UEFI Specification 2.0, Errata

1) Throughout:

Add clarification to the spec so that we avoid references to GUIDs that do not comply to the <32bit><16bit><16bit><byte><byte><byte><byte><byte><byte><byte><byte> format.

EFI_GLOBAL_VARIABLE

GUID

```
#define EFI_GLOBAL_VARIABLE \

{0x8BE4DF61,0x93CA,0x11d2,0xAA,0x0D,0x00,0xE0,0x98,0x03,0x2
B,0x8C}
```

EFI_SIMPLE_TEXT_INPUT_PROTOCOL_GUID

GUID

```
#define EFI_SIMPLE_TEXT_INPUT_PROTOCOL_GUID \

{0x387477c1,0x69c7,0x11d2,0x8e,0x39,0x00,0xa0,0xc9,0x69,0x7
2,0x3b}
```

EFI_LOAD_FILE_PROTOCOL_GUID

GUID

```
#define EFI_LOAD_FILE_PROTOCOL_GUID \

{0x56EC3091,0x954C,0x11d2,0x8E,0x3F,0x00,0xA0,0xC9,0x69,0x7
2,0x3B}
```

EFI_SIMPLE_NETWORK_PROTOCOL_GUID

GUID

```
#define EFI_SIMPLE_NETWORK_PROTOCOL_GUID \

{0xA19832B9,0xAC25,0x11D3,0x9A,0x2D,0x00,0x90,0x27,0x3f,0xc
1,0x4d}
```

EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL_GUID

GUID

```
#define EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL_GUID   \

 {0xf36ff770,0xa7e1,0x42cf,0x9e,0xd2,0x56,0xf0,0xf2,0x71,0xf4, 0x4c}
```

EFI_ARP_SERVICE_BINDING_PROTOCOL_GUID

GUID

```
#define EFI_ARP_SERVICE_BINDING_PROTOCOL_GUID  \
{0xf44c00ee,0x1f2c,0x4a00,0xaa,0x09,0x1c,0x9f,0x3e,0x08,0x00,
0xa3}
```

EFI_ARP_PROTOCOL_GUID

GUID

```
#define EFI_ARP_PROTOCOL_GUID    \
{0xf4b427bb,0xba21,0x4f16,0xbc,0x4e,0x43,0xe4,0x16,0xab,0x61,
0x9c}
```

EFI_SERIAL_IO_PROTOCOL_GUID

GUID

```
#define EFI_SERIAL_IO_PROTOCOL_GUID \

{0xBB25CF6F,0xF1D4,0x11D2,0x9A,0x0C,0x00,0x90,0x27,0x3F,0xC1,
0xFD}
```

EFI_DEVICE_PATH_PROTOCOL_GUID

GUID

```
#define EFI_DEVICE_PATH_PROTOCOL_GUID \

{0x09576e91,0x6d3f,0x11d2,0x8e,0x39,0x00,0xa0,0xc9,0x69,0x72,
0x3b}
```

EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL_GUID

GUID

```
#define EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL_GUID \

{0x387477c2,0x69c7,0x11d2,0x8e,0x39,0x00,0xa0,0xc9,0x69,0x72,
0x3b}
```

EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID

GUID

```
#define EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID \

{0x0964e5b22,0x6459,0x11d2,0x8e,0x39,0x00,0xa0,0xc9,0x69,0x72,
0x3b}
```

EFI_DISK_IO_PROTOCOL_GUID

GUID

```
#define EFI_DISK_IO_PROTOCOL_GUID       \

{0xCE345171,0xBA0B,0x11d2,0x8e,0x4F,0x00,0xa0,0xc9,0x69,0x72,
0x3b}
```

EFI_BLOCK_IO_PROTOCOL_GUID

GUID

```
#define EFI_BLOCK_IO_PROTOCOL_GUID      \

{0x964e5b21,0x6459,0x11d2,0x8e,0x39,0x00,0xa0,0xc9,0x69,0x72,
0x3b}
```

EFI_UNICODE_COLLATION_PROTOCOL_GUID

GUID

```
#define EFI_UNICODE_COLLATION_PROTOCOL_GUID \

{0x1d85cd7f,0xf43d,0x11d2,0x9a,0x0c,0x00,0x90,0x27,0x3f,0xc1,
0x4d}
```

EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL_GUID

GUID

```
#define EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL_GUID \

{0xE18541CD,0xF755,0x4f73,0x92,0x8D,0x64,0x3C,0x8A,0x79,0xB2,
0x29}
```

EFI_PXE_BASE_CODE_PROTOCOL_GUID

GUID

```
#define EFI_PXE_BASE_CODE_PROTOCOL_GUID \

{0x03C4E603,0xAC28,0x11d3,0x9A,0x2D,0x00,0x90,0x27,0x3F,0xC1,
0x4D}
```

EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL_GUID

GUID

```
#define EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL_GUID \

{0x245DCA21,0xFB7B,0x11d3,0x8F,0x01,0x00,0xA0,0xC9,0x69,0x72,
0x3B}
```

EFI_MANAGED_NETWORK_PROTOCOL_GUID

GUID

```
#define EFI_MANAGED_NETWORK_PROTOCOL_GUID      \

{0x3b95aa31,0x3793,0x434b,0x86,0x67,0xc8,0x07,0x08,0x92,0xe0,
0x5e}
```

EFI_DHCP4_PROTOCOL_GUID

GUID

```
#define EFI_DHCP4_PROTOCOL_GUID   \

    {0x8a219718,0x4ef5,0x4761,0x91,0xc8,0xc0,0xf0,0x4b,0xda,0x9e,
0x56}
```

EFI_DHCP4_SERVICE_BINDING_PROTOCOL_GUID

GUID

```
#define EFI_DHCP4_SERVICE_BINDING_PROTOCOL_GUID      \

{0x9d9a39d8,0xbd42,0x4a73,0xa4,0xd5,0x8e,0xe9,0x4b,0xe1,0x13,
0x80}
```

EFI_TCP4_PROTOCOL_GUID

GUID

```
#define EFI_TCP4_PROTOCOL_GUID    \

{0x65530BC7,0xA359,0x410f,0xB0,0x10,0x5A,0xAD,0xC7,0xEC,0x2B,
0x62}
```

EFI_TCP4_SERVICE_BINDING_PROTOCOL_GUID

GUID

```
#define EFI_TCP4_SERVICE_BINDING_PROTOCOL_GUID       \

    {0x00720665,0x67EB,0x4a99,0xBA,0xF7,0xD3,0xC3,0x3A,0x1C,0x7C,
0xC9}
```

EFI_IP4_PROTOCOL_GUID

GUID

```
#define EFI_IP4_PROTOCOL_GUID    \

{0x41d94cd2,0x35b6,0x455a,0x82,0x58,0xd4,0xe5,0x13,0x34,0xaa,
0xdd}
```

EFI_IP4_SERVICE_BINDING_PROTOCOL_GUID

GUID

```
#define EFI_IP4_SERVICE_BINDING_PROTOCOL_GUID  \

{0xc51711e7,0xb4bf,0x404a,0xbf,0xb8,0x0a,0x04,0x8e,0xf1,0xff,
0xe4}
```

EFI_IP4_CONFIG_PROTOCOL_GUID

GUID

```
#define EFI_IP4_CONFIG_PROTOCOL_GUID    \

{0x3b95aa31,0x3793,0x434b,0x86,0x67,0xc8,0x07,0x08,0x92,0xe0,
0x5e}
```

EFI_UDP4_PROTOCOL_GUID

GUID

```
#define EFI_UDP4_PROTOCOL_GUID    \

{0x3ad9df29,0x4501,0x478d,0xb1,0xf8,0x7f,0x7f,0xe7,0x0e,0x50,
0xf3}
```

EFI_UDP4_SERVICE_BINDING_PROTOCOL_GUID

GUID

```
#define EFI_UDP4_SERVICE_BINDING_PROTOCOL_GUID       \

{0x83f01464,0x99bd,0x45e5,0xb3,0x83,0xaf,0x63,0x05,0xd8,0xe9,
0xe6}
```

EFI_MTFTP4_PROTOCOL_GUID

GUID

```
#define EFI_MTFTP4_PROTOCOL_GUID \

{0x78247c57,0x63db,0x4708,0x99,0xc2,0xa8,0xb4,0xa9,0xa6,0x1f,0x6b}
```

EFI_MTFTP4_SERVICE_BINDING_PROTOCOL_GUID

GUID

```
#define EFI_MTFTP4_SERVICE_BINDING_PROTOCOL_GUID     \

{0x2FE800BE,0x8F01,0x4aa6,0x94,0x6B,0xD7,0x13,0x88,0xE1,0x83,
0x3F}
```

EFI_AUTHENTICATION_CHAP_RADIUS_GUID

GUID

```
#define EFI_AUTHENTICATION_CHAP_RADIUS_GUID \

{0xd6062b50,0x15ca,0x11da,0x92,0x19,0x00,0x10,0x83,0xff,0xca,
0x4d}
```

EFI_AUTHENTICATION_CHAP_LOCAL_GUID

GUID

```
#define EFI_AUTHENTICATION_CHAP_LOCAL_GUID \

{0xc280c73e,0x15ca,0x11da,0xb0,0xca,0x00,0x10,0x83,0xff,0xca,
0x4d}
```

2) Page 26, Section 2.3.2, IA-32 Platforms.Replace the NOTE with the following:

Note: *Note: Previous EFI specifications allowed ACPI tables loaded at runtime to be in the EfiReservedMemoryType and there was no guidance provided for other EFI Configuration Tables. EfiReservedMemoryType is not intended to be used for the storage of any EFI Configuration Tables. UEFI 2.0 intends to clarify the situation moving forward. Also, only OSes conforming to UEFI 2.0 are guaranteed to handle SMBIOS tables in memory of type EfiBootServicesData.*

3) PAGE 35, Table 6. Delete DEVICE_IO as an UEFI protocol.

