intel

Extensible Firmware Interface Specification, Version 1.10

Specification Update

Version -001

November 26, 2003

The *Extensible Firmware Interface Specification*, version 1.10, may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

Revision History

Revision	Revision History	Date
-001	This document is the first Specification Update for the EFI Specification, version 1.10 (final).	11/26/03

Preface

This document is an update to the *Extensible Firmware Interface Specification*, version 1.10. This document is a compilation of documentation errata and specification clarifications and changes. It is intended for software developers of applications, operating systems, or tools.

We have endeavored to include all documented errata in the consolidation process; however, we make no representations or warranties concerning the completeness of the Specification Update. This document may also contain information that was not previously published.

Nomenclature

Specification Changes are modifications to the current published specifications. These changes will be incorporated in the next release of the specifications.

Errata are design defects or errors that may cause EFI's behavior to deviate from published specifications.

Specification Clarifications and Corrections describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications and corrections will be incorporated in the next release of the specifications.

Documentation Changes include typos, errors, or omissions from the current published specifications.

Contents

General Information

This document contains the errata, specification changes and clarifications, and documentation changes to the *Extensible Firmware Interface Specification,* version 1.10 (hereafter referred to as the "EFI 1.10 Specification"). This document contains clarifications and corrections to the *EFI 1.10 Specification* based on industry feedback.

This document is broken into the following sections:

- Specification changes
- Errata
- Specification clarifications and corrections
- Documentation changes

The specific changes in each section are numbered and grouped by chapter in the *EFI 1.10 Specification.* In each subsection, changes are listed in order by page number. If a page number is not listed, then no changes were made to the *EFI 1.10 Specification* on that page.

This document is intended to be used along with the *EFI 1.10 Specification,* which is available from the EFI Web site at:
http://developer.intel.com/technology/efi/

Summary Table of Changes

The following table indicates the specification changes, errata, specification clarifications and corrections, or documentation changes that apply to the *EFI 1.10 Specification.*

In the table below and throughout this document, the identification number for the change is in the format *X-N.m*, where *X-N* is the page number (using the chapter-page numbering format; for example, page 6-4 is the fourth page in chapter 6) and *m* indicates the specific erratum number. New errata that are found and included in subsequent releases of this document will be added sequentially by page number and shaded in the table below.

Table 1. Summary of Changes to *EFI 1.10 Specification*

No.	Plans	Specification Changes
E-1.1	Plan fix	Delete the note in section E.1 (page E-1)
E-8.1	Plan fix	Change the Minor field definition in Table E-4 (page E-8)
No.	**Plans**	**Errata**
		None
No.	**Plans**	**Specification Clarifications and Corrections**
2-10.1	Plan fix	Change the last bullet in section 2.3.2 (page 2-10)
3-6.1	Plan fix	Change the second paragraph in section 3.2 (page 3-6)
3-6.2	Plan fix	Change the fourth paragraph in section 3.2 (page 3-6)
4-8.1	Plan fix	Change the `InstallMultipleProtocolInterfaces` field in "Related Definitions" (page 4-8)

continued

Table 1. Summary of Changes to *EFI 1.10 Specification* (continued)

No.	Plans	Specification Clarifications and Corrections (continued)
5-4.1	Plan fix	Add lines to the TPL restrictions in Table 5-3 (page 5-4)
5-6.1	Plan fix	Change the description of **EVT_NOTIFY_WAIT** in "Related Definitions" (page 5-6)
5-6.2	Plan fix	Change the description of **EVT_NOTIFY_SIGNAL** in "Related Definitions" (page 5-6)
5-6.3	Plan fix	Change the description of **EVT_SIGNAL_EXIT_BOOT_SERVICES** in "Related Definitions" (page 5-6)
5-6.4	Plan fix	Change the description of **EVT_SIGNAL_VIRTUAL_ADDRESS_CHANGE** in "Related Definitions" (page 5-6)
5-8.1	Plan fix	Change the last paragraph of the "Description" subsection (page 5-8)
5-8.2	Plan fix	Change the "Status Codes Returned" table (page 5-8)
5-10.1	Plan fix	Change the "Description" subsection (page 5-10)
5-11.1	Plan fix	Change the "Description" subsection in its entirety (page 5-11)
5-11.2	Plan fix	Change the **EFI_INVALID_PARAMETER** entry in the "Status Codes Returned" table (page 5-11)
5-12.1	Plan fix	Change the third bullet in the "Description" subsection (page 5-12)
5-13.1	Plan fix	Change the description of *TriggerTime* in the "Parameters" subsection (page 5-13)
5-28.1	Plan fix	Change the "Status Codes Returned" table (page 5-28)
5-36.1	Plan fix	Change the "Summary" paragraph (page 5-36)
5-37.1	Plan fix	Change the "Status Codes Returned" table (page 5-37)
5-38.1	Plan fix	Change the "Summary" paragraph (page 5-38)
5-39.1	Plan fix	Change the "Status Codes Returned" table (page 5-39)
5-41.1	Plan fix	Change the "Status Codes Returned" table (page 5-41)
5-42.1	Plan fix	Change the "Status Codes Returned" table (page 5-42)
5-44.1	Plan fix	Change the "Status Codes Returned" table (page 5-44)
5-46.1	Plan fix	Change the "Status Codes Returned" table (page 5-46)
5-48.1	Plan fix	Change the "Status Codes Returned" table (page 5-48)
5-61.1	Plan fix	Change the description of *DriverImageHandle* in the "Parameters" subsection (page 5-61)
5-62.1	Plan fix	Change the description of Context Override in the "Description" subsection (page 5-62)
5-67.1	Plan fix	Add an entry to the "Status Codes Returned" table (page 5-67)
5-79.1	Plan fix	Change the "Status Codes Returned" table (page 5-79)
5-83.1	Plan fix	Change the "Summary" paragraph (page 5-83)
5-83.2	Plan fix	Delete the second sentence in the first paragraph of the "Description" subsection (page 5-83)

continued

Table 1. Summary of Changes to *EFI 1.10 Specification* (continued)

No.	Plans	Specification Clarifications and Corrections (continued)
5-84.1	Plan fix	Change the “Status Codes Returned” table (page 5-84)
5-85.1	Plan fix	Change the first paragraph of the “Description” subsection (page 5-85)
5-91.1	Plan fix	Change the “Status Codes Returned” table (page 5-91)
6-5.1	Plan fix	Change the third paragraph in the “Description” subsection (page 6-5)
6-17.1	Plan fix	Change the first paragraph in the “Description” subsection (page 6-17)
6-19.1	Plan fix	Change the “Status Codes Returned” table (page 6-19)
8-11.1	Plan fix	Change the description of the Length field in Table 8-14 (page 8-11)
9-19.1	Plan fix	Delete the last sentence from the first paragraph in the “Description” subsection (page 9-19)
9-37.1	Plan fix	Change the “Status Codes Returned” table (page 9-37)
9-39.1	Plan fix	Change the “Status Codes Returned” table (page 9-39)
9-42.1	Plan fix	Change the “Status Codes Returned” table (page 9-42)
9-46.1	Plan fix	Change the “Status Codes Returned” table (page 9-46)
9-51.1	Plan fix	Change the “Status Codes Returned” table (page 9-51)
10-17.1	Plan fix	Change the second paragraph of the “Description” subsection (page 10-17)
10-19.1	Plan fix	Change the description of ***Attribute*** in the “Parameters” subsection (page 10-19)
10-20.1	Plan fix	Delete **EFI_UNSUPPORTED** from the “Status Codes Returned” table (page 10-20)
10-28.1	Plan fix	Change the “Status Codes Returned” table (page 10-28)
10-33.1	Plan fix	Change the **EfiUgaVideoFill** entry in Table 10-4 (page 10-33)
11-3.1	Plan fix	Change the **EFI_NO_SUCH_MEDIA** return code (page 11-3)
11-9.1	Plan fix	Change the description of the Revision field in Table 11-1 (page 11-9)
11-26.1	Plan fix	Change the “Status Codes Returned” table (page 11-26)
11-27.1	Plan fix	Change the “Status Codes Returned” table (page 11-27)
11-28.1	Plan fix	Change the “Status Codes Returned” table (page 11-28)
11-29.1	Plan fix	Change the “Status Codes Returned” table (page 11-29)
11-31.1	Plan fix	Change the “Description” subsection in its entirety (page 11-31)
11-31.2	Plan fix	Change the “Status Codes Returned” table (page 11-31)
11-58.1	Plan fix	Change the description of the **StrToFat()** API (page 11-58)
12-11.1	Plan fix	Add three attributes and descriptions to the list of PCI Root Bridge I/O Protocol Attribute bits in the “Related Definitions” subsection (page 12-11)
12-12.1	Plan fix	Change the descriptions of the IDE attributes in the “Related Definitions” subsection (page 12-12)
12-60.1	Plan fix	Add three attributes and descriptions to the list of PCI Root Bridge I/O Protocol Attribute bits in the “Related Definitions” subsection (page 12-60)

continued

Table 1. Summary of Changes to *EFI 1.10 Specification* (continued)

No.	Plans	Specification Clarifications and Corrections (continued)
12-61.1	Plan fix	Change the descriptions of the IDE attributes in the “Related Definitions” subsection (page 12-61)
12-70.1	Plan fix	Change the first sentence in the last paragraph of the “Description” subsection (page 12-70)
12-80.1	Plan fix	Delete the status code **EFI_INVALID_PARAMETER** from the “Status Codes Returned” table (page 12-80)
14-20.1	Plan fix	Change the fourth item in the numbered list in the “Description” subsection (page 14-20)
15-13.1	Plan fix	Change the “Status Codes Returned” table (page 15-13)
15-16.1	Plan fix	Change the “Status Codes Returned” table (page 15-16)
15-17.1	Plan fix	Change the “Status Codes Returned” table (page 15-17)
15-50.1	Plan fix	Change the *BufferSize* parameter in the “Prototype” subsection from **UINTN** to **UINT64** (page 15-50)
17-18.1	Plan fix	Change the “Status Codes Returned” table (page 17-18)
18-3.1	Plan fix	Change the definition of **EFI_IO_WIDTH** in the “Related Definitions” subsection (page 18-3)
18-5.1	Plan fix	Change the description of *Buffer* in the “Parameters” subsection (page 18-5)
19-61.1	Plan fix	Replace “PE32+” with “PE32” (page 19-61)
19-62.1	Plan fix	Replace “PE32+” with “PE32” (page 19-62)
19-65.1	Plan fix	Replace “PE32+” with “PE32” (page 19-65)
19-68.1	Plan fix	Replace “PE32+” with “PE32” (page 19-68)
19-74.1	Plan fix	Replace “PE32+” with “PE32” (page 19-74)
B-3.1	Plan fix	Change the line for Set Mode 80x25 in Table B-2 (page B-3)
E-3.1	Plan fix	Update four RFC numbers in Table E-2 (page E-3)
E-3.2	Plan fix	Change “BC protocol” to “PXE Base Code Protocol” in Table E-2 (page E-3)
E-4.1	Plan fix	Change “BC protocol” to “PXE Base Code Protocol” in Table E-2 (page E-4)
E-32.1	Plan fix	Replace “RFC 1700” with “RFC 3232” (page E-32)
E-38.1	Plan fix	Change the direction of the Shutdown and Stop state transition arrows in Figure E-6 (page E-38)
E-41.1	Plan fix	Replace the last two paragraphs in section E.4.2 (page E-41)
E-44.1	Plan fix	Replace the “Preparing the CPB” subsection in its entirety (page E-44)
E-52.1	Plan fix	Change three **#define** statements (page E-52)
Gloss-4.1	Plan fix	Replace “PE32+” with “PE32” in the EBC Image definition (page Glossary-4)

continued

Table 1. Summary of Changes to *EFI 1.10 Specification* (continued)

No.	Plans	Documentation Changes
3-5.1	Doc	Delete the hyphen in the heading text of section 3.2 (page 3-5)
9-42.1	Doc	Change “attempt” to “attempting” in the EFI_DEVICE_ERROR status code description (page 9-42)
12-96.1	Doc	Change “being in” to “begin on” in the tenth bullet (page 12-96)
12-96.2	Doc	Change “is” to “in” in the first sentence of the twelfth bullet (page 12-96)

Codes used in summary table:

Doc: Document change or update that will be implemented.

Plan fix: This erratum may be fixed in a future revision of the specification.

Fixed: This erratum has been fixed previously.

No fix: There are no plans to fix this erratum.

Appendix E: 32/64-Bit UNDI Specification

E-1.1 Delete the note in section E.1 (page E-1)

Delete the note in section E.1, “Introduction.” This version of the *EFI Specification* is the release version of the 32/64-bit UNDI Specification.

E-8.1 Change the Minor field definition in Table E-4 (page E-8)

In section E.2.1, “32/64-bit UNDI Interface,” change the Minor field definition in Table E-4 FROM:

Minor	0x00	UNDI command interface minor revision

TO:

Minor	Varies	UNDI command interface. Minor revision number. **0x00 (Alpha):** This version of UNDI does not operate as a runtime driver. The callback interface defined in the UNDI Start command is required. **0x10 (Beta):**. This version of UNDI can operate as an OS runtime driver. The callback interface defined in the UNDI Start command is required.

