Intel® Platform Innovation Framework for EFI Boot Script Specification

Draft for Review

Version 0.91

April 1, 2004

Intel order number xxxxxx-001

Revision History

Revision	Revision History	Date
0.9	First public release.	9/16/03
0.91	Removed SMBus type definitions from EFI_BOOT_SCRIPT_SMBUS_EXECUTE_OPCODE and inserted pointers to the definitions in the *SMBus PPI Specification*.	4/1/04

Contents

Figures

Tables

1 Introduction

Overview

This specification defines the core code and services that are required for an implementation of the boot script in the Intel® Platform Innovation Framework for EFI (hereafter referred to as the "Framework"). The Framework *boot script* is a script into which configuration information about the platform is recorded for use during different boot paths. This specification does the following:

- Describes the mechanism that is used to execute and record the boot script
- Provides code definitions for various boot scripts that are architecturally required by the *Intel® Platform Innovation Framework for EFI Architecture Specification*

Requirements

This Framework boot script design must meet the following requirements:

- All aspects of this design must comply with the following:
  - Intel® Platform Innovation Framework for EFI Architecture Specification
  - *Intel® Platform Innovation Framework for EFI Pre-EFI Initialization Core Interface Specification* (PEI CIS)
  - *Intel® Platform Innovation Framework for EFI Driver Execution Environment Core Interface Specification* (DXE CIS)
  - ACPI 2.0 specification
- The design must enable size efficiency, code reuse, and maintainability.

Conventions Used in This Document

This document uses the typographic and illustrative conventions described below.

Data Structure Descriptions

Intel® processors based on 32-bit Intel® architecture (IA-32) are “little endian” machines. This distinction means that the low-order byte of a multibyte data item in memory is at the lowest address, while the high-order byte is at the highest address. Processors of the Intel® Itanium® processor family may be configured for both “little endian” and “big endian” operation. All implementations designed to conform to this specification will use “little endian” operation.

In some memory layout descriptions, certain fields are marked *reserved*. Software must initialize such fields to zero and ignore them when read. On an update operation, software must preserve any reserved field.

The data structures described in this document generally have the following format:

STRUCTURE NAME: The formal name of the data structure.

Summary: A brief description of the data structure.

Prototype: A “C-style” type declaration for the data structure.

Parameters: A brief description of each field in the data structure prototype.

Description: A description of the functionality provided by the data structure, including any limitations and caveats of which the caller should be aware.

Related Definitions: The type declarations and constants that are used only by this data structure.

Protocol Descriptions

The protocols described in this document generally have the following format:

Protocol Name: The formal name of the protocol interface.

Summary: A brief description of the protocol interface.

GUID: The 128-bit Globally Unique Identifier (GUID) for the protocol interface.

Protocol Interface Structure: A "C-style" data structure definition containing the procedures and data fields produced by this protocol interface.

Parameters: A brief description of each field in the protocol interface structure.

Description: A description of the functionality provided by the interface, including any limitations and caveats of which the caller should be aware.

Related Definitions: The type declarations and constants that are used in the protocol interface structure or any of its procedures.

Procedure Descriptions

The procedures described in this document generally have the following format:

ProcedureName(): The formal name of the procedure.

Summary: A brief description of the procedure.

Prototype: A "C-style" procedure header defining the calling sequence.

Parameters: A brief description of each field in the procedure prototype.

Description: A description of the functionality provided by the interface, including any limitations and caveats of which the caller should be aware.

Related Definitions: The type declarations and constants that are used only by this procedure.

Status Codes Returned: A description of any codes returned by the interface. The procedure is required to implement any status codes listed in this table. Additional error codes may be returned, but they will not be tested by standard compliance tests, and any software that uses the procedure cannot depend on any of the extended error codes that an implementation may provide.

PPI Descriptions

A PEIM-to-PEIM Interface (PPI) description generally has the following format:

PPI Name: The formal name of the PPI.

Summary: A brief description of the PPI.

GUID: The 128-bit Globally Unique Identifier (GUID) for the PPI.

PPI Interface Structure: A "C-style" procedure template defining the PPI calling structure.