4) Page 69, Section 4.3, EFI_System _Table, Related Definitions.

Add "**`#define EFI_SPECIFICATION_VERSION  EFI_SYSTEM_TABLE_REVISION`**" and change "**`#define EFI_SYSTEM_TABLE_REVISION  ((2<<16) | (10))`**" to "**`#define EFI_SYSTEM_TABLE_REVISION  EFI_2_10_SYSTEM_TABLE_REVISION`**"

```
#define EFI_SYSTEM_TABLE_SIGNATURE 0x5453595320494249
#define EFI_2_10_SYSTEM_TABLE_REVISION ((2<<16) | (10))
#define EFI_2_00_SYSTEM_TABLE_REVISION ((2<<16) | (00))
#define EFI_1_10_SYSTEM_TABLE_REVISION ((1<<16) | (10))
#define EFI_1_02_SYSTEM_TABLE_REVISION ((1<<16) | (02))
#define EFI_SYSTEM_TABLE_REVISION  EFI_2_10_SYSTEM_TABLE_REVISION
#define EFI_SPECIFICATION_VERSION  EFI_SYSTEM_TABLE_REVISION
```

5) Page 71, Section 4.4, , Related Definitions.

Replace "**`#define EFI_BOOT_SERVICES_REVISION ((2<<16) | (00))`**" with "**`#define EFI_BOOT_SERVICES_REVISION  EFI_SPECIFICATION_VERSION`**" to read as follows:

```
#define EFI_BOOT_SERVICES_SIGNATURE 0x56524553544f4f42

#define EFI_BOOT_SERVICES_REVISION  EFI_SPECIFICATION_VERSION
```

6) Page 71, Section 4.5, Related Definitions.

Replace "**`#define EFI_RUNTIME_SERVICES_REVISION ((2<<16) | (00))`**" with "**`#define EFI_RUNTIME_SERVICES_REVISION  EFI_SPECIFICATION_VERSION`**" to read as follows:

```
#define EFI_RUNTIME_SERVICES_SIGNATURE 0x56524553544e5552
#define EFI_RUNTIME_SERVICES_REVISION  EFI_SPECIFICATION_VERSION
```

7) Page 72, Section 4.4.

Member “VOID *Reserved” of EFI_BOOT_SERVICES structure is defined by EFI 1.10 but removed by UEFI 2.0. This is a place holder to keep the boot services table aligned properly. It should be defined in UEFI 2.0 specification. The Protocol Handler Services area of Related Definitions in Section 4.4, EFI Boot Service Table should read as follows:

```
// Protocol Handler Services
//
EFI_INSTALL_PROTOCOL_INTERFACE    InstallProtocolInterface; // EFI 1.0+
EFI_REINSTALL_PROTOCOL_INTERFACE  ReinstallProtocolInterface; // EFI 1.0+
EFI_UNINSTALL_PROTOCOL_INTERFACE  UninstallProtocolInterface; // EFI 1.0+
EFI_HANDLE_PROTOCOL               HandleProtocol; // EFI 1.0+
VOID*                             Reserved; // EFI 1.0+
EFI_REGISTER_PROTOCOL_NOTIFY      RegisterProtocolNotify; // EFI 1.0+
EFI_LOCATE_HANDLE                 LocateHandle; // EFI 1.0+
EFI_LOCATE_DEVICE_PATH            LocateDevicePath; // EFI 1.0+
EFI_INSTALL_CONFIGURATION_TABLE   InstallConfigurationTable; // EFI 1.0+
```

8) Page 123. Add the following NOTE to AllocatePages():

Note: *Note: UEFI Applications, UEFI Drivers, and UEFI OS Loaders must not allocate memory of type EfiReservedMemoryType.*

9) Page 131, add the following NOTE to AllocatePool():

Note: *Note: UEFI Applications, UEFI Drivers, and UEFI OS Loaders must not allocate memory of type EfiReservedMemoryType.*

10) Page 191, Section 6.4, third paragraph.

Change the description into the following, substituting **`UnloadImage()`** for **`Unload()`**:

It is valid to call **`Exit()`** or **`UnloadImage()`** for an image that was loaded by **`LoadImage()`** before calling **`StartImage()`**. This will free the image from memory without having started it.

11) Page 212, Section 7.1, SetVariable() Description and Status Code Returned.

Add a new return status code EFI_NOT_FOUND to SetVariable service to read as follows:

EFI_VARIABLE_NON_VOLATILE variables are stored in fixed hardware that has a limited

storage capacity; sometimes a severely limited capacity. Software should only use a nonvolatile

variable when absolutely necessary. In addition, if software uses a nonvolatile variable it should

use a variable that is only accessible at boot services time if possible.

A variable must contain one or more bytes of *Data*. Using **SetVariable()** with a *DataSize* of zero causes the entire variable to be deleted. The space consumed by the deleted variable may not be available until the next power cycle.

The Attributes have the following usage rules:

- Storage attributes are only applied to a variable when creating the variable. If a preexisting variable is rewritten with different attributes, the result is indeterminate and may vary between implementations. The correct method of changing the attributes of a variable is to delete the variable and recreate it with different attributes. There is one exception to this rule. If a preexisting variable is rewritten with no access attributes specified, the variable will be deleted.
- Setting a data variable with no access attributes, or zero *DataSize* specified, causes it to be deleted.
- Runtime access to a data variable implies boot service access. Attributes that have **EFI_VARIABLE_RUNTIME_ACCESS** set must also have **EFI_VARIABLE_BOOTSERVICE_ACCESS** set. The caller is responsible for following this rule.
- Once ExitBootServices() is performed, data variables that did not have **EFI_VARIABLE_RUNTIME_ACCESS** set are no longer visible to **GetVariable()**.
- Once **ExitBootServices()** is performed, only variables that have **EFI_VARIABLE_RUNTIME_ACCESS** and **EFI_VARIABLE_NON_VOLATILE** set can be set with SetVariable(). Variables that have runtime access but that are not nonvolatile are read-only data variables once ExitBootServices() is performed.

The only rules the firmware must implement when saving a nonvolatile variable is that it has actually been saved to nonvolatile storage before returning **EFI_SUCCESS,** and that a partial save is not performed. If power fails during a call to **SetVariable()** the variable may contain its previous value, or its new value. In addition there is no read, write, or delete security protection.

Status Codes Returned

EFI_SUCCESS	The firmware has successfully stored the variable and its data as defined by the Attributes.
EFI_INVALID_PARAMETER	An invalid combination of attribute bits was supplied, or the *DataSize* exceeds the maximum allowed.
EFI_INVALID_PARAMETER	*VariableName* is an empty Unicode string.
EFI_OUT_OF_RESOURCES	Not enough storage is available to hold the variable and its data.
EFI_DEVICE_ERROR	The variable could not be saved due to a hardware failure.
EFI_WRITE_PROTECTED	The variable in question is read-only.
EFI_NOT_FOUND	The variable trying to be updated or deleted was not found.

12) Page 213, Section 7.1.

Changes to clarify the expected results from the QueryVariable output fields. Prototype and Description should read as follows:

Prototype

```
typedef
EFI_STATUS
QueryVariableInfo (
  IN UINT32                  Attributes,
  OUT UINT64                 *MaximumVariableStorageSize,
  OUT UINT64                 *RemainingVariableStorageSize,
  OUT UINT64                 *MaximumVariableSize
  );
```

Attributes	Attributes bitmask to specify the type of variables on which to return information. Refer to the **GetVariable()** function description.
MaximumVariableStorageSize	On output the maximum size of the storage space available for the EFI variables associated with the attributes specified.
RemainingVariableStorageSize	Returns the remaining size of the storage space available for EFI variables associated with the attributes specified.
MaximumVariableSize	Returns the maximum size of an individual EFI variable associated with the attributes specified.

Description

The **QueryVariableInfo()** function allows a caller to obtain the information about the maximum size of the storage space available for the EFI variables, the remaining size of the storage space available for the EFI variables and the maximum size of each individual EFI variable, associated with the attributes specified.

The *MaximumVariableSize* value will reflect the overhead associated with the saving of a single EFI variable with the exception of the overhead associated with the length of the string name of the EFI variable.

The returned *MaximumVariableStorageSize*, *RemainingVariableStorageSize*, *MaximumVariableSize* information may change immediately after the call based on other runtime activities including asynchronous error events. Also, these values associated with different attributes are not additive in nature.

13) Page 213, Section 7.2.