Errata

There are currently no errata in the *EFI 1.10 Specification.*

Specification Clarifications and Corrections

Chapter 2: Overview

2-10.1 Change the last bullet in section 2.3.2 (page 2-10)

In section 2.3.2, “IA-32 Platforms,” change the last bullet FROM:

- ACPI tables loaded at runtime must be contained in memory of type **EfiACPIMemoryNVS** or **EfiFirmareReserved**. The cacheability attributes for ACPI tables loaded at runtime (via ACPI LoadTable) should be defined in the EFI memory map. If no information about the table location exists in the EFI memory map, the table is assumed to be noncached.

TO:

- ACPI tables loaded at runtime must be contained in memory of type **EfiACPIMemoryNVS** or **EfiReservedMemoryType**. The cacheability attributes for ACPI tables loaded at runtime (via ACPI LoadTable) should be defined in the EFI memory map. If no information about the table location exists in the EFI memory map, the table is assumed to be noncached.

Chapter 3: Boot Manager

3-6.1 Change the second paragraph in section 3.2 (page 3-6)

In section 3.2, “Globally-Defined Variables,” change the second paragraph FROM:

The *Lang* variable contains the 3-character (8-bit ASCII characters) ISO-639-2 language code that the machine has been configured for. This value may be changed to any value supported by *LangCodes*; however, the change does not take effect until the next boot. If the language code is set to an unsupported value, the firmware will choose a supported default at initialization and set *Lang* to a supported value.

TO:

The *Lang* variable contains the 3-character (8-bit ASCII characters) ISO-639-2 language code that the machine has been configured for. This value may be changed to any value supported by *LangCodes*. If this change is made in the preboot environment, then the change will take effect immediately. If this change is made at OS runtime, then the change does not take effect until the next boot. If the language code is set to an unsupported value, the firmware will choose a supported default at initialization and set *Lang* to a supported value.

3-6.2 Change the fourth paragraph in section 3.2 (page 3-6)

In section 3.2, “Globally-Defined Variables,” change the fourth paragraph FROM:

The *ConIn*, *ConOut*, and *ErrOut* variables each contain an **EFI_DEVICE_PATH** descriptor that defines the default device to use on boot. Changes to these values do not take

effect until the next boot. If the firmware cannot resolve the device path, it is allowed to automatically replace the value(s) as needed to provide a console for the system.

TO:

The *ConIn*, *ConOut*, and *ErrOut* variables each contain an **EFI_DEVICE_PATH** descriptor that defines the default device to use on boot. Changes to these values made in the preboot environment take effect immediately. Changes to these values at OS runtime do not take effect until the next boot. If the firmware cannot resolve the device path, it is allowed to automatically replace the value(s) as needed to provide a console for the system.

Chapter 4: EFI System Table

4-8.1 Change the *IninstallMultipleProtocolInterfaces* field in "Related Definitions" (page 4-8)

In the "Related Definitions" subsection in section 4.4, "EFI Boot Services Table," change the following field of the EFI Boot Services Table FROM:

```
EFI_UNINSTALL_MULTIPLE_PROTOCOL_INTERFACES
                IninstallMultipleProtocolInterfaces;
```

TO:

```
EFI_UNINSTALL_MULTIPLE_PROTOCOL_INTERFACES
                UninstallMultipleProtocolInterfaces;
```

Chapter 5: Boot Services

5-4.1 Add lines to the TPL restrictions in Table 5-3 (page 5-4)

In section 5.1, "Event, Timer, and Task Priority Services," add the following lines to the TPL restrictions in Table 5-3, before the line with WaitForEvent():

CreateEvent()	<	**TPL_HIGH_LEVEL**
CloseEvent()	<	**TPL_HIGH_LEVEL**
CheckEvent()	<	**TPL_HIGH_LEVEL**
SetTimer()	<	**TPL_HIGH_LEVEL**

5-6.1 Change the description of **EVT_NOTIFY_WAIT** in "Related Definitions" (page 5-6)

In section 5.1, "Event, Timer, and Task Priority Services – CreateEvent()," change the description of **EVT_NOTIFY_WAIT** in the "Related Definitions" subsection FROM:

EVT_NOTIFY_WAIT The event's *NotifyFunction* is to be invoked whenever the event is being waited on via **WaitForEvent()** or **CheckEvent()**.

TO:

EVT_NOTIFY_WAIT If an event of this type is not already in the signaled state, then the event's *NotificationFunction* will be queued at the event's *NotifyTpl* whenever the event is being waited on via **WaitForEvent()** or **CheckEvent()**.

5-6.2 Change the description of EVT_NOTIFY_SIGNAL in "Related Definitions" (page 5-6)

In section 5.1, "Event, Timer, and Task Priority Services – CreateEvent()," change the description of **EVT_NOTIFY_SIGNAL** in the "Related Definitions" subsection FROM:

EVT_NOTIFY_SIGNAL

The event's *NotifyFunction* is to be invoked whenever the event is signaled via **SignalEvent()**.

TO:

EVT_NOTIFY_SIGNAL The event's *NotifyFunction* is queued whenever the event is signaled.

5-6.3 Change the description of EVT_SIGNAL_EXIT_BOOT_SERVICES in "Related Definitions" (page 5-6)

In section 5.1, "Event, Timer, and Task Priority Services – CreateEvent()," change the description of **EVT_SIGNAL_EXIT_BOOT_SERVICES** in the "Related Definitions" subsection FROM:

EVT_SIGNAL_EXIT_BOOT_SERVICES

This event is to be notified by the system when **ExitBootServices()** is invoked. This type cannot be used with any other EVT bit type. The notification function for this event is not allowed to use the Memory Allocation Services, or call any functions that use the Memory Allocation Services, because these services modify the current memory map.

TO:

EVT_SIGNAL_EXIT_BOOT_SERVICES

This event is to be notified by the system when **ExitBootServices()** is invoked. This event is of type **EVT_NOTIFY_SIGNAL** and should not be combined with any other event types. The notification function for this event is not allowed to use the Memory Allocation Services, or call any functions that use the Memory Allocation Services, because these services modify the current memory map.

5-6.4 Change the description of EVT_SIGNAL_VIRTUAL_ADDRESS_CHANGE in "Related Definitions" (page 5-6)

In section 5.1, "Event, Timer, and Task Priority Services – CreateEvent()," change the description of **EVT_SIGNAL_VIRTUAL_ADDRESS_CHANGE** in the "Related Definitions" subsection FROM:

EVT_SIGNAL_VIRTUAL_ADDRESS_CHANGE

The event is to be notified by the system when **SetVirtualAddressMap()** is performed. This type cannot be used with any other EVT bit type. See the discussion of **EVT_RUNTIME**.

TO:

EVT_SIGNAL_VIRTUAL_ADDRESS_CHANGE

The event is to be notified by the system when **SetVirtualAddressMap()** is performed. This event type is a composite of **EVT_NOTIFY_SIGNAL**, **EVT_RUNTIME**, and **EVT_RUNTIME_CONTEXT** and should not be combined with any other event types.

5-8.1 Change the last paragraph of the "Description" subsection (page 5-8)

In section 5.1, "Event, Timer, and Task Priority Services – CreateEvent()," change the last paragraph of the "Description" subsection FROM:

The **EVT_NOTIFY_WAIT** and **EVT_NOTIFY_SIGNAL** flags are exclusive. If neither flag is specified, the caller does not require any notification concerning the event and the *NotifyTpl*, *NotifyFunction*, and *NotifyContext* parameters are ignored. If **EVT_NOTIFY_WAIT** is specified, then the event is signaled and its notify function is queued whenever a consumer of the event is waiting for it (via **WaitForEvent()** or **CheckEvent()**). If the **EVT_NOTIFY_SIGNAL** flag is specified then the event's notify function is queued whenever the event is signaled.

TO:

The **EVT_NOTIFY_WAIT** and **EVT_NOTIFY_SIGNAL** flags are exclusive. If neither flag is specified, the caller does not require any notification concerning the event and the *NotifyTpl*, *NotifyFunction*, and *NotifyContext* parameters are ignored. If **EVT_NOTIFY_WAIT** is specified and the event is not in the signaled state, then its notify function is queued whenever a consumer of the event is waiting for it (via **WaitForEvent()** or **CheckEvent()**). If the **EVT_NOTIFY_SIGNAL** flag is specified, then the event's notify function is queued whenever the event is signaled.

5-8.2 Change the "Status Codes Returned" table (page 5-8)

In section 5.1, "Event, Timer, and Task Priority Services – CreateEvent()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The event structure was created.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.
EFI_OUT_OF_RESOURCES	The event could not be allocated.

TO:

EFI_SUCCESS	The event structure was created.
EFI_INVALID_PARAMETER	*Event* is **NULL**.
EFI_INVALID_PARAMETER	*Type* has an unsupported bit set.
EFI_INVALID_PARAMETER	*Type* has both **EVT_NOTIFY_SIGNAL** and **EVT_NOTIFY_WAIT** set.
EFI_INVALID_PARAMETER	*Type* has either **EVT_NOTIFY_SIGNAL** or **EVT_NOTIFY_WAIT** set and *NotifyFunction* is **NULL**.
EFI_INVALID_PARAMETER	Type has either **EVT_NOTIFY_SIGNAL** or **EVT_NOTIFY_WAIT** set and *NotifyTpl* is not a supported TPL level.
EFI_OUT_OF_RESOURCES	The event could not be allocated.

5-10.1 Change the "Description" subsection (page 5-10)

In section 5.1, "Event, Timer, and Task Priority Services – SignalEvent()," change the "Description" subsection FROM:

The supplied *Event* is signaled and, if the event has a signal notification function, it is scheduled to be invoked at the event's notification task priority level. **SignalEvent()** may be invoked from any task priority level.

TO:

The supplied *Event* is placed in the signaled state. If *Event* is already in the signaled state, then **EFI_SUCCESS** is returned. If *Event* is of type **EVT_NOTIFY_SIGNAL**, then the event's notification function is scheduled to be invoked at the event's notification task priority level. **SignalEvent()** may be invoked from any task priority level.

5-11.1 Change the "Description" subsection in its entirety (page 5-11)

In section 5.1, "Event, Timer, and Task Priority Services – WaitForEvent()," change the "Description" subsection in its entirety to the following:

This function must be called at priority level **TPL_APPLICATION**. If an attempt is made to call it at any other priority level, **EFI_UNSUPPORTED** is returned.

The list of events in the *Event* array are evaluated in order from first to last, and this evaluation is repeated until an event is signaled or an error is detected. The following checks are performed on each event in the *Event* array.

- If an event if of type **EVT_NOTIFY_SIGNAL**, then **EFI_INVALID_PARAMETER** is returned and *Index* indicates the event that caused the failure.
- If an event is in the signaled state, the signaled state is cleared and **EFI_SUCCESS** is returned, and *Index* indicates the event that was signaled.
- If an event is not in the signaled state but does have a notification function, the notification function is queued at the event's notification task priority level. If the execution of the event's notification function causes the event to be signaled, then the signaled state is cleared, **EFI_SUCCESS** is returned, and *Index* indicates the event that was signaled.

To wait for a specified time, a timer event must be included in the *Event* array.

To check if an event is signaled without waiting, an already signaled event can be used as the last event in the list being checked, or the **CheckEvent()** interface may be used.

5-11.2 Change the EFI_INVALID_PARAMETER entry in the "Status Codes Returned" table (page 5-11)

In section 5.1, "Event, Timer, and Task Priority Services – WaitForEvent()," change the entry for **EFI_INVALID_PARAMETER** in the "Status Codes Returned" table FROM:

EFI_INVALID_PARAMETER	The event indicated by *Index* has a notification function or *Event* was not a valid type.

TO:

EFI_INVALID_PARAMETER	*NumberOfEvents* is 0.
EFI_INVALID_PARAMETER	The event indicated by *Index* is of type **EVT_NOTIFY_SIGNAL**.

5-12.1 Change the third bullet in the "Description" subsection (page 5-12)

In section 5.1, "Event, Timer, and Task Priority Services – CheckEvent()," change the third bullet in the "Description" subsection FROM:

- If *Event* is not in the signaled state but does have a notification function, the function is executed. If that causes *Event* to be signaled, it is cleared and **EFI_SUCCESS** is returned; if it does not cause *Event* to be signaled, **EFI_NOT_READY** is returned.

TO:

- If *Event* is not in the signaled state but does have a notification function, the notification function is queued at the event's notification task priority level. If the execution of the notification function causes *Event* to be signaled, then the signaled state is cleared and **EFI_SUCCESS** is returned; if the *Event* is not signaled, then **EFI_NOT_READY** is returned.

5-13.1 Change the description of *TriggerTime* in the "Parameters" subsection (page 5-13)

In section 5.1, "Event, Timer, and Task Priority Services – SetTimer()," change the description of *TriggerTime* in the "Parameters" subsection FROM:

TriggerTime	The number of 100ns units until the timer expires.

TO:

TriggerTime	The number of 100ns units until the timer expires. A *TriggerTime* of 0 is legal. If *Type* is **TimerRelative** and *TriggerTime* is 0, then the timer event will be signaled on the next timer tick. If *Type* is **TimerPeriodic** and *TriggerTime* is 0, then the timer event will be signaled on every timer tick.