Parameters: A brief description of each field in the PPI structure.

Description: A description of the functionality provided by the interface, including any limitations and caveats of which the caller should be aware.

Related Definitions: The type declarations and constants that are used only by this interface.

Status Codes Returned: A description of any codes returned by the interface. The PPI is required to implement any status codes listed in this table. Additional error codes may be returned, but they will not be tested by standard compliance tests, and any software that uses the procedure cannot depend on any of the extended error codes that an implementation may provide.

Pseudo-Code Conventions

Pseudo code is presented to describe algorithms in a more concise form. None of the algorithms in this document are intended to be compiled directly. The code is presented at a level corresponding to the surrounding text.

In describing variables, a *list* is an unordered collection of homogeneous objects. A *queue* is an ordered list of homogeneous objects. Unless otherwise noted, the ordering is assumed to be First In First Out (FIFO).

Pseudo code is presented in a C-like format, using C conventions where appropriate. The coding style, particularly the indentation style, is used for readability and does not necessarily comply with an implementation of the *Extensible Firmware Interface Specification*.

Typographic Conventions

This document uses the typographic and illustrative conventions described below:

Plain text	The normal text typeface is used for the vast majority of the descriptive text in a specification.
Plain text (blue)	In the online help version of this specification, any plain text that is underlined and in blue indicates an active link to the cross-reference. Click on the word to follow the hyperlink. Note that these links are *not* active in the PDF of the specification.
Bold	In text, a **Bold** typeface identifies a processor register name. In other instances, a **Bold** typeface can be used as a running head within a paragraph.
Italic	In text, an *Italic* typeface can be used as emphasis to introduce a new term or to indicate a manual or specification name.
`BOLD Monospace`	Computer code, example code segments, and all prototype code segments use a `BOLD Monospace` typeface with a dark red color. These code listings normally appear in one or more separate paragraphs, though words or segments can also be embedded in a normal text paragraph.
`Bold Monospace`	In the online help version of this specification, words in a `Bold Monospace` typeface that is underlined and in blue indicate an active hyperlink to the code definition for that function or type definition. Click on the word to follow the hyperlink. Note that these links are *not* active in the PDF of the specification. Also, these inactive links in the PDF may instead have a `Bold Monospace` appearance that is underlined but in dark red. Again, these links are not active in the PDF of the specification.
`Italic Monospace`	In code or in text, words in *`Italic Monospace`* indicate placeholder names for variable information that must be supplied (i.e., arguments).
`Plain Monospace`	In code, words in a `Plain Monospace` typeface that is a dark red color but is not bold or italicized indicate pseudo code or example code. These code segments typically occur in one or more separate paragraphs.
text text text	In the PDF of this specification, text that is highlighted in yellow indicates that a change was made to that text since the previous revision of the PDF. The highlighting indicates only that a change was made since the previous version; it does not specify what changed. If text was deleted and thus cannot be highlighted, a note in red and highlighted in yellow (that looks like *(Note: text text text.)*) appears where the deletion occurred.

See the master Framework glossary in the Framework Interoperability and Component Specifications help system for definitions of terms and abbreviations that are used in this document or that might be useful in understanding the descriptions presented in this document.

See the master Framework references in the Interoperability and Component Specifications help system for a complete list of the additional documents and specifications that are required or suggested for interpreting the information presented in this document.

The Framework Interoperability and Component Specifications help system is available at the following URL:

http://www.intel.com/technology/framework/spec.htm

2 Design Discussion

Framework Boot Script

The Framework boot script is intended to generalize the process of platform initialization in that it can be viewed as a sequence of the following:

- Accessing the I/O, memory, and PCI configuration space
- Executing specific microprocessor instructions

The Framework boot script is especially useful in a resume from the Advanced Configuration and Power Interface (ACPI) S3 system sleep state. During a normal boot, the Framework initializes the platform in a phased fashion. In the Pre-EFI Initialization (PEI) phase, the Framework initializes the platform with enough configurations to allow execution of the Driver Execution Environment (DXE) phase. During the DXE phase, numerous DXE drivers collectively continue to configure the platform to a final preboot state. In contrast, in the ACPI resume boot path, the Framework needs to restore configuration done in both the PEI and DXE phases.