Correct errors for the PCI device node text representations and clarify the AppendDeviceNode and AppendDevicePath functions regarding what should happen when the device path & device nodes are NULL. The Description should read as follows:

Description

The **QueryVariableInfo()** function allows a caller to obtain the information about the maximum size of the storage space available for the EFI variables, the remaining size of the storage space available for the EFI variables and the maximum size of each individual EFI variable, associated with the attributes specified.

The *RemainingVariableStorageSize* value will reflect the overhead associated with the saving of a single EFI variable with the exception of the overhead associated with the length of the string name of the EFI variable.

The returned *MaximumVariableStorageSize*, *RemainingVariableStorageSize*, *MaximumVariableSize* information may change immediately after the call based on other runtime activities including asynchronous error events. Also, these values associated with different attributes are not additive in nature.

After the system has transitioned into runtime (after **ExitBootServices()** is called), an implementation may not be able to accurately return information about the Boot Services variable store. In such cases, **EFI_INVALID_PARAMETER** should be returned.

14) Page 227, Section 7.4.1, ResetSystem(), Description. Delete last sentence from the fourth paragraph of the Description, to read as follows:

Calling this interface with *ResetType* of **EfiResetShutdown** causes the system to enter a power state equivalent to the ACPI G2/S5 or G3 states. If the system does not support this reset type, then when the system is rebooted, it should exhibit the **EfiResetCold** attributes.

15) Page 230, Section 7.4.3.

The UpdateCapsule API description should read as follows.

```
typedef
EFI_STATUS
UpdateCapsule (
  IN EFI_CAPSULE_HEADER    **CapsuleHeaderArray,
  IN UINTN                 CapsuleCount,
  IN EFI_PHYSICAL_ADDRESS  ScatterGatherList   OPTIONAL
  );
```

16) Page 231, UpdateCapsule(), Related Definitions. This should have **Union** added to the next to last line and formatting corrected, to read as follows:

```
typedef struct (
UINT64                      Length;
union {
     EFI_PHYSICAL_ADDRESS   DataBlock;
     EFI_PHYSICAL_ADDRESS  ContinuationPointer;
    } Union;
) UEFI_CAPSULE_BLOCK_DESCRIPTOR;
```

17) Page 232, Section 7.4.3, UpdateCapsule(), Description. Replace the next to the last (third) paragraph of section 7.4.3 Description to read as follows:

A capsule which has the **CAPSULE_FLAGS_POPULATE_SYSTEM_TABLE** *Flag* must have **CAPSULE_FLAGS_PERSIST_ACROSS_RESET** set in its header as well. Firmware that processes a capsule that has the **CAPSULE_FLAGS_POPULATE_SYSTEM_TABLE** *Flag* set in its header will coalesce the contents of the capsule from the *ScatterGatherList* into a contiguous buffer and must then place a pointer to this coalesced capsule in the EFI System Table after the system has been reset. Agents searching for this capsule will look in the **EFI_CONFIGURATION_TABLE** and search for the capsule's GUID and associated pointer to retrieve the data after the reset.

Table (#) Flag Firmware Behavior

Flags	Firmware Behavior
No Specification defined flags	Firmware attempts to immediately processes or launch the capsule. If capsule is not recognized, can expect an error.
CAPSULE_FLAGS_PERSIST_ACROSS_RESET	Firmware will attempt to process or launch the capsule across a reset. If capsule is not recognized, can expect an error. If the processing requires a reset which is unsupported by the platform, expect an error.
CAPSULE_FLAGS_PERSIST_ACROSS_RESET + CAPSULE_FLAGS_POPULATE_SYSTEM_TABLE	Firmware will coalesce the capsule from the ScatterGatherList into a contiguous buffer and place a pointer to the coalesced capsule in the EFI System Table. Platform recognition of the capsule type is not required. If the action requires a reset which is unsupported by the platform, expect an error.

The EFI System Table entry must use the GUID from the *CapsuleGuid* field of the **EFI_CAPSULE_HEADER**. The EFI System Table entry must point to an array of capsules that contain the same *CapsuleGuid* value. The array must be prefixed by a **UINT32** that represents the size of the array of capsules.

18) Page 234, Section 7.4.3.

In the UpdateCapsule API Description, the last paragraph before Status Codes Returned should read as follows:

The set of capsules is pointed to by *ScatterGatherList* and *CapsuleHeaderArray* so the firmware will know both the physical and virtual addresses of the operating system allocated buffers. The scatter-gather list supports the situation where the virtual address range of a capsules is contiguous, but the physical address are not. See 6.1.1 for more complete definition of capsule construction.

If any of the capsules that are passed into this function encounter an error, the entire set of capsules will not be processed and the error encountered will be returned to the caller.

19) Page 234, Section 7.4.3.

In the UpdateCapsule Description, the Status Codes Returned table should read as follows.

Status Codes Returned

EFI_SUCCESS	Valid capsule was passed. I Valid capsule was passed. If CAPSULE_FLAGS_PERSIT_ACROSS_RESET is not set, the capsule has been successfully processed by the firmware.
EFI_INVALID_PARAMETER	*CapsuleImageSize* or *HeaderSize* is **NULL**.
EFI_INVALID_PARAMETER	*CapsuleCount* is 0.
EFI_DEVICE_ERROR	The capsule update was started, but failed due to a device error.
EFI_UNSUPPORTED	The capsule type is not supported on this platform.
EFI_OUT_OF_RESOURCES	There were insufficient resources to process the capsule.

20) Page 235, Section 7.4.3.

Delete the QueryCapsuleCapabilities Description third paragraph (shown here with strikethrough text to emphasize deletion):

~~The firmware must support any capsule that has the **CAPSULE_FLAGS_PERSIST_ACROSS_RESET** flag set in **EFI_CAPSULE_HEADER**. The firmware sets the policy for what capsules are supported that do not have the~~

~~**CAPSULE_FLAGS_PERSIST_ACROSS_RESET** flag set.~~

21) Page 235, Section 7.4.3.1.

In QueryCapsuleCapabilities the Prototype description for MaxiumCapsuleSize should read as follows:

MaximumCapsuleSize	On output the maximum size in bytes that **UpdateCapsule()** can support as an argument to **UpdateCapsule()** via *CapsuleHeaderArray* and ScatterGatherList. Undefined on input.

22) Page 238, Section 7.4.3.

In the QueryCapsuleCapabilities Description, the Status Codes Returned table should read as follows.

Status Codes Returned

EFI_SUCCESS	Valid answer returned.
EFI_INVALID_PARAMETER	MaximumCapsuleSize is NULL.
EFI_UNSUPPORTED	The capsule type is not supported on this platform, and MaximumCapsuleSize and ResetType are undefined.
EFI_OUT_OF_RESOURCES	There were insufficient resources to process the query request.

23) Page 261, Section 9.3.5.17.2. The first sentence of this section should read as follows:

Second Byte (At offset 41 into the structure). Valid only if bits 0-3 of More Information in Byte 40 have a value of 2:

24) Page 263, Section 9.3.5.18.

Change Table 60 to read as follows:

Table 60. iSCSI Device Path Node (Base Information)

Mnemonic	Byte Offset	Byte Length	Description
Type	0	1	Type 3 – Messaging Device Path
Sub-Type	1	1	Sub-Type 19 – iSCSI
Length	2	2	Length of this structure in bytes. Length is (18 + n) Bytes
Protocol	4	2	Network Protocol (0 = TCP, 1+ = reserved)
Options	6	2	iSCSI Login Options
Logical Unit Number	8	8	SCSI Logical Unit Number
Target Portal group tag	16	2	iSCSI Target Portal group tag the initiator intends to establish a session with.
iSCSI Target Name	18	n	iSCSI NodeTarget Name. The length of the name is determined by subtracting the offset of this field from *Length*.

25) Page 277, Section 9.5.1.6.

In Table 70, the Type 1, SubType 3 row for **`MemoryMapped`** and the Type 1, SubType 4 row for **`VenH`** should read as follows:

Type: 1 (Hardware Device Path) SubType: 3 (Memory Mapped)	**`MemoryMapped`**(*EfiMemoryType*,*`StartingAddress, EndingAddress`*) The *EfiMemoryType* is a 32-bit integer and is required. The *StartingAddress* and *EndingAddress* are both 64-bit integers and are both required.
Type: 1 (Hardware Device Path) SubType: 4 (Vendor)	**`VenHw`**(*`Guid, Data`*) The *Guid* is a GUID and is required. The *Data* is a Hex Dump and is optional. The default value is zero bytes.

26) Page 279, Section 9.5.1.6.

In Table 70, the Type 2, SubType2 row for **AcpiEx** should read as follows:

Type: 2 (ACPI Device Path) SubType: 2 (ACPI Expanded Device Path)	**AcpiEx(HID,CID,UID,HIDSTR,CIDSTR,UIDSTR)** **AcpiEx(HID|HIDSTR,UID|UIDSTR,CID|CIDSTR)** (Display Only) The *HID* parameter is an EISAID. The default value is 0. Either *HID* or *HIDSTR* must be present. The *CID* parameter is an EISAID. The default value is 0. Either *CID* must be 0 or *CIDSTR* must be empty. The *UID* parameter is an integer. The default value is 0. Either *UID* must be 0 or *UIDSTR* must be empty. The *HIDSTR* is a string. The default value is the empty string. Either *HID* or *HIDSTR* must be present. The *CIDSTR* is a string. The default value is an empty string. Either *CID* must be 0 or *CIDSTR* must be empty. The *UIDSTR* is a string. The default value is an empty string. Either *UID* must be 0 or *UIDSTR* must be empty.