5-28.1 Change the "Status Codes Returned" table (page 5-28)

In section 5.2, "Memory Allocation Services – GetMemoryMap()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The memory map was returned in the *MemoryMap* buffer.
EFI_BUFFER_TOO_SMALL	The *MemoryMap* buffer was too small. The current buffer size needed to hold the memory map is returned in *MemoryMapSize*.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.

TO:

EFI_SUCCESS	The memory map was returned in the *MemoryMap* buffer.
EFI_BUFFER_TOO_SMALL	The *MemoryMap* buffer was too small. The current buffer size needed to hold the memory map is returned in *MemoryMapSize*.
EFI_INVALID_PARAMETER	*MemoryMapSize* is **NULL**.
EFI_INVALID_PARAMETER	The *MemoryMap* buffer is not too small and *MemoryMap* is **NULL**.

5-36.1 Change the "Summary" paragraph (page 5-36)

In section 5.3, "Protocol Handler Services – InstallProtocolInterface()," change the "Summary" paragraph FROM:

Installs a protocol interface on a device handle. If the handle does not exist, it is created and added to the list of handles in the system.

TO:

Installs a protocol interface on a device handle. If the handle does not exist, it is created and added to the list of handles in the system.
InstallMultipleProtocolInterfaces() performs more error checking than **InstallProtocolInterface()**, so it is recommended that

InstallMultipleProtocolInterfaces() be used in place of **InstallProtocolInterface()**.

5-37.1 Change the "Status Codes Returned" table (page 5-37)

In section 5.3, "Protocol Handler Services – InstallProtocolInterface()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The protocol interface was installed.
EFI_OUT_OF_RESOURCES	Space for a new handle could not be allocated.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.

TO:

EFI_SUCCESS	The protocol interface was installed.
EFI_OUT_OF_RESOURCES	Space for a new handle could not be allocated.
EFI_INVALID_PARAMETER	*Handle* is **NULL**.
EFI_INVALID_PARAMETER	*Protocol* is **NULL**.
EFI_INVALID_PARAMETER	*InterfaceType* is not **EFI_NATIVE_INTERFACE**.
EFI_INVALID_PARAMETER	*Protocol* is already installed on the handle specified by *Handle*.

5-38.1 Change the "Summary" paragraph (page 5-38)

In section 5.3.1, "Protocol Handler Services – UninstallProtocolInterface()," change the "Summary" paragraph FROM:

Removes a protocol interface from a device handle.

TO:

Removes a protocol interface from a device handle. It is recommended that **UninstallMultipleProtocolInterfaces()** be used in place of **UninstallProtocolInterface()**.

5-39.1 Change the "Status Codes Returned" table (page 5-39)

In section 5.3, "Protocol Handler Services – UninstallProtocolInterface()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The interface was removed.
EFI_NOT_FOUND	The interface was not found.
EFI_ACCESS_DENIED	The interface was not removed because the interface is still being used by a driver.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.

TO:

EFI_SUCCESS	The interface was removed.
EFI_NOT_FOUND	The interface was not found.
EFI_ACCESS_DENIED	The interface was not removed because the interface is still being used by a driver.
EFI_INVALID_PARAMETER	*Handle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*Protocol* is **NULL**.

5-41.1 Change the "Status Codes Returned" table (page 5-41)

In section 5.3, "Protocol Handler Services – ReinstallProtocolInterface()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The protocol interface was reinstalled.
EFI_NOT_FOUND	The *OldInterface* on the handle was not found.
EFI_ACCESS_DENIED	The protocol interface could not be reinstalled, because *OldInterface* is still being used by a driver that will not release it.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.

TO:

EFI_SUCCESS	The protocol interface was reinstalled.
EFI_NOT_FOUND	The *OldInterface* on the handle was not found.
EFI_ACCESS_DENIED	The protocol interface could not be reinstalled, because *OldInterface* is still being used by a driver that will not release it.
EFI_INVALID_PARAMETER	*Handle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*Protocol* is **NULL**.

5-42.1 Change the "Status Codes Returned" table (page 5-42)

In section 5.3.1, "Protocol Handler Services – RegisterProtocolNotify()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The notification event has been registered.
EFI_OUT_OF_RESOURCES	Space for the notification event could not be allocated.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.

TO:

EFI_SUCCESS	The notification event has been registered.
EFI_OUT_OF_RESOURCES	Space for the notification event could not be allocated.
EFI_INVALID_PARAMETER	*Protocol* is **NULL**.
EFI_INVALID_PARAMETER	*Event* is **NULL**.
EFI_INVALID_PARAMETER	*Registration* is **NULL**.

5-44.1 Change the "Status Codes Returned" table (page 5-44)

In section 5.3.1, "Protocol Handler Services – LocateHandle()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The array of handles was returned.
EFI_NOT_FOUND	No handles match the search.
EFI_BUFFER_TOO_SMALL	The *BufferSize* is too small for the result. *BufferSize* has been updated with the size needed to complete the request.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.

TO:

EFI_SUCCESS	The array of handles was returned.
EFI_NOT_FOUND	No handles match the search.
EFI_BUFFER_TOO_SMALL	The *BufferSize* is too small for the result. *BufferSize* has been updated with the size needed to complete the request.
EFI_INVALID_PARAMETER	*SearchType* is not a member of **EFI_LOCATE_SEARCH_TYPE**.
EFI_INVALID_PARAMETER	*SearchType* is **ByRegisterNotify** and *SearchKey* is **NULL**.
EFI_INVALID_PARAMETER	*SearchType* is **ByProtocol** and *Protocol* is **NULL**.
EFI_INVALID_PARAMETER	One or more matches are found and *BufferSize* is **NULL**.
EFI_INVALID_PARAMETER	*BufferSize* is large enough for the result and *Buffer* is **NULL**.

5-46.1 Change the "Status Codes Returned" table (page 5-46)

In section 5.3.1, "Protocol Handler Services – HandleProtocol()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The interface information for the specified protocol was returned.
EFI_UNSUPPORTED	The device does not support the specified protocol.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.

TO:

EFI_SUCCESS	The interface information for the specified protocol was returned.
EFI_UNSUPPORTED	The device does not support the specified protocol.
EFI_INVALID_PARAMETER	*Handle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*Protocol* is **NULL**.
EFI_INVALID_PARAMETER	*Interface* is **NULL**.

5-48.1 Change the "Status Codes Returned" table (page 5-48)

In section 5.3.1, "Protocol Handler Services – LocateDevicePath()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The resulting handle was returned.
EFI_NOT_FOUND	No handles matched the search.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.

TO:

EFI_SUCCESS	The resulting handle was returned.
EFI_NOT_FOUND	No handles matched the search.
EFI_INVALID_PARAMETER	*Protocol* is **NULL**.
EFI_INVALID_PARAMETER	*DevicePath* is **NULL**.
EFI_INVALID_PARAMETER	A handle matched the search and *Device* is **NULL**.

5-61.1 Change the description of *DriverImageHandle* in the "Parameters" subsection (page 5-61)

In section 5.3.1, "Protocol Handler Services – ConnectController()," change the description of *DriverImageHandle* in the "Parameters" subsection FROM:

DriverImageHandle A pointer to an ordered list of driver image handles. The list is terminated by a **NULL** image handle. These driver image handles are candidates for the driver(s) that will manage the controller specified by *ControllerHandle*. This is an optional parameter that may be **NULL**. This parameter is typically used to debug new drivers.

TO:

DriverImageHandle A pointer to an ordered list of handles that support the **EFI_DRIVER_BINDING_PROTOCOL**. The list is terminated by a **NULL** handle value. These handles are candidates for the Driver Binding Protocol(s) that will manage the controller specified by *ControllerHandle*. This is an optional parameter that may be **NULL**. This parameter is typically used to debug new drivers.

5-62.1 Change the description of Context Override in the "Description" subsection (page 5-62)

In section 5.3.1, "Protocol Handler Services – ConnectController()," change the description of Context Override in the "Description" subsection FROM:

1. ***Context Override*** : *DriverImageHandle* is an ordered list of image handles. The highest priority image handle is the first element of the list, and the lowest priority image handle is the last element of the list. The list is terminated with a **NULL** image handle.

TO:

1. ***Context Override*** : *DriverImageHandle* is an ordered list of handles that support the **EFI_DRIVER_BINDING_PROTOCOL**. The highest priority handle is the first element of the list, and the lowest priority handle is the last element of the list. The list is terminated with a **NULL** handle value.

5-67.1 Add an entry to the "Status Codes Returned" table (page 5-67)

In section 5.3.1, "Protocol Handler Services – DisconnectController()," add the following entry at the end of the "Status Codes Returned" table:

EFI_INVALID_PARAMETER	*DriverImageHandle* does not support the **EFI_DRIVER_BINDING_PROTOCOL**.

5-79.1 Change the "Status Codes Returned" table (page 5-79)

In section 5.4, "Image Services – LoadImage()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	Image was loaded into memory correctly.
EFI_NOT_FOUND	The *FilePath* was not found.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.
EFI_UNSUPPORTED	The image type is not supported, or the device path cannot be parsed to locate the proper protocol for loading the file.
EFI_OUT_OF_RESOURCES	Image was not loaded due to insufficient resources.
EFI_LOAD_ERROR	Image was not loaded because the image format was corrupt or not understood.
EFI_DEVICE_ERROR	Image was not loaded because the device returned a read error.

TO:

EFI_SUCCESS	Image was loaded into memory correctly.
EFI_NOT_FOUND	Both *SourceBuffer* and *FilePath* are **NULL**.
EFI_NOT_FOUND	The *FilePath* was not found.
EFI_INVALID_PARAMETER	*ImageHandle* is **NULL**.
EFI_INVALID_PARAMETER	*ParentImageHandle* is **NULL**.
EFI_INVALID_PARAMETER	*ParentImageHandle* is not a valid **EFI_HANDLE**.
EFI_UNSUPPORTED	The image type is not supported.
EFI_OUT_OF_RESOURCES	Image was not loaded due to insufficient resources.
EFI_LOAD_ERROR	Image was not loaded because the image format was corrupt or not understood.
EFI_DEVICE_ERROR	Image was not loaded because the device returned a read error.

5-83.1 Change the "Summary" paragraph (page 5-83)

In section 5.4, "Image Services – Exit()," change the "Summary" paragraph FROM:

Terminates the currently loaded EFI image and returns control to boot services.

TO:

Terminates a loaded EFI image and returns control to boot services.

5-83.2 Delete the second sentence in the first paragraph of the "Description" subsection (page 5-83)

In section 5.4, "Image Services – Exit()," delete the second sentence in the first paragraph of the "Description" subsection, so it changes FROM (sentence to delete in yellow):

The **`Exit()`** function terminates the image referenced by *`ImageHandle`* and returns control to boot services. This function can only be called by the currently executing image. This function may not be called if the image has already returned from its entry point (**`EFI_IMAGE_ENTRY_POINT`**) or if it has loaded any child images that have not exited (all child images must exit before this image can exit).

TO:

The **`Exit()`** function terminates the image referenced by *`ImageHandle`* and returns control to boot services. This function may not be called if the image has already returned from its entry point (**`EFI_IMAGE_ENTRY_POINT`**) or if it has loaded any child images that have not exited (all child images must exit before this image can exit).

5-84.1 Change the "Status Codes Returned" table (page 5-84)

In section 5.4, "Image Services – Exit()," change the "Status Codes Returned" table FROM:

(Does not return.)	Image exit. Control is returned to the **`StartImage()`** call that invoked the image.
EFI_SUCCESS	The image was unloaded. **`Exit()`** only returns success if the image has not been started; otherwise, the exit returns to the **`StartImage()`** call that invoked the image.
EFI_INVALID_PARAMETER	The specified image is not the current image.

TO:

(Does not return.)	Image exit. Control is returned to the **`StartImage()`** call that invoked the image specified by *`ImageHandle`*.
EFI_SUCCESS	The image specified by *`ImageHandle`* was unloaded. This condition only occurs for images that have been loaded with **`LoadImage()`** but have not been started with **`StartImage()`**.
EFI_INVALID_PARAMETER	The image specified by *`ImageHandle`* has been loaded and started with **`LoadImage()`** and **`StartImage()`**, but the image is not the currently executing image.

5-85.1 Change the first paragraph of the "Description" subsection (page 5-85)

In section 5.4, "Image Services – ExitBootServices()," change the first paragraph of the "Description" subsection FROM:

> The **ExitBootServices()** function is called by the currently executing EFI OS loader image to terminate all boot services. On success, the loader becomes responsible for the continued operation of the system.

TO:

> The **ExitBootServices()** function is called by the currently executing EFI OS loader image to terminate all boot services. On success, the loader becomes responsible for the continued operation of the system. All events of type **EVT_SIGNAL_EXIT_BOOT_SERVICES** must be signaled before **ExitBootServices()** returns.

5-91.1 Change the "Status Codes Returned" table (page 5-91)

In section 5.5, "Miscellaneous Boot Services – GetNextMonotonicCount()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The next monotonic count was returned.
EFI_DEVICE_ERROR	The device is not functioning properly.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.

