UEFI drivers, like UEFI applications, can be found inside the system's firmware, or on storage media, such as a SSD/HDD, PCI card internal memory, or USB Key.

What does UEFI have to offer over traditional BIOS?

The primary goal of UEFI is to define an architecture that can scale with time, and offer a structured coding environment that allows easy enablement of newer technologies. Some of the distinguishing characteristics of UEFI, when compared to a traditional BIOS, are:

- **Abstraction for the OS.** The UEFI specification provides the interface between the platform firmware and the OS. The interfaces/API/protocols mark a clear boundary between the firmware and the OS.
- **Abstraction for devices and related code**. UEFI abstracts interfaces that make it possible to build code that works on a range of underlying hardware devices without having explicit knowledge of the specifics for each device in the range. This specification defines interfaces to platform capabilities including standard bus types such as PCI, USB, and SCSI. The list of supported bus types may grow over time, allowing code to utilize newer hardware through standard protocols without being rewritten.
- **Scalable platform environment.** The specification defines a complete solution for the firmware to describe all platform features and surface platform capabilities to the OS during the boot process. These definitions cover a range of the contemporary platform designs and the simple enough to be able to extend in the future.
- **OS Agnostic Rich Pre-Boot environment.** The UEFI spec defines extensible interfaces that enable creation of platform drivers. The UEFI drivers, analogous to OS drivers, provide support for new devices and may provide enhanced platform capabilities, such as firmware update, platform configuration, diagnostics and deployment services. The existence of networking, USB and file system capabilities adds to the richness of the pre-boot environment.
- **Consistent Configuration Infrastructure.** The UEFI spec defines a methodology of describing the platform configuring data in a standard way. The rendering of the data is left to the platform vendor. This allows UEFI to bring all the platform configurations like BIOS, Storage and Network options under a single setup application with a consistent navigation and look/feel.
- **GUID Partition Table.** The UEFI defines a new standard layout for the partition table known as GUID Partition Table (GPT). GPT provides a more flexible mechanism for partitioning disks than the older Master Boot Record (MBR) partitioning scheme that has been common to PCs. MBR disks support only four partition table entries and the partition size is limited to 2TB (2.20×10^{12} bytes). GPT scheme allows up to 128 primary partitions and can support partitions up to 9.4 Zettabytes (9.4×10^{21} bytes). There are some near-term limitations to 2 terabyte support due to device support, but once devices fully support GPT/UEFI, this will no longer be an issue (explained in more detail in the “Limitations” section). For more info on GPT, see *http://en.wikipedia.org/wiki/GUID_Partition_Table*.
- **Secure Boot.** The UEFI 2.2 (or later) specification brings security to the boot process by only loading the driver or OS loaders that are signed by a known/trusted digital signature. Secure boot keys are managed by the BIOS and OS. Secure boot can also be placed in a "Custom" mode, where additional public keys can be added by the platform administrator to allow execution of custom code or restriction of code that may be trusted by some, but not by the platform’s owner.