27) Page 280, Section 9.5.1.6.

In Table 70, the Type 3, SubType 9 row for **Infiniband** should read as follows:

Type: 3 (Messaging Device Path) SubType: 9 (Infiniband)	**Infiniband** (*Flags, Guid, ServiceId, TargetId, DeviceId*) *Flags* is an integer. *Guid* is a guid. *ServiceId, TargetId* and *DeviceId* are 64-bit unsigned integers. All fields are required.

28) Page 277, Table 70.

Text for PCI, second column should be:

Pci (*Device*, *Function*)

The *Device* is an integer from 0-31 and is required.

The *Function* is an integer from 0-7 and is required.

29) Page 283, Section 9.5.1.6.

In Table 70, the Type 3, SubType11 row for **MAC** should read as follows:

Type: 3 (Messaging Device Path) SubType: 11 (MAC Address)	**MAC**(*MacAddr, IfType*) The *MacAddr* is a Hex Dump and is required. If *IfType* is 0 or 1, then the *MacAddr* must be exactly six bytes. The *IfType* is an integer from 0-255 and is optional. The default is zero.

30) Page 283, Section 9.5.1.6.

In Table 70, the Type 3, SubType15, Class 1 row for **UsbAudio** should read as follows:

Type: 3 (Messaging Device Path) SubType: 15 (USB Class) Class 1	**UsbAudio**(*VID,PID,SubClass,Protocol*) The *VID* is an integer between 0 and 65535 and is optional. The default value is 0xFFFF. The *PID* is an integer between 0 and 65535 and is optional. The default value is 0xFFFF. The *SubClass* is an integer between 0 and 255 and is optional. The default value is 0xFF. The *Protocol* is an integer between 0 and 255 and is optional. The default value is 0xFF.

31) Page 286, Section 9.5.1.6.

In Table 70, the Type 3, SubType15, Class 7 row for **UsbPrinter** should read as follows:

Type: 3 (Messaging Device Path) SubType: 15 (USB Class) Class 7	**UsbPrinter**(*VID,PID,SubClass,Protocol*) The *VID* is an integer between 0 and 65535 and is optional. The default value is 0xFFFF. The *PID* is an integer between 0 and 65535 and is optional. The default value is 0xFFFF. The *SubClass* is an integer between 0 and 255 and is optional. The default value is 0xFF. The *Protocol* is an integer between 0 and 255 and is optional. The default value is 0xFF.

32) Page 287, Section 9.5.1.6,.

In Table 70, the Type 3, SubType15, Class 11 row for **`UsbSmartCard`** should read as follows:

Type: 3 (Messaging Device Path) SubType: 15 (USB Class)	**`UsbSmartCard`**(*`VID,PID,SubClass,Protocol`*)
Class 11	The *VID* is an integer between 0 and 65535 and is optional. The default value is 0xFFFF. The *PID* is an integer between 0 and 65535 and is optional. The default value is 0xFFFF. The *SubClass* is an integer between 0 and 255 and is optional. The default value is 0xFF. The *Protocol* is an integer between 0 and 255 and is optional. The default value is 0xFF.

33) Page 288, Section 9.5.1.6,.

In Table 70, the Type 3, SubType15, Class 254, SubClass 1 row for **`UsbDeviceFirmwareUpdate`** should read as follows:

Type: 3 (Messaging Device Path) SubType: 15 (USB Class)	**`UsbDeviceFirmwareUpdate`**(*`VID,PID,Protocol`*)
Class 254 SubClass: 1	The *VID* is an integer between 0 and 65535 and is optional. The default value is 0xFFFF. The *PID* is an integer between 0 and 65535 and is optional. The default value is 0xFFFF. The *Protocol* is an integer between 0 and 255 and is optional. The default value is 0xFF.

34) Page 289, Section 9.5.2.

EFI_DEVICE_PATH_UTILITIES_PROTOCOL GUID and Protocol Interface Structure should read as follows:

GUID

```
#define EFI_DEVICE_PATH_UTILITIES_PROTOCOL_GUID \
                  {0x0379be4e, 0xd706, 0x437d, \
                  0xb0, 0x37, 0xed, 0xb8, 0x2f, 0xb7, 0x72, 0xa4 }
```

Protocol Interface Structure

```
typedef struct _EFI_DEVICE_PATH_UTILITIES_PROTOCOL {
  EFI_DEVICE_PATH_UTILS_GET_DEVICE_PATH_SIZE   GetDevicePathSize;
  EFI_DEVICE_PATH_UTILS_DUP_DEVICE_PATH        DuplicateDevicePath;
  EFI_DEVICE_PATH_UTILS_APPEND_PATH            AppendDevicePath;
  EFI_DEVICE_PATH_UTILS_APPEND_NODE            AppendDeviceNode;
  EFI_DEVICE_PATH_UTILS_APPEND_INSTANCE        AppendDevicePathInstance;
  EFI_DEVICE_PATH_UTILS_GET_NEXT_INSTANCE      GetNextDevicePathInstance;
  EFI_DEVICE_PATH_UTILS_IS_MULTI_INSTANCE      IsDevicePathMultiInstance;
  EFI_DEVICE_PATH_UTILS_CREATE_NODE            CreateDeviceNode;
} EFI_DEVICE_PATH_UTILITIES_PROTOCOL;
```

35) Page 290, Section 9.5.1.6.

In Table 70, the Type 4 row for **MediaPath** and Type 4 , SubType1 row for **HD** should read as follows:

Type: 4	**MediaPath(subtype, data)** The *subtype* is an integer from 0-255 and is required. The *data* is a hex dump.
Type: 4 (Media Device Path) SubType: 1 (Hard Drive)	**HD(Partition,Type,Signature,Start, Size)** **HD(Partition,Type,Signature) (Display Only)** The *Partition* is an integer representing the partition number. It is optional and the default is 0. If *Partition* is 0, then *Start* and *Size* are prohibited. The *Type* is an integer between 0-255 or else the keyword **MBR** (1) or **GPT** (2). The type is optional and the default is 2. The *Signature* is an integer if *Type* is 1 or else GUID if *Type* is 2. The signature is required. The *Start* is a 64-bit unsigned integer. It is prohibited if *Partition* is 0. Otherwise it is required. The *Size* is a 64-bit unsigned integer. It is prohibited if *Partition* is 0. Otherwise it is required.

36) Page 291, Section 9.5.2.

EFI_DEVICE_PATH_UTILITIES.GetDevicePathSize Prototype, Parameters and Description should read as follows:

Prototype

```
typedef
UINTN
(EFIAPI *EFI_DEVICE_PATH_GET_DEVICE_PATH_SIZE) (
  IN CONST EFI_DEVICE_PATH*       DevicePath
  );
```

Parameters

DevicePath

Points to the start of the EFI device path (or **NULL**).

Description

This function returns the size of the specified device path, in bytes, including the end-of-path tag. If *DevicePath* is **NULL** then zero is returned.

37) Page 292Section 9.5.2.

EFI_DEVICE_PATH_UTILITIES_PROTOCOL.DuplicateDevicePath Prototype, Parameters and Description should read as follows:

Parameters

DevicePath

Points to the source device path or **NULL**.

Description

This function creates a duplicate of the specified device path. The memory is allocated from EFI boot services memory. It is the responsibility of the caller to free the memory allocated. If *DevicePath* is **NULL** then **NULL** will be returned and no memory will be allocated.

38)

Page 292, Section 9.5.2 (EFI_DEVICE_PATH_UTILITIES_PROTOCOL)

The function prototypes for all functions need to be changed to include _UTILS per the following table:

EFI_DEVICE_PATH_GET_DEVICE_PATH_SIZE	EFI_DEVICE_PATH_UTILS_GET_DEVICE_PATH_SIZE
EFI_DEVICE_PATH_DUP_DEVICE_PATH	EFI_DEVICE_PATH_UTILS_DUP_DEVICE_PATH
EFI_DEVICE_PATH_APPEND_DEVICE_PATH	EFI_DEVICE_PATH_UTILS_APPEND_DEVICE_PATH
EFI_DEVICE_PATH_APPEND_DEVICE_NODE	EFI_DEVICE_PATH_UTILS_APPEND_DEVICE_NODE
EFI_DEVICE_PATH_APPEND_DEVICE_PATH_INSTANCE	EFI_DEVICE_PATH_UTILS_APPEND_DEVICE_PATH_INS
EFI_DEVICE_PATH_GET_NEXT_INSTANCE	EFI_DEVICE_PATH_UTILS_GET_NEXT_INSTANCE
EFI_DEVICE_PATH_CREATE_NODE	EFI_DEVICE_PATH_UTILS_CREATE_NODE

EFI_DEVICE_PATH_IS_MULTI_INSTANCE	EFI_DEVICE_PATH_UTILS_IS_MULTI_INSTANCE

39) Page 292, Section 9.5.2.