TO:

EFI_SUCCESS	The next monotonic count was returned.
EFI_DEVICE_ERROR	The device is not functioning properly.
EFI_INVALID_PARAMETER	*Count* is **NULL**.

Chapter 6: Runtime Services

6-5.1 Change the third paragraph in the "Description" subsection (page 6-5)

In section 6.1, "Variable Services – GetNextVariableName()," change the third paragraph in the "Description" subsection FROM:

> To start the search, a Null-terminated string is passed in *VariableName*; that is, *VariableName* is a pointer to a Null Unicode character. This is always done on the initial call to **GetNextVariableName()**. When *VariableName* is a pointer to a Null Unicode character, *VendorGuid* is ignored. **GetNextVariableName()** cannot be used as a filter to return variable names with a specific GUID. Instead, the entire list of variables must be retrieved, and the caller may act as a filter if it chooses. Calls to **SetVariable()** between calls to **GetNextVariableName()** may produce unpredictable results.

TO:

To start the search, a Null-terminated string is passed in *VariableName*; that is, *VariableName* is a pointer to a Null Unicode character. This is always done on the initial call to **GetNextVariableName()**. When *VariableName* is a pointer to a Null Unicode character, *VendorGuid* is ignored. **GetNextVariableName()** cannot be used as a filter to return variable names with a specific GUID. Instead, the entire list of variables must be retrieved, and the caller may act as a filter if it chooses. Calls to **SetVariable()** between calls to **GetNextVariableName()** may produce unpredictable results. Passing in a *VariableName* parameter that is neither a Null-terminated string nor a value that was returned on the previous call to **GetNextVariableName()** may also produce unpredictable results.

6-17.1 Change the first paragraph in the "Description" subsection (page 6-17)

In section 6.3, "Virtual Memory Services – SetVirtualAddressMap()," change the first paragraph in the "Description" subsection FROM:

The **SetVirtualAddressMap()** function is used by the OS loader. The function can only be called at runtime, and is called by the owner of the system's memory map. I.e., the component which called **ExitBootServices()**.

TO:

The **SetVirtualAddressMap()** function is used by the OS loader. The function can only be called at runtime, and is called by the owner of the system's memory map. I.e., the component which called **ExitBootServices()**. All events of type **EVT_SIGNAL_VIRTUAL_ADDRESS_CHANGE** must be signaled before **SetVirtualAddressMap()** returns.

6-19.1 Change the "Status Codes Returned" table (page 6-19)

In section 6.3, "Virtual Memory Services – ConvertPointer()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The pointer pointed to by *Address* was modified.
EFI_NOT_FOUND	The pointer pointed to by *Address* was not found to be part of the current memory map. This is normally fatal.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.

TO:

EFI_SUCCESS	The pointer pointed to by *Address* was modified.
EFI_NOT_FOUND	The pointer pointed to by *Address* was not found to be part of the current memory map. This is normally fatal.
EFI_INVALID_PARAMETER	*Address* is **NULL**.
EFI_INVALID_PARAMETER	**Address* is **NULL** and *DebugDisposition* does not have the **EFI_OPTIONAL_PTR** bit set.

Chapter 8: Protocols – Device Path Protocol

8-11.1 Change the description of the Length field in Table 8-14 (page 8-11)

In section 8.3.4.5, "USB Device Path," change the description of the Length field in Table 8-14 FROM:

Length of this structure in bytes. Length is 16 bytes.

TO:

Length of this structure in bytes. Length is 6 bytes.

Chapter 9: Protocols – EFI Driver Model

9-19.1 Delete the last sentence from the first paragraph in the "Description" subsection (page 9-19)

In section 9.1, "EFI Driver Binding Protocol – Stop()" delete the last sentence from the first paragraph in the "Description" subsection on page 9-19, so it changes FROM (sentence to delete in yellow):

If *ControllerHandle* cannot be stopped, then **EFI_DEVICE_ERROR** is returned. If, for some reason, there are not enough resources to stop *ControllerHandle*, then **EFI_OUT_OF_RESOURCES** is returned. If *ControllerHandle* was not started by the driver specified by *This*, then **EFI_UNSUPPORTED** is returned.

TO:

If *ControllerHandle* cannot be stopped, then **EFI_DEVICE_ERROR** is returned. If, for some reason, there are not enough resources to stop *ControllerHandle*, then **EFI_OUT_OF_RESOURCES** is returned.

9-37.1 Change the "Status Codes Returned" table (page 9-37)

In section 9.4, "EFI Driver Configuration Protocol – SetOptions()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The driver specified by *This* successfully set the configuration options for the controller specified by *ControllerHandle*.
EFI_INVALID_PARAMETER	*ControllerHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*ChildHandle* is not **NULL** and it is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*ActionRequired* is **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* does not support setting configuration options for the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_UNSUPPORTED	The driver specified by *This* does not support the language specified by *Language*.

EFI_DEVICE_ERROR	A device error occurred while attempt to set the configuration options for the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_OUT_RESOURCES	There are not enough resources available to set the configuration options for the controller specified by *ControllerHandle* and *ChildHandle*.

TO:

EFI_SUCCESS	The driver specified by *This* successfully set the configuration options for the controller specified by *ControllerHandle*.
EFI_INVALID_PARAMETER	*ControllerHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	The driver specified by *This* is not a device driver, and *ChildHandle* is not **NULL**, and *ChildHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*ActionRequired* is **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* is a device driver and *ChildHandle* is not **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* does not support setting configuration options for the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_UNSUPPORTED	The driver specified by *This* does not support the language specified by *Language*.
EFI_DEVICE_ERROR	A device error occurred while attempt to set the configuration options for the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_OUT_RESOURCES	There are not enough resources available to set the configuration options for the controller specified by *ControllerHandle* and *ChildHandle*.

9-39.1 Change the “Status Codes Returned” table (page 9-39)

In section 9.4, “EFI Driver Configuration Protocol – OptionsValid(),” change the “Status Codes Returned” table FROM:

EFI_SUCCESS	The controller specified by *ControllerHandle* and *ChildHandle* that is being managed by the driver specified by *This* has a valid set of configuration options.
EFI_INVALID_PARAMETER	*ControllerHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*ChildHandle* is not **NULL** and it is not a valid **EFI_HANDLE**.
EFI_UNSUPPORTED	The driver specified by *This* is not currently managing the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_DEVICE_ERROR	The controller specified by *ControllerHandle* and *ChildHandle* that is being managed by the driver specified by *This* has an invalid set of configuration options.

TO:

EFI_SUCCESS	The controller specified by *ControllerHandle* and *ChildHandle* that is being managed by the driver specified by *This* has a valid set of configuration options.
EFI_INVALID_PARAMETER	*ControllerHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	The driver specified by *This* is not a device driver, and *ChildHandle* is not **NULL**, and *ChildHandle* is not a valid **EFI_HANDLE**.
EFI_UNSUPPORTED	The driver specified by *This* is a device driver and *ChildHandle* is not **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* is not currently managing the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_DEVICE_ERROR	The controller specified by *ControllerHandle* and *ChildHandle* that is being managed by the driver specified by *This* has an invalid set of configuration options.

9-42.1 Change the "Status Codes Returned" table (page 9-42)

In section 9.4, "EFI Driver Configuration Protocol – ForceDefaults()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The driver specified by *This* successfully forced the default configuration options on the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_INVALID_PARAMETER	*ControllerHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*ChildHandle* is not **NULL** and it is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*ActionRequired* is **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* does not support forcing the default configuration options on the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_UNSUPPORTED	The driver specified by This does not support the configuration type specified by *DefaultType*.
EFI_DEVICE_ERROR	A device error occurred while attempt to force the default configuration options on the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_OUT_RESOURCES	There are not enough resources available to force the default configuration options on the controller specified by *ControllerHandle* and *ChildHandle*.

TO:

EFI_SUCCESS	The driver specified by *This* successfully forced the default configuration options on the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_INVALID_PARAMETER	*ControllerHandle* is not a valid **EFI_HANDLE**.

EFI_INVALID_PARAMETER	The driver specified by *This* is not a device driver, and *ChildHandle* is not **NULL**, and *ChildHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*ActionRequired* is **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* is a device driver and *ChildHandle* is not **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* does not support forcing the default configuration options on the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_UNSUPPORTED	The driver specified by *This* does not support the configuration type specified by *DefaultType*.
EFI_DEVICE_ERROR	A device error occurred while attempt to force the default configuration options on the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_OUT_RESOURCES	There are not enough resources available to force the default configuration options on the controller specified by *ControllerHandle* and *ChildHandle*.

9-46.1 Change the "Status Codes Returned" table (page 9-46)

In section 9.5, "EFI Driver Diagnostics Protocol – RunDiagnostics()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The controller specified by *ControllerHandle* and *ChildHandle* passed the diagnostic.
EFI_INVALID_PARAMETER	*ControllerHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*ChildHandle* is not **NULL** and it is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*Language* is **NULL**.
EFI_INVALID_PARAMETER	*ErrorType* is **NULL**.
EFI_INVALID_PARAMETER	*BufferType* is **NULL**.
EFI_INVALID_PARAMETER	*Buffer* is **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* does not support running diagnostics for the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_UNSUPPORTED	The driver specified by *This* does not support the type of diagnostic specified by *DiagnosticType*.
EFI_UNSUPPORTED	The driver specified by *This* does not support the language specified by *Language*.
EFI_OUT_OF_RESOURCES	There are not enough resources available to complete the diagnostics.
EFI_OUT_OF_RESOURCES	There are not enough resources available to return the status information in *ErrorType*, *BufferSize*, and *Buffer*.
EFI_DEVICE_ERROR	The controller specified by *ControllerHandle* and *ChildHandle* did not pass the diagnostic.

TO:

EFI_SUCCESS	The controller specified by *ControllerHandle* and *ChildHandle* passed the diagnostic.
EFI_INVALID_PARAMETER	*ControllerHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	The driver specified by *This* is not a device driver, and *ChildHandle* is not **NULL**, and *ChildHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*Language* is **NULL**.
EFI_INVALID_PARAMETER	*ErrorType* is **NULL**.
EFI_INVALID_PARAMETER	*BufferSize* is **NULL**.
EFI_INVALID_PARAMETER	*Buffer* is **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* is a device driver and *ChildHandle* is not **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* does not support running diagnostics for the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_UNSUPPORTED	The driver specified by *This* does not support the type of diagnostic specified by *DiagnosticType*.
EFI_UNSUPPORTED	The driver specified by *This* does not support the language specified by *Language*.
EFI_OUT_OF_RESOURCES	There are not enough resources available to complete the diagnostics.
EFI_OUT_OF_RESOURCES	There are not enough resources available to return the status information in *ErrorType*, *BufferSize*, and *Buffer*.
EFI_DEVICE_ERROR	The controller specified by *ControllerHandle* and *ChildHandle* did not pass the diagnostic.

9-51.1 Change the “Status Codes Returned” table (page 9-51)

In section 9.6, “EFI Component Name Protocol – GetControllerName(),” change the “Status Codes Returned” table FROM:

EFI_SUCCESS	The Unicode string for the user readable name specified by *This*, *ControllerHandle*, *ChildHandle*, and *Language* was returned in *ControllerName*.
EFI_INVALID_PARAMETER	*ControllerHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*ChildHandle* is not **NULL** and it is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*Language* is **NULL**.
EFI_INVALID_PARAMETER	*ControllerName* is **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* is not currently managing the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_UNSUPPORTED	The driver specified by *This* does not support the language specified by *Language*.

TO:

EFI_SUCCESS	The Unicode string for the user readable name specified by *This*, *ControllerHandle*, *ChildHandle*, and *Language* was returned in *ControllerName*.
EFI_INVALID_PARAMETER	*ControllerHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	The driver specified by *This* is not a device driver, and *ChildHandle* is not **NULL**, and *ChildHandle* is not a valid **EFI_HANDLE**.
EFI_INVALID_PARAMETER	*Language* is **NULL**.
EFI_INVALID_PARAMETER	*ControllerName* is **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* is a device driver and *ChildHandle* is not **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* is not currently managing the controller specified by *ControllerHandle* and *ChildHandle*.
EFI_UNSUPPORTED	The driver specified by *This* does not support the language specified by *Language*.

Chapter 10: Protocols – Console Support

10-17.1 Change the second paragraph of the "Description" subsection (page 10-17)

In section 10.3, "SIMPLE_TEXT_OUTPUT_PROTOCOL – QueryMode()," change the second paragraph of the "Description" subsection FROM:

> It is required that all output devices support at least 80x25 text mode. This mode is defined to be mode 0. If the output devices support 80x50, that is defined to be mode 1. Any other text dimensions supported by the device may then follow as mode 2 and above. (For example, it is a prerequisite that 80x25 and 80x50 text modes be supported before any other modes are.)

TO:

> It is required that all output devices support at least 80x25 text mode. This mode is defined to be mode 0. If the output devices support 80x50, that is defined to be mode 1. All other text dimensions supported by the device will follow as modes 2 and above. If an output device supports modes 2 and above, but does not support 80x50, then querying for mode 1 will return **EFI_UNSUPPORTED**.