However, it is not effective to make the DXE phase aware of the boot path because the time that is allotted to complete an S3 resume is very constrained; Microsoft* requires 0.5 seconds. To avoid the DXE phase in an S3 resume, various chipset drivers record information on the following as a boot script:

- I/O
- PCI
- Memory
- System Management Bus (SMBus)
- Other specific operations or routines that are necessary to restore the chipset and processor configuration

This boot script can be copied into a nonvolatile storage (NVS) memory region. When the system wakes up and runs the S3 resume boot path, a boot script engine Pre-EFI Initialization Module (PEIM) is able to execute the boot script to restore the configuration done in the previous DXE phase.

See the next topics for figures showing how the boot script works in a normal boot path and during an S3 resume.

Boot Script Usage Model

The figure below shows how the boot script works in a normal boot path and during an S3 resume.

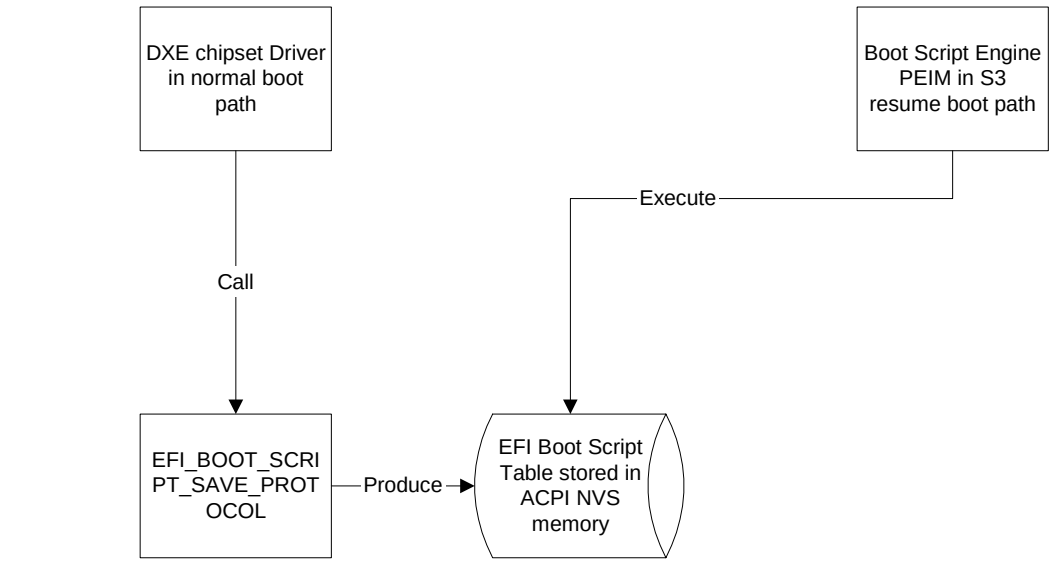

Figure 2-1. Boot Script Usage Model

Role of Boot Script in S3 Resume Boot Path

The figure below shows the role of the boot script and the boot script table in a normal boot path and the S3 resume boot path.