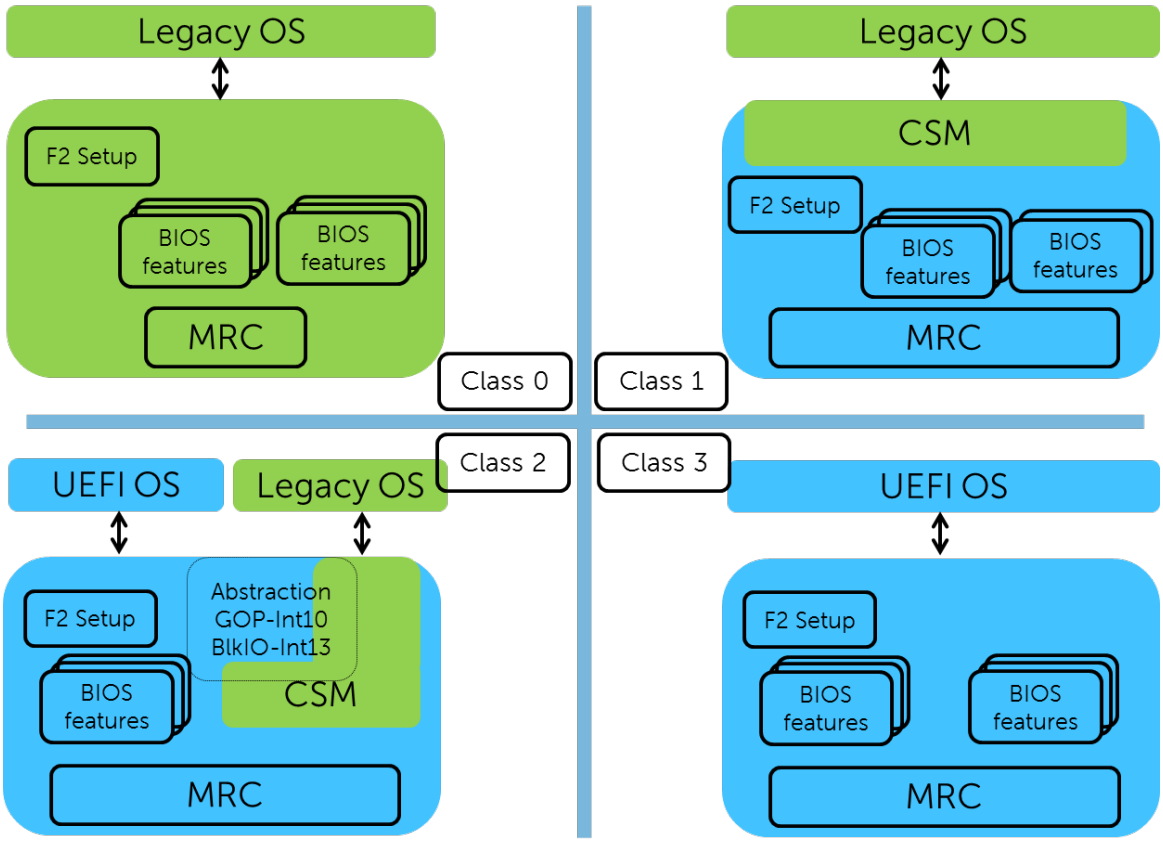
Drawing Parallels Between Legacy & UEFI

Below are some parallels to help distinguish between UEFI and legacy in how they exist and work.

	Legacy BIOS	**UEFI Firmware**
Programming Language	Assembly	C
Processors Supported	Intel Architecture	Intel Architecture, Itanium, and ARM officially supported by UEFI Spec
Processor Mode Used	Mostly 16-bit, Single CPU, Single Threaded	Native (64-bit or32-bit), Single CPU, Single Threaded
Expansion Card Firmware	Legacy Option ROMs	UEFI Drivers or UEFI Option ROMs
Provided Services	Interrupts	Protocols
Video Support	Int10h & VBIOS	Graphics Output Protocol (GOP)
Storage Support	Int13h, Master Boot Record (MBR) Partitioning	BlockIO Protocol, GUID Partition Table (GPT) and Master Boot Record (MBR) Partitioning
Peripheral & Feature Setup	F2 Setup, Ctrl-M, Ctrl-A No Industry Standard	UEFI Human Interface Infrastructure (HII) Protocol as Industry Standard
OS Boot Loader	Int19h loads 16-bit boot sector in MBR One boot loader per device	UEFI loads boot loader executable file(s) per priority/ordering defined by UEFI Spec. Multiple boot loader files, unique names/paths, can co-exist on the same partition/device.
OS Handoff	No clear definition	ExitBootServices() function defined by UEFI Spec.

UEFI Platform Classification

For better understanding the technology context and UEFI adaption progression the platforms can be classified in 4 classes.