10.19-1 Change the description of *Attribute* in the "Parameters" subsection (page 10-19)

In section 10.3, "SIMPLE_TEXT_OUTPUT_PROTOCOL – SetAttribute()," change the description of *Attribute* in the "Parameters" subsection FROM:

Attribute	The attribute to set. Bits 0..3 are the foreground color, and bits 4..6 are the background color. All other bits are undefined and must be zero. See "Related Definitions" below.

TO:

Attribute	The attribute to set. Bits 0..3 are the foreground color, and bits 4..6 are the background color. All other bits are reserved. See "Related Definitions" below.

10-20.1 Delete EFI_UNSUPPORTED from the "Status Codes Returned" table (page 10-20)

In section 10.3, "SIMPLE_TEXT_OUTPUT_PROTOCOL – SetAttribute()," delete **EFI_UNSUPPORTED** from the "Status Codes Returned" table, so it changes FROM (row to delete in yellow):

EFI_SUCCESS	The requested attributes were set.
EFI_DEVICE_ERROR	The device had an error and could not complete the request.
EFI_UNSUPPORTED	The attribute requested is not defined by this specification.

TO:

EFI_SUCCESS	The requested attributes were set.
EFI_DEVICE_ERROR	The device had an error and could not complete the request.

10-28.1 Change the "Status Codes Returned" table (page 10-28)

In section 10.5, "EFI_UGA_DRAW_PROTOCOL – GetMode()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	Valid mode information was returned.
EFI_DEVICE_ERROR	A hardware error occurred trying to retrieve the video mode.
EFI_INVALID_PARAMETER	*HorizontalResolution*, or *VerticalResolution*, or *RefreshRate*, is **NULL**.

TO:

EFI_SUCCESS	Valid mode information was returned.
EFI_DEVICE_ERROR	A hardware error occurred trying to retrieve the video mode.
EFI_INVALID_PARAMETER	*HorizontalResolution* is **NULL**.
EFI_INVALID_PARAMETER	*VerticalResolution* is **NULL**.
EFI_INVALID_PARAMETER	*ColorDepth* is **NULL**.
EFI_INVALID_PARAMETER	*RefreshRate* is **NULL**.

10-33.1 Change the `EfiUgaVideoFill` entry in Table 10-4 (page 10-33)

In section 10.5, "EFI_UGA_DRAW_PROTOCOL – Blt()," change the **`EfiUgaVideoFill`** entry in Table 10-4 in the "Description" subsection FROM:

`EfiUgaVideoFill`	Write data from the *`BltBuffer`* pixel (*`SourceX`*, *`SourceY`*) directly to every pixel of the video display rectangle (*`DestinationX`*, *`DestinationY`*) (*`DestinationX`* + *`Width`*, *`DestinationY`* + *`Height`*). Only one pixel will be used from the *`BltBuffer`*. *`Delta`* is NOT used.

TO:

`EfiUgaVideoFill`	Write data from the *`BltBuffer`* pixel (0,0) directly to every pixel of the video display rectangle (*`DestinationX`*, *`DestinationY`*) (*`DestinationX`* + *`Width`*, *`DestinationY`* + *`Height`*). Only one pixel will be used from the *`BltBuffer`*. *`Delta`* is NOT used.

Chapter 11: Protocols – Bootable Image Support

11-3.1 Change the `EFI_NO_SUCH_MEDIA` return code (page 11-3)

In section 11.1, "LOAD_FILE Protocol – LoadFile()," change the **`EFI_NO_SUCH_MEDIA`** return code in the "Status Codes Returned" table FROM:

EFI_NO_SUCH_MEDIA	No medium was present to load the file.

TO:

EFI_NO_MEDIA	No medium was present to load the file.

11-9.1 Change the description of the Revision field in Table 11-1 (page 11-9)

In section 11.2.2.1, "EFI Partition Header," change the description of the Revision field in Table 11-1 FROM:

Revision	8	4	The specification revision number that this header complies to. For version 1.0 of the specification the correct value is 0x00010000.

TO:

Revision	8	4	The revision number for this header. This revision value is not related to the EFI Specification version. This header is version 1.0, so the correct value is 0x00010000.

11-26.1 Change the "Status Codes Returned" table (page 11-26)

In section 11.4, "EFI_FILE Protocol – Read()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The data was read.
EFI_NO_MEDIA	The device has no medium.
EFI_DEVICE_ERROR	The device reported an error.
EFI_VOLUME_CORRUPTED	The file system structures are corrupted.
EFI_BUFFER_TOO_SMALL	The *BufferSize* is too small to read the current directory entry. *BufferSize* has been updated with the size needed to complete the request.

TO:

EFI_SUCCESS	The data was read.
EFI_NO_MEDIA	The device has no medium.
EFI_DEVICE_ERROR	The device reported an error.
EFI_DEVICE_ERROR	An attempt was made to read from a deleted file.
EFI_DEVICE_ERROR	On entry, the current file position is beyond the end of the file.
EFI_VOLUME_CORRUPTED	The file system structures are corrupted.
EFI_BUFFER_TOO_SMALL	The *BufferSize* is too small to read the current directory entry. *BufferSize* has been updated with the size needed to complete the request.

11-27.1 Change the "Status Codes Returned" table (page 11-27)

In section 11.4, "EFI_FILE Protocol – Write()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The data was written.
EFI_UNSUPPORT	Writes to open directory files are not supported.
EFI_NO_MEDIA	The device has no medium.
EFI_DEVICE_ERROR	The device reported an error.
EFI_VOLUME_CORRUPTED	The file system structures are corrupted.
EFI_WRITE_PROTECTED	The file or medium is write protected.
EFI_ACCESS_DENIED	The file was opened read only.
EFI_VOLUME_FULL	The volume is full.

TO:

EFI_SUCCESS	The data was written.
EFI_UNSUPPORT	Writes to open directory files are not supported.
EFI_NO_MEDIA	The device has no medium.
EFI_DEVICE_ERROR	The device reported an error.
EFI_DEVICE_ERROR	An attempt was made to write to a deleted file.
EFI_VOLUME_CORRUPTED	The file system structures are corrupted.
EFI_WRITE_PROTECTED	The file or medium is write protected.
EFI_ACCESS_DENIED	The file was opened read only.
EFI_VOLUME_FULL	The volume is full.

11-28.1 Change the "Status Codes Returned" table (page 11-28)

In section 11.4, "EFI_FILE Protocol – SetPosition()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The position was set.
EFI_UNSUPPORTED	The seek request for nonzero is not valid on open directories.

TO:

EFI_SUCCESS	The position was set.
EFI_UNSUPPORTED	The seek request for nonzero is not valid on open directories.
EFI_DEVICE_ERROR	An attempt was made to set the position of a deleted file.

11-29.1 Change the "Status Codes Returned" table (page 11-29)

In section 11.4, "EFI_FILE Protocol – GetPosition()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The position was returned.
EFI_UNSUPPORTED	The request is not valid on open directories.

TO:

EFI_SUCCESS	The position was returned.
EFI_UNSUPPORTED	The request is not valid on open directories.
EFI_DEVICE_ERROR	An attempt was made to get the position from a deleted file.

11-31.1 Change the "Description" subsection in its entirety (page 11-31)

In section 11.4, "EFI_FILE Protocol – SetInfo()," change the "Description" subsection in its entirety FROM:

> The **`SetInfo()`** function sets information of type *`InformationType`* on the requested file.

TO:

The **SetInfo()** function sets information of type *InformationType* on the requested file.
Because a read-only file can be opened only in read-only mode, an *InformationType* of **EFI_FILE_INFO_ID** can be used with a read-only file because this method is the only one that can be used to convert a read-only file to a read-write file. In this circumstance, only the *Attribute* field of the **EFI_FILE_INFO** structure may be modified. One or more calls to **SetInfo()** to change the *Attribute* field are permitted before it is closed. The file attributes will be valid the next time the file is opened with **Open()**.
An *InformationType* of **EFI_FILE_SYSTEM_INFO_ID** or **EFI_FILE_SYSTEM_VOLUME_LABEL_ID** may not be used on read-only media.

11-31.2 Change the "Status Codes Returned" table (page 11-31)

In section 11.4, "EFI_FILE Protocol – SetInfo()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The information was set.
EFI_UNSUPPORTED	The *InformationType* is not known.
EFI_NO_MEDIA	The device has no medium.
EFI_DEVICE_ERROR	The device reported an error.
EFI_VOLUME_CORRUPTED	The file system structures are corrupted.
EFI_WRITE_PROTECTED	The file or medium is write protected.
EFI_ACCESS_DENIED	The file was opened read-only.
EFI_VOLUME_FULL	The volume is full.
EFI_BAD_BUFFER_SIZE	*BufferSize* is smaller than the size of the type indicated by *InformationType.*

TO:

EFI_SUCCESS	The information was set.
EFI_UNSUPPORTED	The *InformationType* is not known.
EFI_NO_MEDIA	The device has no medium.
EFI_DEVICE_ERROR	The device reported an error.
EFI_VOLUME_CORRUPTED	The file system structures are corrupted.
EFI_WRITE_PROTECTED	*InformationType* is **EFI_FILE_INFO_ID** and the media is read-only.
EFI_WRITE_PROTECTED	*InformationType* is **EFI_FILE_SYSTEM_INFO_ID** and the media is read only.
EFI_WRITE_PROTECTED	*InformationType* is **EFI_FILE_SYSTEM_VOLUME_LABEL_ID** and the media is read-only.
EFI_ACCESS_DENIED	An attempt is made to change the name of a file to a file that is already present.

EFI_ACCESS_DENIED	An attempt is being made to change the **EFI_FILE_DIRECTORY** *Attribute*.
EFI_ACCESS_DENIED	An attempt is being made to change the size of a directory.
EFI_ACCESS_DENIED	*InformationType* is **EFI_FILE_INFO_ID** and the file was opened read-only and an attempt is being made to modify a field other than *Attribute*.
EFI_VOLUME_FULL	The volume is full.
EFI_BAD_BUFFER_SIZE	*BufferSize* is smaller than the size of the type indicated by *InformationType.*

11-58.1 Change the description of the `StrToFat()` API (page 11-58)

In section 11.7, "UNICODE_COLLATION Protocol – StrToFat()," change the description of this API in its entirety to the following (specific changes in yellow):

UNICODE_COLLATION.StrToFat()

Summary

Converts a Null-terminated Unicode string to legal characters in a FAT filename using an OEM character set.

Prototype

```
BOOLEAN
(EFIAPI *EFI_UNICODE_COLLATION_STRTOFAT) (
  IN UNICODE_COLLATION_INTERFACE   *This,
  IN CHAR16                        *String,
  IN UINTN                         FatSize,
  OUT CHAR8                        *Fat
  );
```

Parameters

This	A pointer to the **UNICODE_COLLATION_INTERFACE** instance. Type **UNICODE_COLLATION_INTERFACE** is defined in Section 11.7.
String	A pointer to a Null-terminated Unicode string.
FatSize	The size of the string *Fat* in bytes.
Fat	A pointer to a string that contains the converted version of *String* using legal FAT characters from an OEM character set.

Description

This function converts the Unicode characters from *String* into legal FAT characters in an OEM character set and stores then in the string *Fat*. This conversion continues until either *FatSize*

bytes are stored in *Fat*, or the end of *String* is reached. The Unicode characters '.' (period) and ' ' (space) are ignored for this conversion. Unicode characters that map to an illegal FAT character are substituted with an '_'. If no valid mapping from a Unicode character to an OEM character is available, then it is also substituted with an '_'. If any of the Unicode characters conversions are substituted with a '_', then **TRUE** is returned. Otherwise **FALSE** is returned.

Status Codes Returned

TRUE	One or more conversions failed and were substituted with '_'.
FALSE	None of the conversions failed.

Chapter 12: Protocols – PCI Bus Support

12-11.1 Add three attributes and descriptions to the list of PCI Root Bridge I/O Protocol Attribute bits in the "Related Definitions" subsection (page 12-11)

In section 12.2, "EFI PCI Root Bridge I/O Protocol," add the following three attributes and descriptions to the end of the list of PCI Root Bridge I/O Protocol Attribute bits in the "Related Definitions" subsection:

```
#define EFI_PCI_ATTRIBUTE_ISA_IO_16           0x10000
#define EFI_PCI_ATTRIBUTE_VGA_IO_16           0x20000
#define EFI_PCI_ATTRIBUTE_VGA_PALETTE_IO_16   0x40000
```

EFI_PCI_ATTRIBUTE_ISA_IO_16

If this bit is set, then the PCI I/O cycles between 0x100 and 0x3FF are forwarded onto a PCI root bridge using a 16-bit address decoder on address bits 0..15. Address bits 16..31 must be zero. This bit is used to forward I/O cycles for legacy ISA devices onto a PCI root bridge. This bit may not be combined with **EFI_PCI_ATTRIBUTE_ISA_IO**.

EFI_PCI_ATTRIBUTE_VGA_PALETTE_IO_16

If this bit is set, then the PCI I/O write cycles for 0x3C6, 0x3C8, and 0x3C9 are forwarded onto a PCI root bridge using a 16-bit address decoder on address bits 0..15. Address bits 16..31 must be zero. This bit is used to forward I/O write cycles to the VGA palette registers onto a PCI root bridge. This bit may not be combined with **EFI_PCI_ATTRIBUTE_VGA_IO** or **EFI_PCI_ATTRIBUTE_VGA_PALETTE_IO**.