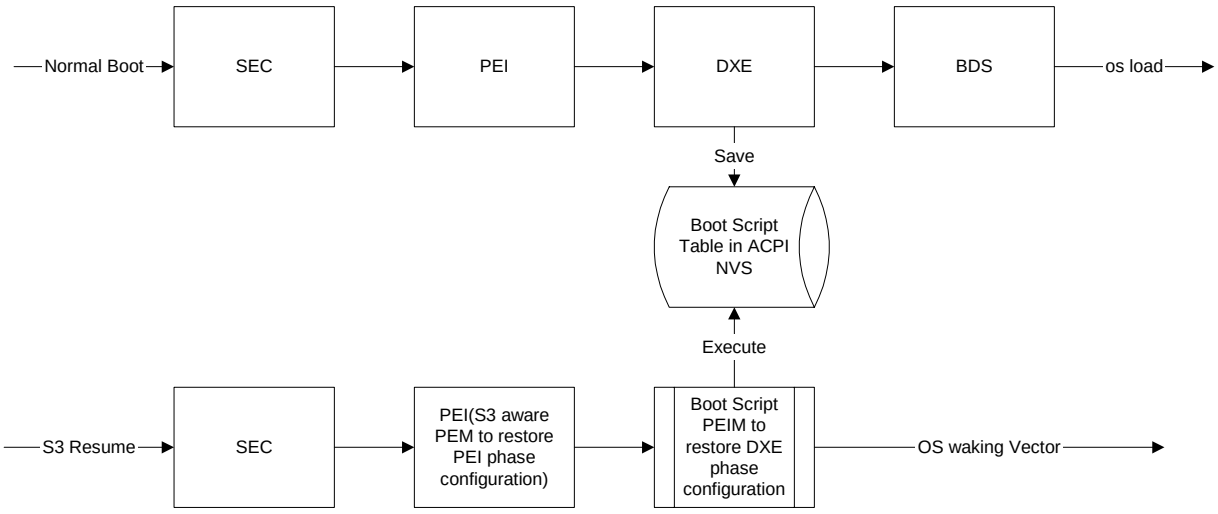

Figure 2-2. Role of Boot Script Usage in S3 Resume Boot Path

3
Code Definitions

Introduction

This section contains the basic definitions for storing firmware volumes in block access type devices. The following protocols, PPIs, and their respective member functions are defined in this section:

- **EFI_BOOT_SCRIPT_SAVE_PROTOCOL**
- **EFI_PEI_BOOT_SCRIPT_EXECUTER_PPI**

This section also contains the definitions for additional data types and structures that are subordinate to the structures in which they are called. The following types or structures can be found in "Related Definitions" of the parent data structure, protocol, or function definition:

- **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**
- Boot script opcode definitions
- **EFI_BOOT_SCRIPT_WIDTH**

(Note: The last three bullets that were in the second bulleted list in the 0.9 version were deleted for the 0.91 version.)

Boot Script Save Protocol

EFI_BOOT_SCRIPT_SAVE_PROTOCOL

Summary

Used to store or record various boot scripts into boot script tables.

GUID

```
#define EFI_BOOT_SCRIPT_SAVE_GUID \
{ 0x470e1529, 0xb79e, 0x4e32, 0xa0, 0xfe, 0x6a,0x15, 0x6d, 0x29,
0xf9, 0xb2 }
```

Protocol Interface Structure

```
typedef struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL {
  EFI_BOOT_SCRIPT_WRITE              Write;
  EFI_BOOT_SCRIPT_CLOSE_TABLE        CloseTable;
} EFI_BOOT_SCRIPT_SAVE_PROTOCOL;
```

Parameters

Write

Writes various boot scripts to a boot script table. See the **Write()** function description.

CloseTable

Retrieves and closes a script table. See the **CloseTable()** function description.

Description

The **EFI_BOOT_SCRIPT_SAVE_PROTOCOL** publishes the Framework boot script abstractions. This protocol is not required for all platforms.

This protocol allows different drivers to record boot scripts. There are different types of boot scripts, which are then grouped into tables. Currently the only meaningful table is for the S3 resume boot path. The implementer can also choose to hide this protocol behind a DXE library.

EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()

Summary

Adds a record into a specified Framework boot script table.

Prototype

```
typedef
EFI_STATUS
 (EFIAPI *EFI_BOOT_SCRIPT_WRITE) (
  IN struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL    *This,
  IN UINT16                                   TableName,
  IN UINT16                                   OpCode,
  ...
);
```

Parameters

This

A pointer to the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL** instance.

TableName

Name of the script table. Currently, the only meaningful value is **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**. Type **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** is defined in "Related Definitions" below.

OpCode

The operation code (opcode) number. See "Related Definitions" below for the defined opcode types.

…

Argument list that is specific to each opcode. See the following subsections for the definition of each opcode.

Description

This function is used to store a boot script record into a given boot script table. If the table specified by *TableName* is nonexistent in the system, a new table will automatically be created and then the script record will be added into the new table. A boot script table can add new script records until **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.CloseTable()** is called. Currently, the only meaningful table name is **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**. This function is responsible for allocating necessary memory for the script.