EFI_DEVICE_PATH_UTILITIES_PROTOCOL.DuplicateDevicePath Prototype should read as follows :

Prototype

```
typedef
EFI_DEVICE_PATH*
(EFIAPI *EFI_DEVICE_PATH_DUP_DEVICE_PATH) (
  IN CONST EFI_DEVICE_PATH*       DevicePath
  );
```

40) Page 293Section 9.5.2.

AppendDevicePath paramenters, etc., should read as follows:

Parameters

Src1 Points to the first device path.

Src2 Points to the second device path.

Description

This function creates a new device path by appending a copy of the second device path to a copy of the first device path in a newly allocated buffer. Only the end-of-device-path device node from the second device path is retained. If *Src1* is **NULL** and *Src2* is non-**NULL**, then a duplicate of *Src2* is returned. If *Src1* is non-**NULL** and *Src2* is **NULL,** then a duplicate of *Src1* is returned. If *Src1* and *Src2* are both **NULL**, then a copy of an end-of-device-path is returned.

The memory is allocated from EFI boot services memory. It is the responsibility of the caller to free the memory allocated.

Returns

This function returns a pointer to the newly created device path or NULL if memory could not be allocated.

41) Page 293, Section 9.5.2.

EFI_DEVICE_PATH_UTILITIES_PROTOCOL.AppendDevicePath Prototype should read as follows :

Prototype

```
typedef
EFI_DEVICE_PATH*
(EFIAPI *EFI_DEVICE_PATH_APPEND_DEVICE_PATH)
  IN CONST EFI_DEVICE_PATH* Src1,
  IN CONST EFI_DEVICE_PATH* Src2
  );
```

42) Page 29, 4Section 9.5.2.

AppendDeviceNode paramenters, etc., should read as follows:

Parameters

DevicePath Points to the device path.

DeviceNode Points to the device node.

Description

This function creates a new device path by appending a copy of the specified device node to a copy of the specified device path in an allocated buffer. The end-of-device-path device node is moved after the end of the appended device node. If *DeviceNode* is NULL then a copy of *DevicePath* is returned. If *DevicePath* is **NULL** then a copy of *DeviceNode*, followed by an end-of-device path device node is returned. If both *DeviceNode* and *DevicePath* are **NULL** then a copy of an end-of-device-path device node is returned.

The memory is allocated from EFI boot services memory. It is the responsibility of the caller to free the memory allocated.

Returns

This function returns a pointer to the allocated device path or NULL if there was insufficient memory.

43) Page 297, Section 9.5.2.

EFI_DEVICE_PATH_UTILITIES_PROTOCOL.CreateDeviceNode Prototype should read as follows :

Prototype

```
typedef
EFI_DEVICE_PATH*
(EFIAPI *EFI_DEVICE_PATH_CREATE_NODE) (
  IN UINT8 NodeType,
  IN UINT8 NodeSubType,
  IN UINT16 NodeLength
  );
```

44) Page 296, Section 9.5.2.

EFI_DEVICE_PATH_UTILITIES_PROTOCOL.GetNextDevicePathInstance Prototype and Paramenters should read as follows :

Prototype

```
typedef
EFI_DEVICE_PATH_PROTOCOL*
(EFIAPI *EFI_DEVICE_PATH_GET_NEXT_INSTANCE) (
  IN OUT EFI_DEVICE_PATH_PROTOCOL **DevicePathInstance,
  OUT UINTN                       *DevicePathInstanceSize OPTIONAL
  );
```

Parameters

DevicePathInstance

On input, this holds the pointer to the current device path instance. On output, this holds the pointer to the next device path instance or **NULL** if there are no more device path instances in the device path.

DevicePathInstanceSize

On output, this holds the size of the device path instance, in bytes or zero, if *DevicePathInstance* is **NULL**. If **NULL**, then the instance size is not output.

45) Page 339, Section 10.4, EFI Driver Configuration Protocol EFI_DRIVER_CONFIGURATION_PROTOCOL. Replace the Protocol Interface structure with the following:

```
typedef struct _EFI_DRIVER_CONFIGURATION2_PROTOCOL {
        EFI_DRIVER_CONFIGURATION_SET_OPTIONS          SetOptions;
        EFI_DRIVER_CONFIGURATION_OPTIONS_VALID        OptionsValid;
        EFI_DRIVER_CONFIGURATION_FORCE_DEFAULTS       ForceDefaults;
        CHAR8                                         *SupportedLanguages;
} EFI_DRIVER_CONFIGURATION2_PROTOCOL;
```

46) Page 339and following, Section 10.4:.
Change all references to EFI_DRIVER_CONFIGURATION_PRTOCOL to EFI_DRIVER_CONFIGURATION2_PROTOCOL, including all EFI_DRIVER_CONFIGURATION_PROTOCOL function names.

47) Page 349; Section 10.5 EFI Driver Diagnostoics Protocol,EFI_DRIVER_DIAGNOSTICS_PROTOCOL. Replaces the Protocol Interface Structure with the following:

```
typedef struct _EFI_DRIVER_DIAGNOSTICS2_PROTOCOL {
        EFI_DRIVER_DIAGNOSTICS_RUN_DIAGNOSTICS        RunDiagnostics;
          CHAR8                                       *SupportedLanguages;
            } EFI_DRIVER_DIAGNOSTICS2_PROTOCOL;
```

48)) Page 349, ;Section 10.5,and following. **UEFI_CAPSULE_BLOCK_DESCRIPTOR**
Change all references to EFI_DRIVER_DIAGNOSTICS_PROTOCOL to EFI_DRIVER_DIAGNOSTICS2_PROTOCOL, including all EFI_DRIVER_DIAGNOSTICS_PROTOCOL function names.

49) Page 352, Section 10.5.
To the Status Codes Returned, add a return code to the EFI_DRIVER_DIAGNOSTICS_PROTOCOL.RunDiagnostics() function. Add The following return code between the first return code (EFI_SUCCESS) and second return code EFI_VALID _PARAMETER):

EFI_ACCESS_DENIED	The request for initiating diagnostics was unable to be completed due to some underlying hardware or software state.

50) Pages 353 and following, Section 10.6. Replace all of the section 10.6 content with the following content:

EFI Component Name Protocol

This section provides a detailed description of the **EFI_COMPONENT_NAME2_PROTOCOL**. This is a protocol that allows an driver to provide a user readable name of a UEFI Driver, and a user readable name for each of the controllers that the driver is managing. This protocol is used by platform management utilities that wish to display names of components. These names may include the names of expansion slots, external connectors, embedded devices, and add-in devices.

EFI_COMPONENT_NAME2_PROTOCOL

Summary

Used to retrieve user readable names of drivers and controllers managed by UEFI Drivers.

GUID

```
#define EFI_COMPONENT_NAME2_PROTOCOL_GUID \
                    {0x6a7a5cff, 0xe8d9, 0x4f70, 0xba, 0xda, 0x75, 0xab, 0x30,
      0x25, 0xce, 0x14}
```

Protocol Interface Structure

```
typedef struct _EFI_COMPONENT_NAME2_PROTOCOL {
  EFI_COMPONENT_NAME_GET_DRIVER_NAME       GetDriverName;
  EFI_COMPONENT_NAME_GET_CONTROLLER_NAME   GetControllerName;
  CHAR8                                    *SupportedLanguages;
} EFI_COMPONENT_NAME2_PROTOCOL;
```

Parameters

GetDriverName Retrieves a Unicode string that is the user readable name of the driver. See the **GetDriverName()** function description.

GetControllerName Retrieves a Unicode string that is the user readable name of a controller that is being managed by a driver. See the **GetControllerName()** function description.

SupportedLanguages A Null-terminated ASCII string array that contains one or more supported language codes. This is the list of language codes that this protocol supports. The number of languages supported by a driver is up to the driver writer.

SupportedLanguages is specified in RFC 4646 format. See Appendix M for the format of language codes and language code arrays.

Description

The **EFI_COMPONENT_NAME2_PROTOCOL** is used retrieve a driver's user readable name and the names of all the controllers that a driver is managing from the driver's point of view. Each of these names is returned as a Null-terminated Unicode string. The caller is required to specify the language in which the Unicode string is returned, and this language must be present in the list of languages that this protocol supports specified by *SupportedLanguages*.