- **Class 0:** Non UEFI platforms, the set of platforms based on traditional legacy BIOS. These platforms are not UEFI aware.
- **Class 1:** In the earlier days of EFI/UEFI when all the leading OSs were not EFI/UEFI aware a special Compatibility Support Module (CSM) was used to present the traditional BIOS like interface. These platforms only booted to traditional, legacy OSs,
- **Class 2:** These platforms came about when EFI was adapted as UEFI industry standard and OS started adding support for UEFI. These platforms support booting using the traditional method of int19h, where in BIOS loads the boot sector and hands of execution to the boot loader, as well as loading a UEFI boot loader application. Majority of platforms shipping today are Class 2 platforms.
- **Class 3:** These platforms support booting only using the UEFI defined method of loading the boot loader application from a specific location. Class 3 platforms do not sport a Compatibility Support Module. In fact any class 2 platform with CSM turned off functions like a Class 3 platform.

Booting UEFI

The way boot options work in UEFI mode differs from that of the legacy BIOS greatly. The UEFI boot option:

- Specifies a file name/path on a storage device as a boot target (vs. a drive as in legacy BIOS).
- Is automatically created by the operating system during installation and points to its own unique file.
- Is not tied to 1:1 to a partition and/or drive, and a partition can hold more than one boot loader.
- Removable media can utilize a default boot file path/file name (defined by the UEFI spec) to boot a file/OS previously unknown or installed on a system. This allows behavior like OS install from CD.
- Can be manually added as a file name/path by the user via the UEFI Boot Manager.

Automatic and Manual Boot Options

In UEFI boot mode, options are automatically added for removable devices (described in further detail below). OS installation also automatically adds a boot option that points to the OS Boot loader. The Boot options can also be manipulated manually by the user via the UEFI Boot Manager.

Multiple boot options per device, or per file, are allowed. One may want to have two boot options for the same file with different input parameters, such as a debug parameter.

Booting to a Removable Media Device

To make a removable device bootable the UEFI application simply needs to be renamed to BOOTx64.EFI (case insensitive) and placed in the \EFI\BOOT directory in a FAT32 partition.

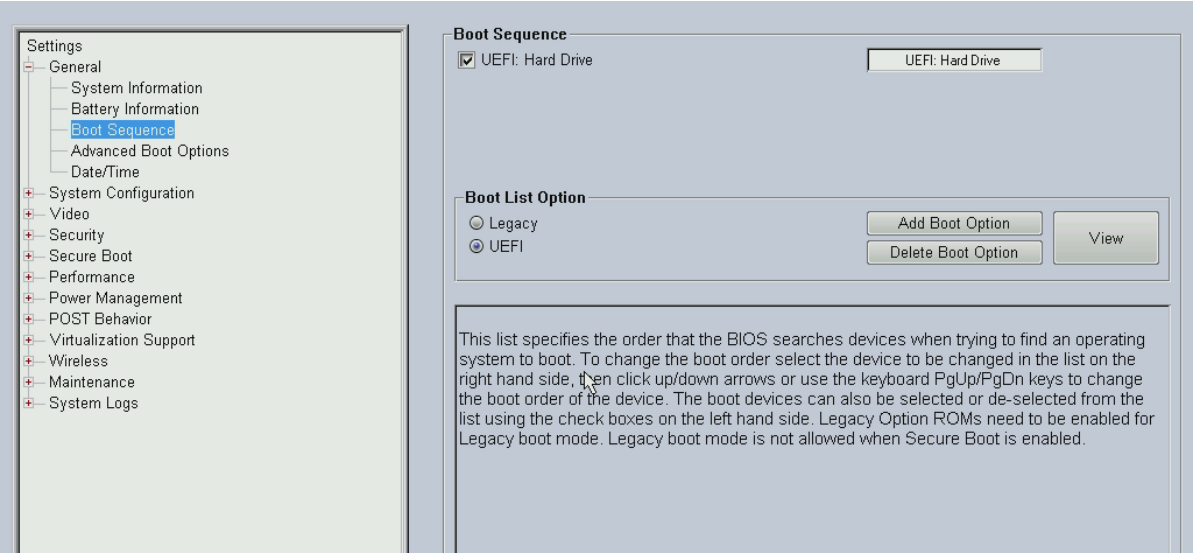
When a removable device such as a USB key is detected in UEFI Boot Mode, a boot option at the end of the current boot list is automatically added to point to the following location:

```
\EFI\BOOT\BOOTx64.EFI
```