EFI_PCI_ATTRIBUTE_VGA_IO_16

If this bit is set, then the PCI I/O cycles in the ranges 0x3B0–0x3BB and 0x3C0–0x3DF are forwarded onto a PCI root bridge using a 16-bit address decoder on address bits 0..15. Address bits 16..31 must be zero. This bit is used to forward I/O cycles for a VGA controller onto a PCI root bridge. This bit may not be combined with **EFI_PCI_ATTRIBUTE_VGA_IO** or **EFI_PCI_ATTRIBUTE_VGA_PALETTE_IO**. Because **EFI_PCI_ATTRIBUTE_VGA_IO_16** also includes the I/O range described by **EFI_PCI_ATTRIBUTE_VGA_PALETTE_IO_16**, the

EFI_PCI_ATTRIBUTE_VGA_PALETTE_IO_16 bit is ignored if **EFI_PCI_ATTRIBUTE_VGA_IO_16** is set.

12-12.1 Change the descriptions of the IDE attributes in the "Related Definitions" subsection (page 12-12)

In section 12.1.1, "EFI PCI Root Bridge I/O Protocol," change the descriptions of the IDE attributes in the "Related Definitions" subsection FROM:

EFI_PCI_ATTRIBUTE_IDE_PRIMARY_IO

If this bit is set, then the PCI I/O cycles in the ranges 0x1F0-0x1F7 and 0x3F6-0x3F7 are forwarded onto a PCI root bridge using a 10-bit address decoder on address bits 0..9. Address bits 10..15 are not decoded, and address bits 16..31 must be zero. This bit is used to forward I/O cycles for a Primary IDE controller onto a PCI root bridge.

EFI_PCI_ATTRIBUTE_IDE_SECONDARY_IO

If this bit is set, then the PCI I/O cycles in the ranges 0x170-0x177 and 0x376-0x377 are forwarded onto a PCI root bridge using a 10-bit address decoder on address bits 0..9. Address bits 10..15 are not decoded, and address bits 16..31 must be zero. This bit is used to forward I/O cycles for a Secondary IDE controller onto a PCI root bridge.

TO:

EFI_PCI_ATTRIBUTE_IDE_PRIMARY_IO

If this bit is set, then the PCI I/O cycles in the ranges 0x1F0–0x1F7 and 0x3F6–0x3F7 are forwarded onto a PCI root bridge using a 16-bit address decoder on address bits 0..15. Address bits 16..31 must be zero. This bit is used to forward I/O cycles for a Primary IDE controller onto a PCI root bridge.

EFI_PCI_ATTRIBUTE_IDE_SECONDARY_IO

If this bit is set, then the PCI I/O cycles in the ranges 0x170–0x177 and 0x376–0x377 are forwarded onto a PCI root bridge using a 16-bit address decoder on address bits 0..15. Address bits 16..31 must be zero. This bit is used to forward I/O cycles for a Secondary IDE controller onto a PCI root bridge.

12-60.1 Add three attributes and descriptions to the list of PCI I/O Protocol Attribute bits in the "Related Definitions" subsection (page 12-60)

In section 12.4, "EFI PCI I/O Protocol," add the following three attributes and descriptions to the end of the list of PCI I/O Protocol Attribute bits:

```
#define EFI_PCI_IO_ATTRIBUTE_ISA_IO_16          0x10000
#define EFI_PCI_IO_ATTRIBUTE_VGA_PALETTE_IO_16  0x20000
#define EFI_PCI_IO_ATTRIBUTE_VGA_IO_16          0x40000
```

EFI_PCI_IO_ATTRIBUTE_ISA_IO_16

If this bit is set, then the PCI I/O cycles between 0x100 and 0x3FF are forwarded to the PCI controller using a 16-bit address decoder on address bits 0..15. Address bits 16..31 must be zero. This bit is used to forward I/O cycles for legacy ISA devices. If this bit is set, then the PCI Host Bus Controller and all the PCI to PCI bridges between the PCI Host Bus Controller and the PCI Controller are configured to forward these PCI I/O cycles. This bit may not be combined with **EFI_PCI_IO_ATTRIBUTE_ISA_IO**.

EFI_PCI_IO_ATTRIBUTE_VGA_PALETTE_IO_16

If this bit is set, then the PCI I/O write cycles for 0x3C6, 0x3C8, and 0x3C9 are forwarded to the PCI controller using a 16-bit address decoder on address bits 0..15. Address bits 16..31 must be zero. This bit is used to forward I/O write cycles to the VGA palette registers on a PCI controller. If this bit is set, then the PCI Host Bus Controller and all the PCI to PCI bridges between the PCI Host Bus Controller and the PCI Controller are configured to forward these PCI I/O cycles. This bit may not be combined with **EFI_PCI_IO_ATTRIBUTE_VGA_IO** or **EFI_PCI_IO_ATTRIBUTE_VGA_PALETTE_IO**.

EFI_PCI_IO_ATTRIBUTE_VGA_IO_16

If this bit is set, then the PCI I/O cycles in the ranges 0x3B0–0x3BB and 0x3C0–0x3DF are forwarded to the PCI controller using a 16-bit address decoder on address bits 0..15. Address bits 16..31 must be zero. This bit is used to forward I/O cycles for a VGA controller to a PCI controller. If this bit is set, then the PCI Host Bus Controller and all the PCI to PCI bridges between the PCI Host Bus Controller and the PCI Controller are configured to forward these PCI I/O cycles. This bit may not be combined with **EFI_PCI_IO_ATTRIBUTE_VGA_IO** or **EFI_PCI_IO_ATTRIBUTE_VGA_PALETTE_IO**. Because **EFI_PCI_IO_ATTRIBUTE_VGA_IO_16** also includes the I/O range described by **EFI_PCI_IO_ATTRIBUTE_VGA_PALETTE_IO_16**, the **EFI_PCI_IO_ATTRIBUTE_VGA_PALETTE_IO_16** bit is ignored if **EFI_PCI_IO_ATTRIBUTE_VGA_IO_16** is set.

12-61.1 Change the descriptions of EFI_PCI_IO_ATTRIBUTE_IDE_PRIMARY_IO and EFI_PCI_IO_ATTRIBUTE_IDE_SECONDARY_IO in "Related Definitions" (page 12-61)

In section 12.4, "EFI PCI I/O Protocol," change the descriptions of the IDE attributes **EFI_PCI_IO_ATTRIBUTE_IDE_PRIMARY_IO** and **EFI_PCI_IO_ATTRIBUTE_IDE_SECONDARY_IO** in the "Related Definitions" subsection FROM:

EFI_PCI_IO_ATTRIBUTE_IDE_PRIMARY_IO

If this bit is set, then the PCI I/O cycles in the ranges 0x1F0-0x1F7 and 0x3F6-0x3F7 are forwarded to a PCI controller using a 10-bit address decoder on address bits 0..9. Address bits 10..15 are not decoded, and address bits 16..31 must be zero. This bit is used to forward I/O cycles for a Primary IDE controller to a PCI controller. If this bit is set, then the PCI Host Bus Controller and all the PCI to PCI bridges between the PCI

Host Bus Controller and the PCI Controller are configured to forward these PCI I/O cycles.

EFI_PCI_IO_ATTRIBUTE_IDE_SECONDARY_IO

If this bit is set, then the PCI I/O cycles in the ranges 0x170-0x177 and 0x376-0x377 are forwarded to a PCI controller using a 10-bit address decoder on address bits 0..9. Address bits 10..15 are not decoded, and address bits 16..31 must be zero. This bit is used to forward I/O cycles for a Secondary IDE controller to a PCI controller. If this bit is set, then the PCI Host Bus Controller and all the PCI to PCI bridges between the PCI Host Bus Controller and the PCI Controller are configured to forward these PCI I/O cycles.

TO:

EFI_PCI_IO_ATTRIBUTE_IDE_PRIMARY_IO

If this bit is set, then the PCI I/O cycles in the ranges 0x1F0-0x1F7 and 0x3F6-0x3F7 are forwarded to a PCI controller using a 16-bit address decoder on address bits 0..15. Address bits 16..31 must be zero. This bit is used to forward I/O cycles for a Primary IDE controller to a PCI controller. If this bit is set, then the PCI Host Bus Controller and all the PCI to PCI bridges between the PCI Host Bus Controller and the PCI Controller are configured to forward these PCI I/O cycles.

EFI_PCI_IO_ATTRIBUTE_IDE_SECONDARY_IO

If this bit is set, then the PCI I/O cycles in the ranges 0x170-0x177 and 0x376-0x377 are forwarded to a PCI controller using a 16-bit address decoder on address bits 0..15. Address bits 16..31 must be zero. This bit is used to forward I/O cycles for a Secondary IDE controller to a PCI controller. If this bit is set, then the PCI Host Bus Controller and all the PCI to PCI bridges between the PCI Host Bus Controller and the PCI Controller are configured to forward these PCI I/O cycles.

12-70.1 Change the first sentence in the last paragraph of the "Description" subsection (page 12-70)

In section 12.4, "EFI_PCI_IO_PROTOCOL – Mem.Read() and Mem.Write()," change the start of the first sentence in the last paragraph of the "Description" subsection FROM:

All the PCI transactions generated by this function are guaranteed to be completed before this function returns.

TO:

All the PCI read transactions generated by this function are guaranteed to be completed before this function returns.

12-80.1 Delete the status code EFI_INVALID_PARAMETER from the "Status Codes Returned" table (page 12-80)

In section 12.4, "EFI PCI I/O Protocol – Unmap()," delete the status code **EFI_INVALID_PARAMETER** from the "Status Codes Returned" table.

Chapter 14: Protocols – USB Support

14-20.1 Change the fourth item in the numbered list in the "Description" subsection (page 14-20)

In section 14.1.1, "USB Host Controller Protocol – SyncInterruptTransfer()," change the fourth item in the numbered list in the "Description" subsection FROM:

4. *MaximumPacketLength* is not valid. The legal value of this parameter is for the full-speed device, it should be 8, 16, 32, or 64; for the slow device, it is limited to 8.

TO:

4. *MaximumPacketLength* is not valid. The legal value of this parameter should be 64 or less for a full-speed device; for a slow device, it is limited to 8 or less.

Chapter 15: Protocols – Network Support

15-13.1 Change the "Status Codes Returned" table (page 15-13)

In section 15.1, "EFI_SIMPLE_NETWORK Protocol – StationAddress()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The network interface's station address was updated.
EFI_NOT_STARTED	The network interface has not been started.
EFI_INVALID_PARAMETER	One or more of the parameters has an unsupported value.
EFI_DEVICE_ERROR	The command could not be sent to the network interface.
EFI_UNSUPPORTED	This function is not supported by the network interface.

TO:

EFI_SUCCESS	The network interface's station address was updated.
EFI_NOT_STARTED	The Simple Network Protocol interface has not been started by calling **Start()**.
EFI_INVALID_PARAMETER	The *New* station address was not accepted by the NIC.
EFI_INVALID_PARAMETER	*Reset* is **FALSE** and *New* is **NULL**.
EFI_DEVICE_ERROR	The Simple Network Protocol interface has not been initialized by calling **Initialize()**.
EFI_DEVICE_ERROR	An error occurred attempting to set the new station address.
EFI_UNSUPPORTED	The NIC does not support changing the network interface's station address.

15-16.1 Change the "Status Codes Returned" table (page 15-16)

In section 15.1, "EFI_SIMPLE_NETWORK Protocol – Statistics()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The statistics were collected from the network interface.
EFI_NOT_STARTED	The network interface has not been started.
EFI_BUFFER_TOO_SMALL	The *Statistics* buffer was too small. The current buffer size needed to hold the statistics is returned in *StatisticsSize*.
EFI_INVALID_PARAMETER	One or more of the parameters has an unsupported value.
EFI_DEVICE_ERROR	The command could not be sent to the network interface.
EFI_UNSUPPORTED	This function is not supported by the network interface.

TO:

EFI_SUCCESS	The requested operation succeeded.
EFI_NOT_STARTED	The Simple Network Protocol interface has not been started by calling **Start()**.
EFI_BUFFER_TOO_SMALL	*StatisticsSize* is not **NULL** and *StatisticsTable* is **NULL**. The current buffer size that is needed to hold all the statistics is returned in *StatisticsSize*.
EFI_BUFFER_TOO_SMALL	*StatisticsSize* is not **NULL** and *StatisticsTable* is not **NULL**. The current buffer size that is needed to hold all the statistics is returned in *StatisticsSize*. A partial set of statistics is returned in *StatisticsTable*.
EFI_INVALID_PARAMETER	*StatisticsSize* is **NULL** and *StatisticsTable* is not **NULL**.
EFI_DEVICE_ERROR	The Simple Network Protocol interface has not been initialized by calling **Initialize()**.
EFI_DEVICE_ERROR	An error was encountered collecting statistics from the NIC.
EFI_UNSUPPORTED	The NIC does not support collecting statistics from the network interface.