This function has a variable parameter list. The exact parameter list depends on the *OpCode* that is passed into the function. If an unsupported *OpCode* or illegal parameter list is passed in, this function returns **EFI_INVALID_PARAMETER**.

If there are not enough resources available for storing more scripts, this function returns **EFI_OUT_OF_RESOURCES**.

Related Definitions

```
//*****************************************
// EFI_ACPI_S3_RESUME_SCRIPT_TABLE
//*****************************************

#define EFI_ACPI_S3_RESUME_SCRIPT_TABLE    0x00


//*****************************************
// EFI Boot Script Opcode definitions
//*****************************************
```

NOTE

Click the links in the #define statements below to jump to the **`Write()`** *function description for that opcode.*

```
#define EFI_BOOT_SCRIPT_IO_WRITE_OPCODE                0x00
#define EFI_BOOT_SCRIPT_IO_READ_WRITE_OPCODE           0x01
#define EFI_BOOT_SCRIPT_MEM_WRITE_OPCODE               0x02
#define EFI_BOOT_SCRIPT_MEM_READ_WRITE_OPCODE          0x03
#define EFI_BOOT_SCRIPT_PCI_CONFIG_WRITE_OPCODE        0x04
#define EFI_BOOT_SCRIPT_PCI_CONFIG_READ_WRITE_OPCODE   0x05
#define EFI_BOOT_SCRIPT_SMBUS_EXECUTE_OPCODE           0x06
#define EFI_BOOT_SCRIPT_STALL_OPCODE                   0x07
#define EFI_BOOT_SCRIPT_DISPATCH_OPCODE                0x08


//*****************************************
// EFI_BOOT_SCRIPT_WIDTH
//*****************************************

typedef enum {
  EfiBootScriptWidthUint8,
  EfiBootScriptWidthUint16,
  EfiBootScriptWidthUint32,
  EfiBootScriptWidthUint64,
  EfiBootScriptWidthFifoUint8,
  EfiBootScriptWidthFifoUint16,
  EfiBootScriptWidthFifoUint32,
  EfiBootScriptWidthFifoUint64,
  EfiBootScriptWidthFillUint8,
  EfiBootScriptWidthFillUint16,
  EfiBootScriptWidthFillUint32,
  EfiBootScriptWidthFillUint64,
  EfiBootScriptWidthMaximum
} EFI_BOOT_SCRIPT_WIDTH;
```

Status Codes Returned

EFI_SUCCESS	The operation succeeded. A record was added into the specified script table.
EFI_INVALID_PARAMETER	The parameter is illegal or the given boot script is not supported.
EFI_OUT_OF_RESOURCES	There is insufficient memory to store the boot script.

Opcodes for Write()

EFI_BOOT_SCRIPT_IO_WRITE_OPCODE

Summary

Adds a record for an I/O write operation into a specified boot script table.

Prototype

```
typedef
EFI_STATUS
 (EFIAPI *EFI_BOOT_SCRIPT_WRITE) (
  IN  struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL *This,
  IN  UINT16                                TableName,
  IN  UINT16                                OpCode,
  IN  EFI_BOOT_SCRIPT_WIDTH                 Width,
  IN  UINT64                                Address,
  IN  UINTN                                 Count,
  IN  VOID                                  *Buffer
);
```

Parameters

This

A pointer to the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL** instance.

TableName

Name of the script table. Currently, the only meaningful value is **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**. Type **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

OpCode

Must be set to **EFI_BOOT_SCRIPT_IO_WRITE_OPCODE**. Type **EFI_BOOT_SCRIPT_IO_WRITE_OPCODE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Width

The width of the I/O operations. Enumerated in **EFI_BOOT_SCRIPT_WIDTH**. Type **EFI_BOOT_SCRIPT_WIDTH** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Address

The base address of the I/O operations.

Count

The number of I/O operations to perform. The number of bytes moved is *Width* size * *Count*, starting at *Address*.

Buffer

The source buffer from which to write data. The buffer size is *Width* size * *Count*.