Taking advantage of state of the art… running as UEFI only

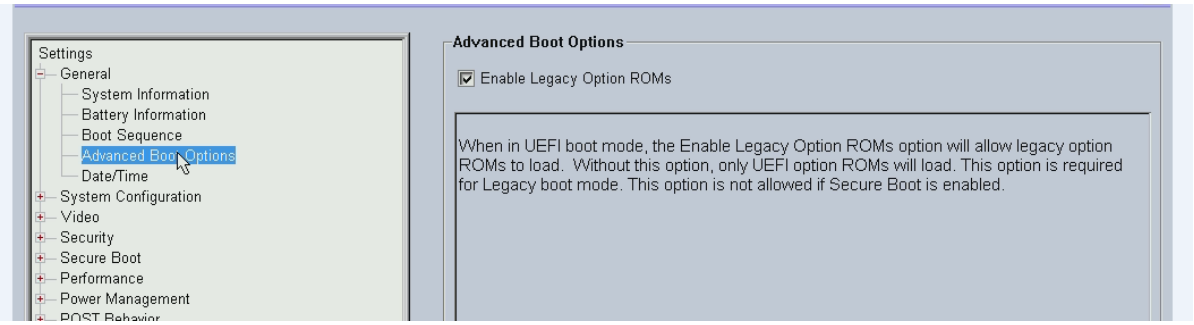
As UEFI is gaining more acceptance, Operating Systems like Win8 are capable of running in UEFI only mode without having to rely on any of the legacy OPROM or the abstraction based on those OPROM.

Figure 1 Selecting UEFI Boot mode



On the Dell systems this can be achieve by selecting UEFI boot mode and turning of legacy OPROM support.

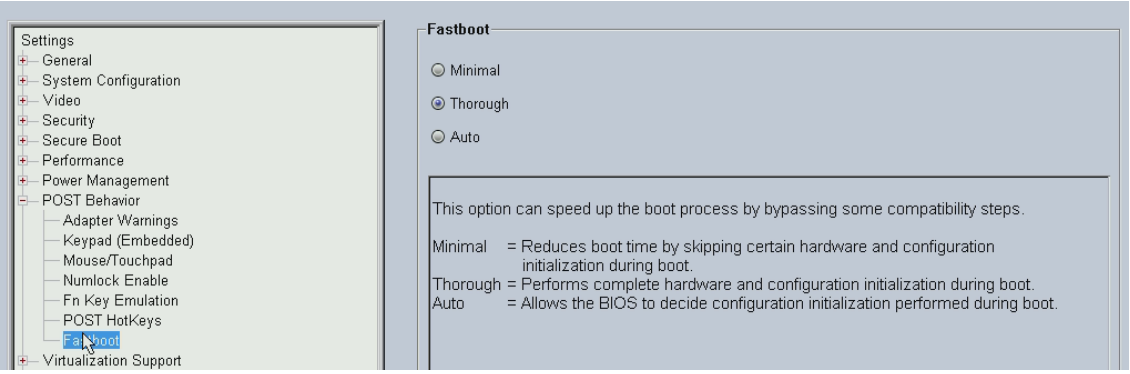
Figure 2: Enable/Disable Legacy Option ROMs



Optimized POST times

When platform is targeted to boot only in UEFI mode, the firmware can skip initialization of the devices that do take direct part in the boot process. Instead these devices get initialized inside the OS. What previously required up to 10 seconds to do inside the firmware and now be done in as little as two seconds. Operating Systems, like Win8, take advantage of reduced operations when loading the OS through UEFI and are also able to provide must faster boot speeds.