15-17.1 Change the "Status Codes Returned" table (page 15-17)

In section 15.1, "EFI_SIMPLE_NETWORK Protocol – MCastIpToMac()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The multicast IP address was mapped to the multicast HW MAC address.
EFI_NOT_STARTED	The network interface has not been started.
EFI_INVALID_PARAMETER	One or more of the parameters has an unsupported value.
EFI_DEVICE_ERROR	The command could not be sent to the network interface.
EFI_UNSUPPORTED	This function is not supported by the network interface.

TO:

EFI_SUCCESS	The multicast IP address was mapped to the multicast HW MAC address.
EFI_NOT_STARTED	The Simple Network Protocol interface has not been started by calling **Start()**.
EFI_INVALID_PARAMETER	*IP* is **NULL**.
EFI_INVALID_PARAMETER	*MAC* is **NULL**.
EFI_INVALID_PARAMETER	*IP* does not point to a valid IPv4 or IPv6 multicast address.
EFI_DEVICE_ERROR	The Simple Network Protocol interface has not been initialized by calling **Initialize()**.
EFI_UNSUPPORTED	*IPv6* is **TRUE** and the implementation does not support IPv6 multicast to MAC address conversion.

15-50.1 Change the *BufferSize* parameter in the "Prototype" subsection from UINTN to UINT64 (page 15-50)

In section 15.3, "PXE Base Code Protocol – Mtftp()," change the *BufferSize* parameter in the "Prototype" subsection from **UINTN** to **UINT64**, so it changes FROM:

```
EFI_STATUS
(EFIAPI *EFI_PXE_BASE_CODE_MTFTP) (
      IN EFI_PXE_BASE_CODE                  *This,
      IN EFI_PXE_BASE_CODE_TFTP_OPCODE      Operation,
      IN OUT VOID                           *BufferPtr, OPTIONAL
      IN BOOLEAN                            Overwrite,
      IN OUT UINTN                          *BufferSize,
      IN UINTN                              *BlockSize, OPTIONAL
      IN EFI_IP_ADDRESS                     *ServerIp,
      IN CHAR8                              *Filename, OPTIONAL
      IN EFI_PXE_BASE_CODE_MTFTP_INFO       *Info, OPTIONAL
      IN BOOLEAN                            DontUseBuffer
);
```

TO:

```
EFI_STATUS
(EFIAPI *EFI_PXE_BASE_CODE_MTFTP) (
  IN EFI_PXE_BASE_CODE                  *This,
  IN EFI_PXE_BASE_CODE_TFTP_OPCODE      Operation,
  IN OUT VOID                           *BufferPtr,   OPTIONAL
  IN BOOLEAN                            Overwrite,
  IN OUT UINT64                         *BufferSize,
  IN UINTN                              *BlockSize,   OPTIONAL
  IN EFI_IP_ADDRESS                     *ServerIp,
  IN CHAR8                              *Filename,    OPTIONAL
  IN EFI_PXE_BASE_CODE_MTFTP_INFO       *Info,        OPTIONAL
  IN BOOLEAN                            DontUseBuffer
  );
```

Chapter 17: Protocols – Compression Algorithm Specification

17-18.1 Change the "Status Codes Returned" table (page 17-18)

In section 17.5, "EFI_DECOMPRESS_PROTOCOL – GetInfo()," change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The size of the uncompressed data was returned in *DestinationSize* and the size of the scratch buffer was returned in *ScratchSize*.
EFI_INVALID_PARAMETER	The size of the uncompressed data or the size of the scratch buffer cannot be determined from the compressed data specified by *Source* and *SourceData*.

TO:

EFI_SUCCESS	The size of the uncompressed data was returned in *DestinationSize* and the size of the scratch buffer was returned in *ScratchSize*.
EFI_INVALID_PARAMETER	The size of the uncompressed data or the size of the scratch buffer cannot be determined from the compressed data specified by *Source* and *SourceSize*.

Chapter 18: Protocols – Device I/O Protocol

18-3.1 Change the definition of EFI_IO_WIDTH in the "Related Definitions" subsection (page 18-3)

In section 18.2, "DEVICE_IO Protocol," change the definition of **EFI_IO_WIDTH** in the "Related Definitions" subsection FROM:

```
typedef enum {
      IO_UINT8  = 0,
      IO_UINT16 = 1,
      IO_UINT32 = 2,
      IO_UINT64 = 3
} EFI_IO_WIDTH;
```

TO:

```
typedef enum {
  IO_UINT8  = 0,
  IO_UINT16 = 1,
  IO_UINT32 = 2,
  IO_UINT64 = 3,
  MMIO_COPY_UINT8 = 4,
  MMIO_COPY_UINT16 = 5,
  MMIO_COPY_UINT32 = 6,
  MMIO_COPY_UINT64 = 7
} EFI_IO_WIDTH;
```

18-5.1 Change the description of *Buffer* in the "Parameters" subsection (page 18-5)

In section 18.2, "DEVICE_IO Protocol – Mem(), .Io(), and .Pci()," change the description of *Buffer* in the "Parameters" subsection FROM:

Buffer	For read operations, the destination buffer to store the results. For write operations, the source buffer to write data from.

TO:

Buffer	For read operations, the destination buffer to store the results. For write operations, the source buffer to write data from. If *Width* is **MMIO_COPY_UINT8**, **MMIO_COPY_UINT16**, **MMIO_COPY_UINT32**, or **MMIO_COPY_UINT64**, then *Buffer* is interpreted as a base address of an I/O operation such as *Address*.

Chapter 19: EFI Byte Code Virtual Machine

19-61.1 Replace "PE32+" with "PE32" (page 19-61)

In section 19.9.5, "Binary Format," replace "PE32+" with "PE32" in the first sentence.

19-62.1 Replace "PE32+" with "PE32" (page 19-62)

In section 19.10.1, "EBC Image Requirements," replace "PE32+" with "PE32" in the first sentence.

19-65.1 Replace "PE32+" with "PE32" (page 19-65)

In section 19.11, "EBC Interpreter Protocol – CreateThunk()," replace "PE32+" with "PE32" in the first sentence of the "Description" subsection.

19-68.1 Replace "PE32+" with "PE32" (page 19-68)

In section 19.11, "EBC Interpreter Protocol – RegisterIcacheFlush()," replace "PE32+" with "PE32" in the first sentence of the "Description" subsection.

19-74.1 Replace “PE32+” with “PE32” (page 19-74)

In section 19.12.11, “EBC Linker,” replace “PE32+” with “PE32” in the second normal text (non-code) paragraph, so it changes FROM:

> In addition, the linker must support EBC images with of the following subsystem types as set in a PE32+ optional header:

TO:

> In addition, the linker must support EBC images with of the following subsystem types as set in a PE32 optional header:

Appendix B: Console

B-3.1 Change the PC ANSI code value for Set Mode 80x25 in Table B-2 (page B-3)

In section B.2, “SIMPLE_TEXT_OUTPUT,” change the PC ANSI code value for Set Mode 80x25 in Table B-2 (second-to-last row in the table) FROM:

ESC [3 h	CSI = 3 h	Set Mode 80x25 color.

TO:

ESC [= 3 h	CSI = 3 h	Set Mode 80x25 color.

Appendix E: 32/64-Bit UNDI Specification

E-3.1 Update four RFC numbers in Table E-2 (page E-3)

In section E.1.2, “Referenced Specifications,” change the Request for Comments (RFC) numbers and URLs for the following rows in Table E-2 FROM (unchanged rows are omitted for brevity):

Assigned Numbers	Lists the reserved numbers used in the RFCs and in this specification - http://www.ietf.org/rfc/rfc1700.txt
BOOTP	**Bootstrap Protocol** – http://www.ietf.org/rfc/rfc0951.txt. - This reference is included for backward compatibility. BC protocol supports DHCP and BOOTP. Required reading for those implementing the BC protocol or PXE Bootservers..
DHCP	**Dynamic Host Configuration Protocol** DHCP for Ipv4 (protocol: http://www.ietf.org/rfc/rfc2131.txt, options: http://www.ietf.org/rfc/rfc2132.txt) Required reading for those implementing the BC protocol or PXE Bootservers.
IGMP	**Internet Group Management Protocol** – http://www.ietf.org/rfc/rfc2236.txt Required reading for those implementing the BC protocol.

TO:

Assigned Numbers	Lists the reserved numbers used in the RFCs and in this specification - http://www.ietf.org/rfc/rfc3232.txt
BOOTP	**Bootstrap Protocol** – http://www.ietf.org/rfc/rfc0951.txt, http://www.ietf.org/rfc/rfc1542.txt, and http://www.ietf.org/rfc/rfc1534.txt. - These references are included for backward compatibility. BC protocol supports DHCP and BOOTP. Required reading for those implementing the BC protocol or PXE Bootservers.
DHCP	**Dynamic Host Configuration Protocol** DHCP for Ipv4 (protocol: http://www.ietf.org/rfc/rfc2131.txt, options: http://www.ietf.org/rfc/rfc2132.txt, http://www.ietf.org/rfc/rfc3203.txt, http://www.ietf.org/rfc/rfc3396.txt, http://www.ietf.org/rfc/rfc1534.txt) Required reading for those implementing the BC protocol or PXE Bootservers.
IGMP	**Internet Group Management Protocol** – http://www.ietf.org/rfc/rfc3376.txt. Required reading for those implementing the BC protocol.

E-3.2 Change “BC protocol” to “PXE Base Code Protocol” in Table E-2 (page E-3)

In section E.1.2, “Referenced Specifications,” change “BC protocol” to “PXE Base Code Protocol” in the following rows in Table E-2 FROM (unchanged rows are omitted for brevity):

ARP	**Address Resolution Protocol** – http://www.ietf.org/rfc/rfc0826.txt. Required reading for those implementing the BC protocol.
BOOTP	**Bootstrap Protocol** – http://www.ietf.org/rfc/rfc0951.txt. - This reference is included for backward compatibility. BC protocol supports DHCP and BOOTP. Required reading for those implementing the BC protocol or PXE Bootservers.
DHCP	**Dynamic Host Configuration Protocol** DHCP for Ipv4 (protocol: http://www.ietf.org/rfc/rfc2131.txt, options: http://www.ietf.org/rfc/rfc2132.txt) Required reading for those implementing the BC protocol or PXE Bootservers.
ICMP	**Internet Control Message Protocol** ICMP for Ipv4: http://www.ietf.org/rfc/rfc0792.txt ICMP for Ipv6: http://www.ietf.org/rfc/rfc2463.txt Required reading for those implementing the BC protocol.
IGMP	**Internet Group Management Protocol** – http://www.ietf.org/rfc/rfc2236.txt Required reading for those implementing the BC protocol.
IP	**Internet Protocol** Ipv4: http://www.ietf.org/rfc/rfc0791.txt Ipv6: http://www.ietf.org/rfc/rfc2460.txt and http://www.ipv6.org Required reading for those implementing the BC protocol.
MTFTP	**Multicast TFTP** – Defined in the 16-bit PXE specification. Required reading for those implementing the BC protocol.

TO (includes changes listed in Specification Clarification E-3.1 above):

ARP	**Address Resolution Protocol** – http://www.ietf.org/rfc/rfc0826.txt. Required reading for those implementing the PXE Base Code Protocol.
BOOTP	**Bootstrap Protocol** – http://www.ietf.org/rfc/rfc0951.txt, http://www.ietf.org/rfc/rfc1542.txt, and http://www.ietf.org/rfc/rfc1534.txt. - These references are included for backward compatibility. BC protocol supports DHCP and BOOTP. Required reading for those implementing the PXE Base Code Protocol or PXE Bootservers.
DHCP	**Dynamic Host Configuration Protocol** DHCP for Ipv4 (protocol: http://www.ietf.org/rfc/rfc2131.txt, options: http://www.ietf.org/rfc/rfc2132.txt, http://www.ietf.org/rfc/rfc3203.txt, http://www.ietf.org/rfc/rfc3396.txt, http://www.ietf.org/rfc/rfc1534.txt) Required reading for those implementing the PXE Base Code Protocol or PXE Bootservers.
ICMP	**Internet Control Message Protocol** ICMP for Ipv4: http://www.ietf.org/rfc/rfc0792.txt ICMP for Ipv6: http://www.ietf.org/rfc/rfc2463.txt Required reading for those implementing the BC protocol.
IGMP	**Internet Group Management Protocol** – http://www.ietf.org/rfc/rfc3376.txt. Required reading for those implementing the PXE Base Code Protocol.
IP	**Internet Protocol** Ipv4: http://www.ietf.org/rfc/rfc0791.txt Ipv6: http://www.ietf.org/rfc/rfc2460.txt and http://www.ipv6.org Required reading for those implementing the PXE Base Code Protocol.
MTFTP	**Multicast TFTP** – Defined in the 16-bit PXE specification. Required reading for those implementing the PXE Base Code Protocol.