EFI_COMPONENT_NAME2_PROTOCOL.GetDriverName()

Summary

Retrieves a Unicode string that is the user readable name of the driver.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_COMPONENT_NAME_GET_DRIVER_NAME) (
  IN  EFI_COMPONENT_NAME2_PROTOCOL *This,
  IN  CHAR8                        *Language,
  OUT CHAR16                       **DriverName
  );
```

Parameters

This A pointer to the EFI_COMPONENT_NAME2_PROTOCOL instance.

Language A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. *Language* is specified in RFC 4646 language code format. See Appendix M for the format of language codes.

DriverName A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by *This* in the language specified by *Language*.

Description

This function retrieves the user readable name of a driver in the form of a Unicode string. If the driver specified by *This* has a user readable name in the language specified by *Language*, then a pointer to the driver name is returned in *DriverName*, and **EFI_SUCCESS** is returned. If the driver specified by *This* does not support the language specified by *Language*, then **EFI_UNSUPPORTED** is returned.

Status Codes Returned

EFI_SUCCESS	The Unicode string for the user readable name in the language specified by *Language* for the driver specified by *This* was returned in *DriverName*.
EFI_INVALID_PARAMETER	*Language* is **NULL**.
EFI_INVALID_PARAMETER	*DriverName* is **NULL**.
EFI_UNSUPPORTED	The driver specified by *This* does not support the language specified by *Language*.

EFI_COMPONENT_NAME2_PROTOCOL.GetControllerName()

Summary

Retrieves a Unicode string that is the user readable name of the controller that is being managed by a driver.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_COMPONENT_NAME_GET_CONTROLLER_NAME) (
  IN  EFI_COMPONENT_NAME2_PROTOCOL    *This,
  IN  EFI_HANDLE                      ControllerHandle,
  IN  EFI_HANDLE                      ChildHandle        OPTIONAL,
  IN  CHAR8                           *Language,
  OUT CHAR16                          **ControllerName
  );
```

Parameters

This A pointer to the EFI_COMPONENT_NAME2_PROTOCOL instance.

ControllerHandle The handle of a controller that the driver specified by *This* is managing. This handle specifies the controller whose name is to be returned.

ChildHandle The handle of the child controller to retrieve the name of. This is an optional parameter that may be **NULL**. It will be **NULL** for device drivers. It will also be **NULL** for bus drivers that attempt to retrieve the name of the bus controller. It will not be **NULL** for a bus driver that attempts to retrieve the name of a child controller.

Language A pointer to a Null- terminated ASCII string array indicating the language. This is the language of the controller name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. *Language* is specified in RFC 4646 language code format. See Appendix M for the format of language codes.

ControllerName A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by *ControllerHandle* and *ChildHandle* in the language specified by *Language* from the point of view of the driver specified by *This*.

Description

This function retrieves the user readable name of the controller specified by *ControllerHandle* and *ChildHandle* in the form of a Unicode string. If the driver specified by *This* has a user readable name in the language specified by *Language*, then a pointer to the controller name is returned in *ControllerName*, and **EFI_SUCCESS** is returned.

If the driver specified by *This* is not currently managing the controller specified by *ControllerHandle* and *ChildHandle*, then **EFI_UNSUPPORTED** is returned.

If the driver specified by *This* does not support the language specified by *Language*, then **EFI_UNSUPPORTED** is returned.

Status Codes Returned

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*.

51) Page 358, Section 10.7.

Change the Description to read as follows:

> The **EFI_SERVICE_BINDING_PROTOCOL** provides member functions to create and destroy child handles. A driver is responsible for adding protocols to the child handle in **CreateChild**() and removing protocols in **DestroyChild**(). It is also required that the **CreateChild ()** function opens the parent protocol **BY_CHILD_CONTROLLER** to establish parent-child relationship, and closes the protocol in **DestoryChild ()**. The pseudo code for **CreateChild()** and **DestoryChild** () is provided to specify the required behavior, not the required implementation. Each consumer of a software protocol is responsible for calling **CreateChild()** when it requires the protocol and calling **DestroyChild()** when it is finished with that protocol.

52) Page 415, Section 11.7.1.

The EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE_INFORMATION structure has a member which has one too many "*"'s in it. This is an unnecessary level of indirection for the member. The structure code of EFI_GRAPHICS_OUTPUT_PROTOCOL, Related Definitions on page 415 should read as follows:

```
typedef struct {
 UINT32                                 MaxMode;
 UINT32                                 Mode;
 EFI_GRAPHICS_OUTPUT_MODE_INFORMATION   *Info;
 UINTN                                  SizeOfInfo;
 EFI_PHYSICAL_ADDRESS                   FrameBufferBase;
 UINTN                                  FrameBufferSize;
```

```
} EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE;
```

53) Page 417, Section 11.7.1.

The EFI_GRAPHICS_OUTPUT_PROTOCOL_QUERY_MODE function has a function parameters which has too few "*"'s in it. This makes the function unimplementable as currently defined since it is intended as a callee allocated field. The EFI_GRAPHICS_OUTPUT_PROTOCOL.QueryMode() Prototype on page 417 should read as follows:

Prototype

```
typedef

EFI_STATUS

(EFIAPI *EFI_GRAPHICS_OUTPUT_PROTOCOL_QUERY_MODE) (

  IN  EFI_GRAPHICS_OUTPUT_PROTOCOL                *This,

  IN  UINT32                                       ModeNumber,

  OUT UINTN                                        *SizeOfInfo

  OUT EFI_GRAPHICS_OUTPUT_MODE_INFORMATION        **Info

   );
```

54) Page 424, Section 11.7.1.

The EFI_EDID_DISCOVERED_PROTOCOL has a field which needs to be constructed with a pointer since it is intended to be a "pointer to an array of bytes that contains the EDID information". The EFI_EDID_DISCOVERED_PROTOCOL, Protocol Interface Structure should read as follows:

Protocol Interface Structure

```
typedef struct {

  UINT32   SizeOfEdid;

  UINT8    *Edid;

} EFI_EDID_DISCOVERED_PROTOCOL;
```

55) Page 424, and page 425;EFI_EDID_DISCOVERED_PROTOCOL**,** , EFI_EDID_ACTIVE_PROTOCOL, repectively,. The last sentence of the Edid parameter should read as follows:

EDID information is defined in the E-EDID EEPROM specification published by VESA (www.vesa.org).

56) Page 430, Section 11.8. One statement is a vestige from its previous UGA inheritance and should not necessarily be a requirement today. Strike the following statement from the specification.:

A plug in graphics device that contains a ROM must have an EBC version of the EFI driver that produces the **EFI_GRAPHICS_OUTPUT_PROTOCOL**.

57) Page 434,EFI_SIMPLE_FILE_SYSTEM_PROTOCOL.OpenVolume(), Prototype. Replace the first parameter line (fourth line) with the following:

```
IN     EFI_SIMPLE_FILE_SYSTEM_PROTOCOL   *This
```

58) Page 492, Section 12.8, EFI_UNICODE_COLLATION_PROTOCOL. Update the EFI_UNICODE_COLLATION_PROTOCOL_GUID with the following:

```
#define EFI_UNICODE_COLLATION_PROTOCOL2_GUID \
{ 0xa4c751fc, 0x23ae, 0x4c3e, 0x92, 0xe9, 0x49, 0x64, 0xcf, 0x63, 0xf3, 0x49
```

59) Page 619, Section 14.5.5, Description.

Remove a reference to a return code that isn't valid for this particular function. The second to the last paragraph on the page should read as follows:

If **EFI_SUCCESS**, **EFI_BAD_BUFFER_SIZE**, **EFI_DEVICE_ERROR**, or **EFI_TIMEOUT** is returned, then the caller must examine the status fields in *Packet* in the following precedence order: *HostAdapterStatus* followed by *TargetStatus* followed by *SenseDataLength*, followed by *SenseData*.

60) Page 619, Section 14.5.5.

Fix references to status codes that were inconsistent within the SCSI I/O ExecuteScsiCommand API. **EFI_SCSI_IO_PROTOCOL.ExecuteScsiCommand()** paragraphs second and fourth from the bottom should be changed to read as follows:

If the data buffer described by *DataBuffer* and *TransferLength* is too big to be transferred in a single command, then **EFI_BAD_BUFFER_SIZE** is returned. The number of bytes actually transferred is returned in *TransferLength*.

…

If **EFI_SUCCESS, EFI_BAD_BUFFER_SIZE**, **EFI_DEVICE_ERROR,** or **EFI_TIMEOUT** is returned, then the caller must examine the status fields in *Packet* in the following precedence order: *HostAdapterStatus* followed by *TargetStatus* followed by *SenseDataLength*, followed by *SenseData*. If non-blocking I/O is being used, then the status fields in *Packet* will not be valid until the *Event* associated with *Packet* is signaled.

61) Page 620, Section 14.5.5. Further correction to inconsistent status codes. Append to the end of Status Codes Returned, EFI_BAD_BUFFER_SIZE fields as follows:

EFI_BAD_BUFFER_SIZE	The SCSI Request Packet was not executed. For read and bi-directional commands, the number of bytes that could be transferred is returned in *InTransferLength.* For write and bi-directional commands, the number of bytes that could be transferred is returned in *OutTransferLength.* See *HostAdapterStatus* and *TargetStatus* in that order for additional status information.

62) Page 628, Section 14.8, EFI_EXT_SCSI_PASS_THRU_PROTOCOL. Update the EFI_EXT_SCSI_PASS_THRU_PROTOCOL_GUID with the following:

```
#define EFI_EXT_SCSI_PASS_THRU_PROTOCOL_GUID \
{0x143b7632, 0xb81b, 0x4cb7, 0xab, 0xd3, 0xb6, 0x25, 0xa5, 0xb9, 0xbf, 0xfe}
```

63) Pages 633 and 636 Section 14.8.