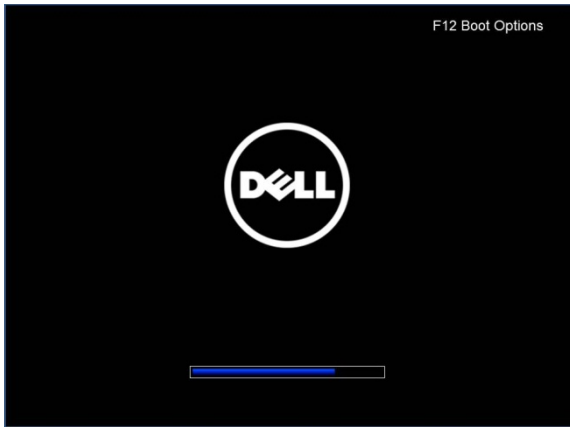
Figure 3: BIOS Fast Boot Setting



Seamless Boot experience

In the UEFI only mode video initialization is done by the UEFI drivers that support Graphics Output Protocol. This allows BIOS to enable video in the native mode and because BIOS does not have to break the flow to give control to any of the legacy oproms the video remains in the same mode all throughout the boot process.

Figure 4: Seamless Boot Splash Screen

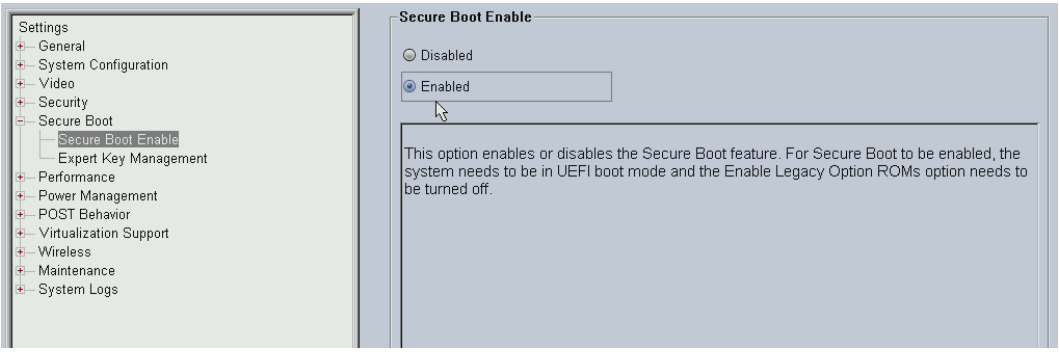


Since the BIOS POST is now capable of native graphics in the UEFI only mode the OSs like Win8 are able to render the boot loader application in graphics mode.

Secure boot

Running UEFI only enables platform with more secure boot process by only loading the driver or OS loaders that are signed by a known/trusted digital signature. Secure Boot is managed by both the UEFI BIOS and a UEFI aware OS by storing settings/data/keys in signed/secure storage controlled by the UEFI BIOS. When Secure Boot is enabled, which is the default state on Win8, the platform has already set certain keys in the signed/secure storage. Turning on secure boot also enforces UEFI only booting by disabling legacy OPROM.