E-4.1 Change “BC protocol” to “PXE Base Code Protocol” in Table E-2 (page E-4)

In section E.1.2, “Referenced Specifications,” change “BC protocol” to “PXE Base Code Protocol” in the following rows in Table E-2 FROM (unchanged rows are omitted for brevity):

TCP	**Transmission Control Protocol** TCPv4: http://www.ietf.org/rfc/rfc0793.txt TCPv6: ftp://ftp.ipv6.org/pub/rfc/rfc2147.txt Required reading for those implementing the BC protocol.
TFTP	**Trivial File Transfer Protocol** TFTP (protocol: http://www.ietf.org/rfc/rfc1350.txt, options: http://www.ietf.org/rfc/rfc2347.txt, http://www.ietf.org/rfc/rfc2348.txt, and http://www.ietf.org/rfc/rfc2349.txt). Required reading for those implementing the BC protocol.
UDP	**User Datagram Protocol** UDP over IPv4: http://www.ietf.org/rfc/rfc0768.txt UDP over IPv6: http://www.ietf.org/rfc/rfc2454.txt Required reading for those implementing the BC protocol.
WfM	**Wired for Management** http://www.intel.com/labs/manage/wfm/wfmspecs.htm Recommended reading for those implementing the BC protocol or PXE Bootservers.

TO:

TCP	**Transmission Control Protocol** TCPv4: http://www.ietf.org/rfc/rfc0793.txt TCPv6: ftp://ftp.ipv6.org/pub/rfc/rfc2147.txt Required reading for those implementing the PXE Base Code Protocol.
TFTP	**Trivial File Transfer Protocol** TFTP (protocol: http://www.ietf.org/rfc/rfc1350.txt, options: http://www.ietf.org/rfc/rfc2347.txt, http://www.ietf.org/rfc/rfc2348.txt, and http://www.ietf.org/rfc/rfc2349.txt). Required reading for those implementing the PXE Base Code Protocol.
UDP	**User Datagram Protocol** UDP over IPv4: http://www.ietf.org/rfc/rfc0768.txt UDP over IPv6: http://www.ietf.org/rfc/rfc2454.txt Required reading for those implementing the PXE Base Code Protocol.
WfM	**Wired for Management** http://www.intel.com/labs/manage/wfm/wfmspecs.htm Recommended reading for those implementing the PXE Base Code Protocol or PXE Bootservers.

E-32.1 Replace “RFC 1700” with “RFC 3232” (page E-32)

In section E.3.4.12, “PXE_IFTYPE,” update the RFC number in the C code snippet FROM:

```
// This information is from the ARP section of RFC 1700.
```

TO:

```
// This information is from the ARP section of RFC 3232.
```

E-38.1 Change the direction of the Shutdown and Stop state transition arrows in Figure E-6 (page E-38)

In section E.4, “UNDI Commands,” change the direction of the Shutdown and Stop state transition arrows in Figure E-6, so that it shows the following:

Shutdown changes the state from Initialized to Started.

Stop changes the state from Started to Stopped.

E-41.1 Replace the last two paragraphs in section E.4.2 (page E-41)

In section E.4.2, “Get State,” change the last two paragraphs in this section FROM:

Drivers, NBPs, and applications should not use UNDIs that are already started or initialized.

No other operational checks are made by this command. If this is a S/W UNDI, the **`PXE_START_CPB.Delay()`** and **`PXE_START_CPB.Virt2Phys()`** callbacks will not be used.

TO:

Drivers and applications must not start using UNDIs that have been placed into the Started or Initialized states by another driver or application.

3.0 and 3.1 S/W UNDI: No callbacks are performed by this UNDI command.

E-44.1 Replace the “Preparing the CPB” subsection in its entirety (page E-44)

In section E.4.3.1, “Issuing the Command,” replace the “Preparing the CPB” subsection in its entirety with the following:

Preparing the CPB

For the 3.1 S/W UNDI Start command, the CPB structure shown below must be filled in and the CDB must be set to **`sizeof(struct s_pxe_cpb_start_31)`**.

```
#pragma pack(1)
typedef struct s_pxe_cpb_start_31 {
  UINT64    Delay;
  //
  // Address of the Delay() callback service.
  // This field cannot be set to zero.
  //
  // VOID
  // Delay(
  //   IN  UINT64    UniqueId,
  //   IN  UINT64    Microseconds);
  //
  // UNDI will never request a delay smaller than 10 microseconds
  // and will always request delays in increments of 10
  // microseconds. The Delay() callback routine must delay
  // between n and n + 10 microseconds before returning control
  // to the UNDI.
  //

  UINT64    Block;
  //
  // Address of the Block() callback service.
  // This field cannot be set to zero.
  //
  // VOID
  // Block(
  //   IN  UINT64    UniqueId,
  //   IN  UINT32    Enable);
  //
  // UNDI may need to block multithreaded/multiprocessor access
  // to critical code sections when programming or accessing the
  // network device.  When UNDI needs a block, it will call the
  // Block()callback service with Enable set to a non-zero value.
  // When UNDI no longer needs the block, it will call Block()
  // with Enable set to zero.
  //

  UINT64    Virt2Phys;
```

```
  //
  // Convert a virtual address to a physical address.
  // This field can be set to zero if virtual and physical
  // addresses are identical.
  //
  // VOID
  // Virt2Phys(
  //  IN  UINT64    UniqueId,
  //  IN  UINT64    Virtual,
  //  OUT UINT64    PhysicalPtr);
  //
  // UNDI will pass in a virtual address and a pointer to storage
  // for a physical address.  The Virt2Phys() service converts
  // the virtual address to a physical address and stores the
  // resulting physical address in the supplied buffer.  If no
  // conversion is needed, the virtual address must be copied
  // into the supplied physical address buffer.
  //

  UINT64    MemIo;
  //
  // Read/Write network device memory and/or I/O register space.
  // This field cannot be set to zero.
  //
  // VOID
  // MemIo(
  //  IN     UINT64    UniqueId,
  //  IN     UINT8     AccessType,
  //  IN     UINT8     Length,
  //  IN     UINT64    Port,
  //  IN OUT UINT64    BufferPtr);
  //
  // UNDI uses the MemIo() service to access the network device
  // memory and/or I/O registers.  The AccessType is one of the
  // PXE_IO_xxx or PXE_MEM_xxx constants defined at the end of
  // this section.  The Length is 1, 2, 4 or 8.  The Port number
  // is relative to the base memory or I/O address space for this
  // device.BufferPtr points to the data to be written to the
  // Port or will contain the data that is read from the Port.
  //

  UINT64    MapMem;
  //
  // Map virtual memory address for DMA.
  // This field can be set to zero if there is no mapping
  // service.
  //
  // VOID
  // MapMem(
```

```
  //   IN  UINT64    UniqueId,
  //   IN  UINT64    Virtual,
  //   IN  UINT32    Size,
  //   IN  UINT32    Direction,
  //   OUT UINT64    PhysicalPtr);
  //
  // When UNDI needs to perform a DMA transfer it will request a
  // virtual-to-physical mapping using the MapMem() service.  The
  // Virtual parameter contains the virtual address to be mapped.
  // The minimum Size of the virtual memory buffer to be mapped.
  // Direction is one of the TO_DEVICE, FROM_DEVICE or
  // TO_AND_FROM_DEVICE constants defined at the end of this
  // section.PhysicalPtr contains the mapped physical address or
  // a copy of the Virtual address if no mapping is required.
  //

  UINT64    UnMapMem;
  //
  // Un-map previously mapped virtual memory address.
  // This field can be set to zero only if the MapMem() service
  // is also set to zero.
  //
  // VOID
  // UnMapMem(
  //   IN  UINT64    UniqueId,
  //   IN  UINT64    Virtual,
  //   IN  UINT32    Size,
  //   IN  UINT32    Direction,
  //   IN  UINT64    PhysicalPtr);
  //
  // When UNDI is done with the mapped memory, it will use the
  // UnMapMem() service to release the mapped memory.
  //

  UINT64    SyncMem;
  //
  // Synchronise mapped memory.
  // This field can be set to zero only if the MapMem() service
  // is also set to zero.
  //
  // VOID
  // SyncMem(
  //   IN  UINT64    UniqueId,
  //   IN  UINT64    Virtual,
  //   IN  UINT32    Size,
  //   IN  UINT32    Direction,
  //   IN  UINT64    PhysicalPtr);
  //
  // When the virtual and physical buffers need to be
```

```
  // synchronized, UNDI will call the SyncMem() service.
  //

  UINT64    UniqueId;
  //
  // UNDI will pass this value to each of the callback services.
  // A unique ID number should be generated for each instance of
  // the UNDI driver that will be using these callback services.
  //
} PXE_CPB_START_31;
#pragma pack()
```

For the 3.0 S/W UNDI Start command, the CPB structure shown below must be filled in and the CDB must be set to **`sizeof(struct s_pxe_cpb_start_30)`**.

```
#pragma pack(1)
typedef struct s_pxe_cpb_start_30 {
  UINT64    Delay;
  //
  // Address of the Delay() callback service.
  // This field cannot be set to zero.
  //
  // VOID
  // Delay(
  //   IN  UINT64    Microseconds);
  //
  // UNDI will never request a delay smaller than 10 microseconds
  // and will always request delays in increments of 10.
  // microseconds The Delay() callback routine must delay between
  // n and n + 10 microseconds before returning control to the
  // UNDI.
  //

  UINT64    Block;
  //
  // Address of the Block() callback service.
  // This field cannot be set to zero.
  //
  // VOID
  // Block(
  //   IN  UINT32    Enable);
  //
  // UNDI may need to block multithreaded/multiprocessor access
  // to critical code sections when programming or accessing the
  // network device.  When UNDI needs a block, it will call the
  // Block()callback service with Enable set to a non-zero value.
  // When UNDI no longer needs the block, it will call Block()
```

```
  // with Enable set to zero.
  //

  UINT64    Virt2Phys;
  //
  // Convert a virtual address to a physical address.
  // This field can be set to zero if virtual and physical
  // addresses are identical.
  //
  // VOID
  // Virt2Phys(
  //  IN  UINT64    Virtual,
  //  OUT UINT64    PhysicalPtr);
  //
  // UNDI will pass in a virtual address and a pointer to storage
  // for a physical address.  The Virt2Phys() service converts
  // the virtual address to a physical address and stores the
  // resulting physical address in the supplied buffer.  If no
  // conversion is needed, the virtual address must be copied
  // into the supplied physical address buffer.
  //

  UINT64    MemIo;
  //
  // Read/Write network device memory and/or I/O register space.
  // This field cannot be set to zero.
  //
  // VOID
  // MemIo(
  //  IN     UINT8     AccessType,
  //  IN     UINT8     Length,
  //  IN     UINT64    Port,
  //  IN OUT UINT64    BufferPtr);
  //
  // UNDI uses the MemIo() service to access the network device
  // memory and/or I/O registers.  The AccessType is one of the
  // PXE_IO_xxx or PXE_MEM_xxx constants defined at the end of
  // this section.  The Length is 1, 2, 4 or 8.  The Port number
  // is relative to the base memory or I/O address space for this
  // device.BufferPtr points to the data to be written to the
  // Port or will contain the data that is read from the Port.
  //
} PXE_CPB_START_30;
#pragma pack()
```

E-52.1 Change three `#define` statements (page E-52)

In section E.4.5.3, “Checking Command Execution Results,” change the following three **`#define`** statements in the “DB” subsection FROM:

```
#define PXE_DUPLEX_AUTO_DETECT_SUPPORTED  1
define PXE_DUPLEX_FORCE_FULL_SUPPORTED    2
#define PXE_DUPLEX_FORCE_HALF_SUPPORTED   4
```

TO:

```
#define PXE_DUPLEX_DEFAULT                0
#define PXE_DUPLEX_ENABLE_FULL_SUPPORTED  1
#define PXE_DUPLEX_FORCE_FULL_SUPPORTED   2
```

Glossary

Gloss-4.1 Replace “PE32+” with “PE32” in the EBC Image definition (page Glossary-4)

In the *EBC Image* definition in the “Glossary” section, replace “PE32+” with “PE32.”

Chapter 3: Boot Manager

3-5.1 Delete the hyphen in the heading text of section 3.2 (page 3-5)

Delete the hyphen in the heading of section 3.2, “Globally-Defined Variables.”

Chapter 9: Protocols – EFI Driver Model

9-42.1 Change “attempt” to “attempting” in the EFI_DEVICE_ERROR status code description (page 9-42)

In section 9.4, “EFI Driver Configuration Protocol – ForceDefaults(),” change “attempt” to “attempting” in the description of EFI_DEVICE_ERROR in the “Status Codes Returned” table, so it changes FROM:

EFI_DEVICE_ERROR	A device error occurred while attempt to force the default configuration options on the controller specified by *ControllerHandle* and *ChildHandle*.

TO:

EFI_DEVICE_ERROR	A device error occurred while attempting to force the default configuration options on the controller specified by *ControllerHandle* and *ChildHandle*.

Chapter 12: Protocols – PCI Bus Support

12-96.1 Change “being in” to “begin on” in the tenth bullet (page 12-96)

In section 12.4.2, “PCI Options ROMs,” change “being in” to “begin on” in the tenth bullet, so it changes FROM:

- The PCIR data structure must being in a 4-byte boundary.

TO:

- The PCIR data structure must begin on a 4-byte boundary.

12-96.2 Change “is” to “in” in the first sentence of the twelfth bullet (page 12-96)

In section 12.4.2, “PCI Options ROMs,” change “is” to “in” in the first sentence of the twelfth bullet, so it changes FROM:

- The images are placed in the PCI Option ROM is order from highest to lowest priority.

TO:

- The images are placed in the PCI Option ROM in order from highest to lowest priority.