In function **EFI_EXT_SCSI_PASS_THRU_PROTOCOL.PassThru()**, in the Related Definitions for **EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET**, the definitions for parameters *InDataBuffer*, *OutDataBuffer*, and *SenseBuffer* should change to read as follows:

InDataBuffer	A pointer to the data buffer to transfer between the SCSI controller and the SCSI device for read and bidirectional commands. For all write and non data commands where *InTransferLength* is 0, this field is optional and may be **NULL**. If this field is not **NULL**, then it must be aligned on the boundary specified by the *IoAlign* field in the **EFI_EXT_SCSI_PASS_THRU_MODE** structure.
OutDataBuffer	A pointer to the data buffer to transfer between the SCSI controller and the SCSI device for write or bidirectional commands. For all read and non data commands where *OutTransferLength* is 0, this field is optional and may be **NULL**. If this field is not **NULL**, then it must be aligned on the boundary specified by the *IoAlign* field in the **EFI_EXT_SCSI_PASS_THRU_MODE** structure.
SenseData	A pointer to the sense data that was generated by the execution of the SCSI Request Packet. If *SenseDataLength* is 0, then this field is optional and may be **NULL**. It is strongly recommended that a sense data buffer of at least 252 bytes be provided to guarantee the entire sense data buffer generated from the execution of the SCSI Request Packet can be returned. If this field is not **NULL**, then it must be aligned to the boundary specified in the *IoAlign* field in the **EFI_EXT_SCSI_PASS_THRU_MODE** structure.

Also, the following notes are added at the end of the description for **EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET**:

Note: *: Some examples of SCSI read commands are READ, INQUIRY, and MODE_SENSE.*

Note: *Some examples of SCSI write commands are WRITE and MODE_SELECT.*

Note: *An example of a SCSI non data command is TEST_UNIT_READY.*

64) Pages 638, 639, Section 14.8,

Change EFI_EXT_SCSI_PASS_THRU_PROTOCOL.GetNextTargetLun() section to read as follows:

Summary

Used to retrieve the list of legal Target IDs and LUNs for SCSI devices on a SCSI channel. These can either be the list SCSI devices that are actually present on the SCSI channel, or the list of legal Target Ids and LUNs for the SCSI channel. Regardless, the caller of this function must probe the Target ID and LUN returned to see if a SCSI device is actually present at that location on the SCSI channel.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_EXT_SCSI_PASS_THRU_GET_NEXT_TARGET_LUN) (
          IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL   *This,
          IN OUT UINT8                         **Target,
          IN OUT UINT64                        *Lun
);
```

Parameters

This	A pointer to the **EFI_EXT_SCSI_PASS_THRU_PROTOCOL** instance. Type **EFI_EXT_SCSI_PASS_THRU_PROTOCOL** is defined in Section 14.7.
Target	On input, a pointer to a legal Target ID (an array of size **TARGET_MAX_BYTES)** for a SCSI device on the SCSI channel. On output, a pointer to the next legal Target ID (an array of **TARGET_MAX_BYTES**) of a SCSI device on a SCSI channel. An input value of **0xFF**'s (all bytes in the array are **0xFF**) in the Target array retrieves the first legal Target ID for a SCSI device ID on a SCSI channel.
Lun	On input, a pointer to the LUN of a SCSI device present on the SCSI channel. On output, a pointer to the LUN of the next SCSI device present on a SCSI channel.

Description

The **EFI_EXT_SCSI_PASS_THRU_PROTOCOL.GetNextTargetLun()** function retrieves A list of legal Target ID and LUN for a SCSI channel. If on input a *Target* is

specified by all **0xFF** in the Target array, then the first legal Target ID and LUN for a SCSI device on a SCSI channel is returned in *Target* and *Lun* and **EFI_SUCCESS** is returned.

If *Target* and *Lun* is a Target ID and LUN value that was returned on a previous call to **GetNextTargetLun()**, then the next legal Target ID and LUN for a SCSI device on the SCSI channel is returned in *Target* and *Lun*, and **EFI_SUCCESS** is returned.

If *Target array* is not all **0xF's** and *Target* and *Lun* were not returned on a previous call to **GetNextTargetLun()**, then **EFI_INVALID_PARAMETER** is returned.

If *Target* and *Lun* are the Target ID and LUN of the last SCSI device on the SCSI channel, then **EFI_NOT_FOUND** is returned.

Status Codes Returned

EFI_SUCCESS	The Target ID and LUN of the next SCSI device on the SCSI channel was returned in *Target* and *Lun*.
EFI_NOT_FOUND	There are no more SCSI devices on this SCSI channel.
EFI_INVALID_PARAMETER	*Target array* is not all **0xFF**'s, and *Target* and *Lun* were not returned on a previous call to **GetNextTargetLun()**.

65) Page 650, Section 15.2,.

The protocol GUID value should be 16 bytes long instead of 15 bytes long for ISCSI Initiator Name Protocol. The correct ISCSI Initiator Name Protocol GUID should read as follows:

```
#define EFI_ISCSI_INITIATOR_NAME_PROTOCOL_GUID
{ \
 0x59324945, 0xec44, 0x4c0d, 0xb1, 0xcd, 0x9d, 0xb1, 0x39, 0xdf,
0x7, 0xc \
}
```

66) Pages 678 and 681 Section 16.1, and.

Add the status code (given below the functions) to the Status Codes Returned tables for the following functions in section 16.1:

EFI_USB2_HC_PROTOCOL.IsochronousTransfer()
EFI_USB2_HC_PROTOCOL.AsyncIsochronousTransfer()

EFI_UNSUPPORTED	The implementation doesn't support Isochronous transfer function

67) Page 685, EFI_USB2_HC_PROTOCOL..GetRootHubPortStatus(), Description, second paragraph should read as follows:

EFI_USB_PORT_STATUS describes the port status of a specified USB port based on the reporting capabilities of that particular port's host controller. This data structure is designed to be common to both a USB root hub port and a USB hub port.

68) Page 684, EFI_USB2_HC.GetRootHubPortStatus(),Table 106. Replace the last row with two rows reading as follows:

11	Release port ownership to companion host controller (USB_PORT_STAT_OWNER) 0 = Port ownership has not been transferred 1 = Port ownership has been transferred.
12-15	Reserved These bits return 0 when read.

69) Pages 686, Section 16.1,

In the function EFI_USB2_HC_PROTOCOL.SetRootHubPortFeature(), in the Related Definitions, add the following value to enumerated type **EFI_USB_PORT_FEATURE**:

```
EfiUsbPortOwner                 = 13,
```

70) Page 687, Section 16.1,Table 108. Following the definition of **EFI_USB_PORT_FEATURE**, insert the table row (given below) following the row for EfiUsbPortPower:

EfiUsbPortOwner	N/A	Releases the port ownership of this port to companion host controller.

71) Page 687, EFI_USB2_HC_ PROTOCOL.SetRootHubPortFeature(), Description, second paragraph should read as follows:

The number of root hub ports attached to the USB host controller can be determined with the function **GetRootHubPortStatus().** If *PortNumber* is greater than or equal to the number of ports returned by **GetRootHubPortNumber()**, then **EFI_INVALID_PARAMETER** is returned. If *PortFeature* is not **EfiUsbPortOwner, EfiUsbPortEnable**, **EfiUsbPortSuspend**, **EfiUsbPortPower**, **EfiUsbPortConnectChange**, **EfiUsbPortResetChange**, **EfiUsbPortEnableChange**, **EfiUsbPortSuspendChange**, or **EfiUsbPortOverCurrentChange**, then **EFI_INVALID_PARAMETER** is returned.

72) Page 687, EFI_USB2_HC_PROTOCOL.SetRootHubPortFeature().Add the following row to Status Codes Returned:

EFI_UNSUPPORTED	*PortFeature* is invalid for the given host controller.

73) Section 16.2.4, pages 708 and 710.

Add the following status code (given below the functions) to the Status Codes Returned tables for the following functions:

EFI_USB_IO_PROTOCOL.UsbIsochronousTransfer()

EFI_USB_IO_PROTOCOL.UsbAsyncIsochronousTransfer()

EFI_UNSUPPORTED	The implementation doesn't support Isochronous transfer function

74) Page 873, Section 20.2, EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL. Update the EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL_GUID with the following:

```
#define EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL_GUID_31 \
  { \
    0x1ACED566, 0x76ED, 0x4218, 0xBC, 0x81, 0x76, 0x7F, 0x1F, 0x97, 0x7A, 0x89
\
  }
```

75) Page 1030, Chapter 23.1.

Add an instance handle to the **EFI_TCP4_SERVICE_POINT of EFI_TCP4_VARIABLE_DATA**.

```
//*******************************************
// EFI_TCP4_VARIABLE_DATA
//*******************************************
typedef struct {
  EFI_HANDLE              DriverHandle;
  UINTN                   ServiceCount;
  EFI_TCP4_SERVICE_POINT  Services[1];
} EFI_TCP4_VARIABLE_DATA;
```

DriverHandle	The handle of the driver that creates this entry.
ServiceCount	The number of address/port pairs following this data structure.
Services	List of address/port pairs that are currently in use. Type **EFI_TCP4_SERVICE_POINT** is defined below.