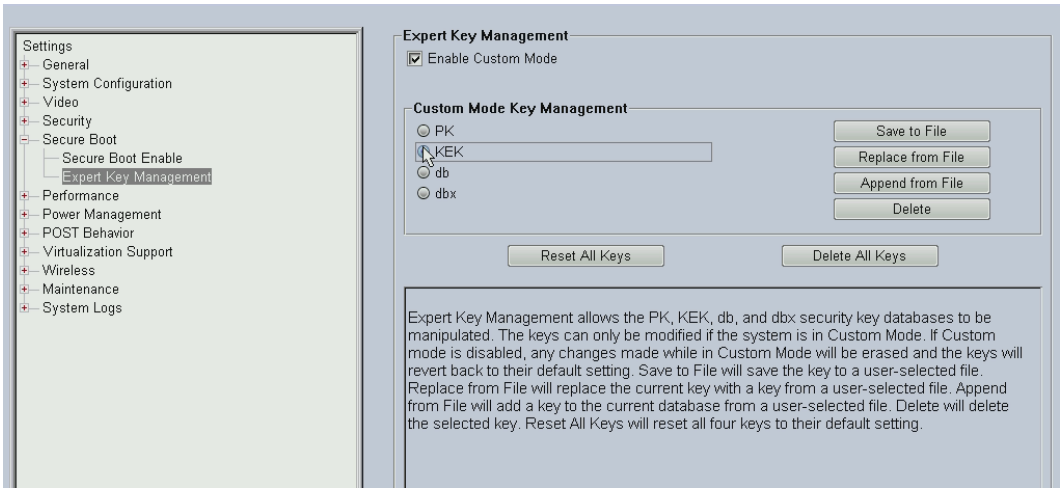
Figure 5: Secure Boot Enable



To change/add/remove keys from the platform from outside the UEFI BIOS, or to sign a driver/loaded so that it can run on the platform, access to the private key for one of the already trusted public keys is required.

Secure boot can also be placed in a "Custom" mode, where additional public keys can be added/changed/removed by the user (only from an expert interface within the UEFI Setup Menu) to allow the owner of the machine to allow/restrict execution of custom drivers/apps/OSs as the owner sees fit.

Figure 6: Custom Key Management



Secure Boot is a groundbreaking evolution in bringing powerful security to pre-boot execution. Yet at the end of the day, the customer is control of their PC, empowering IT departments to make the choices that are right for their environment. Secure Boot is not required to have the machine operate. Any platform owner can turn Secure Boot off (if it was shipped with it on) and easily install a Secure Boot non-aware OS (ex: Windows XP). The security that Secure Boot brings is a benefit that, if possible, should be left on to help protect your platform.

Impact on the legacy cards

Running UEFI only means all the peripheral device/card in the platform needs to support UEFI pre-boot drivers. Otherwise that card would not be available during POST and the boot process with boot mode

set to UEFI and legacy OPROM disabled. Enabling secure boot enforces boot mode to UEFI and disables legacy OPROM

If a platform features a non-UEFI aware storage card then platform would not boot from the storage devices connected to the card. Similarly a video device that does not have support for the UEFI driver cannot be used as a primary video during POST.

Boot mode

With platform capable of booting to legacy or UEFI, and with features like secure boot that put restrictions on the way platform could boot there are certain rules that govern the boot mode selection. Following table

Secure Boot	Secure Boot Off			Secure Boot ON
Boot Mode	Legacy Boot Mode	UEFI Boot Mode		*UEFI Boot Mode Required*
Option Roms Being Loaded	Legacy Oproms Always	Legacy Oproms	UEFI drivers	*UEFI drivers Required*
Type of PXE Network Boot Supported (Default: NIC "Enabled+PXE", UEFI Network Stack Disabled)	Legacy PXE Only	Legacy PXE Only	UEFI PXE Only	UEFI PXE Only
What's Shown in F12 Boot Menu (with Factory Default)	Legacy and UEFI OS's, and Legacy PXE	Legacy and UEFI OS's, and Legacy PXE **	UEFI OS's Only (No PXE) *	UEFI OS's Only (No PXE) *
What's in the Normal Boot Order (with Factory Default)	Legacy OS's & Legacy PXE (if on)	UEFI OS's (No PXE) ***	UEFI OS's (No PXE) *	UEFI OS's (No PXE) *
Example OS's that Can Run In This Configuration	Windows XP/Vista/7/8, Linux, DOS	Windows Vista/7/8, Some Linux Versions	Windows 8, Some Linux Versions	Windows 8, Linux Support Limited

* Until "UEFI Network Stack" is Enabled, UEFI PXE is not active and therefore the NIC acts as though the setting was just "Enabled".

** PXE is supported from an Oprom, therefore the type of PXE supported is directly related to the type of option roms currently being loaded.

*** The boot list used for normal booting is allowed to only consist of the boot options from the current boot mode the machine is in. In "UEFI Boot Mode", no automatic booting to Legacy PXE, but other methods through DA tokens and F12 can result in Legacy PXE boot in this configuration.

January 2013 | Rev 0.1