```
//*******************************************
// EFI_TCP4_SERVICE_POINT
//*******************************************
typedef struct{
  EFI_HANDLE          InstanceHandle;
  EFI_IPv4_ADDRESS    LocalAddress;
  UINT16              LocalPort;
  EFI_IPv4_ADDRESS    RemoteAddress;
  UINT16              RemotePort;
} EFI_TCP4_SERVICE_POINT;
```

InstanceHandle	The EFI TCPv4 Protocol instance handle that is using this service port.
LocalAddress	The local IPv4 address to which this TCPv4 protocol instance is bound.
LocalPort	The local port number in host byte order.
RemoteAddress	The remote IPv4 address. It may be 0.0.0.0 if it isn't connected to any remote host.
RemotePort	The remote port number in host byte order. It may be zero if it isn't connected to any remote host

76) Page 1030 and following (listed below)Section 23.1,.

Some data structure members in **EFI_TCP4_PROTOCOL** are defined as **UINTN** such as the **FragmentLength** in the **EFI_TCP4_FRAGMENT_DATA**.. Change all these types to **UINT32**.

On Page 1030:

```
//*******************************************
// EFI_TCP4_VARIABLE_DATA
//*******************************************
typedef struct {
  EFI_HANDLE                DriverHandle;
  UINT32                    ServiceCount;
  EFI_TCP4_ SERVICE_POINT   Services[1];
} EFI_TCP4_VARIABLE_DATA;
```

On Page 1036:

```
typedef struct {
  UINT32            ReceiveBufferSize;
  UINT32            SendBufferSize;
  UINT32            MaxSynBackLog;
  UINT32            ConnectionTimeout;
  UINT32            DataRetries;
  UINT32            FinTimeout;
  UINT32            TimeWaitTimeout;
  UINT32            KeepAliveProbes;
  UINT32            KeepAliveTime;
```

```
  UINT32            KeepAliveInterval;
  BOOLEAN           EnableNagle;
  BOOLEAN           EnableTimeStamp;
  BOOLEAN           EnableWindowScaling;
  BOOLEAN           EnableSelectiveAck;
  BOOLEAN           EnablePathMtuDiscovery;
} EFI_TCP4_OPTION;
```

On Page 1051 Note: The problematic **IN OUT** modifier for the **DataLength** is also removed here:

```
//*******************************************************************
// EFI_TCP4_RECEIVE_DATA
//*******************************************************************
typedef struct {
  BOOLEAN                 UrgentFlag;
  UINT32                  DataLength;
  UINT32                  FragmentCount;
  EFI_TCP4_FRAGMENT_DATA  FragmentTable[1];
} EFI_TCP4_RECEIVE_DATA;


........

//*******************************************************************
// EFI_TCP4_FRAGMENT_DATA
//*******************************************************************
typedef struct {
  UINT32    FragmentLength;
  VOID      *FragmentBuffer;
} EFI_TCP4_FRAGMENT_DATA;
```

On Page 1052:

```
//*******************************************************************
// EFI_TCP4_TRANSMIT_DATA
```

```
//*******************************************************
typedef struct {
  BOOLEAN                  Push;
  BOOLEAN                  Urgent;
  UINT32                   DataLength;
  UINT32                   FragmentCount;
  EFI_TCP4_FRAGMENT_DATA   FragmentTable[1];
} EFI_TCP4_TRANSMIT_DATA;
```

77) Page 1156, Section 25.2.4,.

Make the *bCertificate* **[…]** a comment because in the GUID'd WIN_CERT; the latter structure has an additional **ANYSIZE_ARRAY**. Changes to **WIN_CERTIFICATE** as follows:

```
typedef struct _WIN_CERTIFICATE {
      UINT32       dwLength;
      UINT16       wRevision;
      UINT16       wCertificateType;
      // UINT8     bCertificate[ANYSIZE_ARRAY];
} WIN_CERTIFICATE;
```

78) Page 1157, Section 25.2.4.

The HashType enumeration in the certificate structure was never set. This changes it to an EFI_GUID to match the rest of Chapter 25 content.

Change To Section 25.2.3 (Replace in **WIN_CERTIFICATE_EFI_PKCS1_15**, starting with Prototype)

Prototype

```
typedef struct _WIN_CERTIFICATE_EFI_PKCS1_15 {
 WIN_CERTIFICATE    Hdr;
 EFI_GUID           HashAlgorithm;
//  UINT8           Signature[ANYSIZE_ARRAY];
} WIN_CERTIFICATE_EFI_PKCS1_15;
```

Hdr

This is the standard **WIN_CERTIFICATE** header, where *wCertificateType* is set to **WIN_CERT_TYPE_UEFI_PKCS1_15**.

HashAlgorithm

This is the hashing algorithm which was performed on the UEFI executable when creating the digital signature. It is one of the enumerated pre-defined GUID values defined in section 25.4.1 (see **EFI_HASH_ALGORITHM_x**).

Signature

This is the actual digital signature. The size of the signature is the same size as the key (1024-bit key is 128 bytes) and can be determined by subtracting the length of the other parts of this header from the total length of the certificate as found in *Hdr.dwLength*.

Information

The **WIN_CERTIFICATE_UEFI_PKCS1_15** structure is derived from **WIN_CERTIFICATE** and encapsulate the information needed to implement the RSASSA-PKCS1-v1_5 digital signature algorithm as specified in RFC2437, sections 8-9.

79) Page 1061, Section 23.2

Removed from the **EFI_IP4_VARIABLE_DATA**: **ProtocolGuid**.

Page 1062: Added an instance handle to the **EFI_IP4_ADDRESS_PAIR**.

```
//*******************************************
// EFI_IP4_VARIABLE_DATA
//*******************************************
typedef struct {
  EFI_HANDLE              DriverHandle;
  UINT32                  AddressCount;
  EFI_IP4_ADDRESS_PAIR    AddressPairs[1];
} EFI_IP4_VARIABLE_DATA;
```

DriverHandle	The handle of the driver that creates this entry.
AddressCount	The number of IPv4 address and subnet mask pairs that follow this data structure.
AddressPairs	List of IPv4 address and subnet mask pairs that are currently in use. Type **EFI_IP4_ADDRESS_PAIR** is defined below.

```
//*******************************************
// EFI_IP4_ADDRESS_PAIR
//*******************************************
typedef struct{
  EFI_HANDLE          InstanceHandle;
  EFI_IPv4_ADDRESS    Ip4Address;
  EFI_IPv4_ADDRESS    SubnetMask;
```

```
} EFI_IP4_ADDRESS_PAIR;
```

InstanceHandle	The EFI IPv4 Protocol instance handle that is using this address/subnetmask pair.
Ip4Address	IPv4 address in network byte order.

80) Page 1167, Appendix A.

Remove ambiguity about GUIDs so that Appendix A reads as follows:

All EFI GUIDs (Globally Unique Identifiers) have the format described in RFC 4122 and comply with the referenced algorithms for generating GUIDs. It should also be noted that TimeLow, TimeMid, TimeHighAndVersion fields in the EFI are encoded as little endian. The following table defines the format of an EFI GUID (128 bits).

Table 168. EFI GUID Format

Mnemonic	Byte Offset	Byte Length	Description
TimeLow	0	4	The low field of the timestamp.
TimeMid	4	2	The middle field of the timestamp.
TimeHighAndVersion	6	2	The high field of the timestamp multiplexed with the version number.
ClockSeqHighAndReserved	8	1	The high field of the clock sequence multiplexed with the variant.
ClockSeqLow	9	1	The low field of the clock sequence.
Node	10	6	The spatially unique node identifier. This can be based on any IEEE 802 address obtained from a network card. If no network card exists in the system, a cryptographic-quality random number can be used.

This appendix for GUID defines a 60-bit timestamp format that is used to generate the GUID. All EFI time information is stored in 64-bit structures that contain the following format: The timestamp is a 60-bit value that is represented by Coordinated Universal Time (UTC) as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582 (the date of Gregorian reform to the Christian calendar). This time value will not roll over until the year 3400 AD. It is assumed that a future version of the EFI specification can deal with the year-3400 issue by extending this format if necessary.

81) Appendix D, page 1181, Table 174.

Supported 32-bit Range, 64-bit Architecture Range and Description values changed for all four rows as follows:

Supported 32-bit Range	Supported 64-bit Architecture Ranges	Description
0x00000000-0x1fffffff	0x0000000000000000-0x1fffffffffffffff	Success and warning codes reserved for use by UEFI main specification.
0x20000000-0x3fffffff	0x2000000000000000-0x3fffffffffffffff	Success and warning codes reserved for use by UEFI main specification.

0x80000000-0x9fffffff	0x8000000000000000-0x9fffffffffffffff	Error codes reserved for use by UEFI main spec.
0xa0000000-0xbfffffff	0xa000000000000000-0xbfffffffffffffff	Error codes reserved for use by Platform Initialization Specification.

82) Page 1215Section E.3.4.12.

Add a type definition “PXE_MEDIA_PROTOCOL” to support PXE in UEFI specification to become Section E.3.4.13, containing the following text:

E.3.4.13 PXE_MEDIA_PROTOCOL

Protocol type. This will be copied into the media header without doing byte swapping. Protocol type numbers can be obtained from the assigned numbers in RFC 1700.

```
typedef UINT16      PXE_MEDIA_PROTOCOL;
```

83) PAGE 1359, Table 184. correct the typo "EFI 11.0" to read "EFI 1.10".