Description

This function adds an I/O write record into a specified boot script table. On script execution, this operation writes the presaved value into the specified I/O ports.

Status Codes Returned

See "Status Codes Returned" in **Write()**.

EFI_BOOT_SCRIPT_IO_READ_WRITE_OPCODE

Summary

Adds a record for an I/O modify operation into a specified boot script table.

Prototype

```
typedef
EFI_STATUS
 (EFIAPI *EFI_BOOT_SCRIPT_WRITE) (
  IN  struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL     *This,
  IN  UINT16                                    TableName,
  IN  UINT16                                    OpCode,
  IN  EFI_BOOT_SCRIPT_WIDTH                     Width,
  IN  UINT64                                    Address,
  IN  VOID                                      *Data,
  IN  VOID                                      *DataMask
  );
```

Parameters

This

A pointer to the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL** instance.

TableName

Name of the script table. Currently, the only meaningful value is **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**. Type **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** is defined in “Related Definitions” in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

OpCode

Must be set to **EFI_BOOT_SCRIPT_IO_READ_WRITE_OPCODE**. Type **EFI_BOOT_SCRIPT_IO_READ_WRITE_OPCODE** is defined in “Related Definitions” in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Width

The width of the I/O operations. Enumerated in **EFI_BOOT_SCRIPT_WIDTH**. Type **EFI_BOOT_SCRIPT_WIDTH** is defined in “Related Definitions” in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Address

The base address of the I/O operations.

Data

A pointer to the data to be **OR**-ed.

DataMask

A pointer to the data mask to be **AND**-ed with the data read from the register.

Description

This function adds an I/O read and write record into the specified boot script table. When the script is executed, the register at *Address* is read, **AND**-ed with *DataMask*, and **OR**-ed with *Data*, and finally the result is written back.

Status Codes Returned

See "Status Codes Returned" in **Write()**.

EFI_BOOT_SCRIPT_MEM_WRITE_OPCODE

Summary

Adds a record for a memory write operation into a specified boot script table.

Prototype

```
typedef
EFI_STATUS
 (EFIAPI *EFI_BOOT_SCRIPT_WRITE) (
  IN  struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL    *This,
  IN  UINT16                                   TableName,
  IN  UINT16                                   OpCode,
  IN  EFI_BOOT_SCRIPT_WIDTH                    Width,
  IN  UINT64                                   Address,
  IN  UINTN                                    Count,
  IN  VOID                                     *Buffer
  );
```

Parameters

This

A pointer to the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL** instance.

TableName

Name of the script table. Currently, the only meaningful value is **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**. Type **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

OpCode

Must be set to **EFI_BOOT_SCRIPT_MEM_WRITE_OPCODE**. Type **EFI_BOOT_SCRIPT_MEM_WRITE_OPCODE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Width

The width of the memory operations. Enumerated in **EFI_BOOT_SCRIPT_WIDTH**. Type **EFI_BOOT_SCRIPT_WIDTH** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Address

The base address of the memory operations. *Address* needs alignment if required.

Count

The number of memory operations to perform. The number of bytes moved is *Width* size * *Count*, starting at *Address*.

Buffer

The source buffer from which to write the data. The buffer size is *Width* size * *Count*.

Description

This function adds a memory write record into a specified boot script table. When the script is executed, this operation writes the presaved value into the specified memory location.

Status Codes Returned

See "Status Codes Returned" in **Write()**.

EFI_BOOT_SCRIPT_MEM_READ_WRITE_OPCODE

Summary

Adds a record for a memory modify operation into a specified boot script table.

Prototype

```
typedef
EFI_STATUS
 (EFIAPI *EFI_BOOT_SCRIPT_WRITE) (
  IN  struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL    *This,
  IN  UINT16                                   TableName,
  IN  UINT16                                   OpCode,
  IN  EFI_BOOT_SCRIPT_WIDTH                    Width,
  IN  UINT64                                   Address,
  IN  VOID                                     *Data,
  IN  VOID                                     *DataMask
  );
```

Parameters

This

A pointer to the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL** instance.

TableName

Name of the script table. Currently, the only meaningful value is **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**. Type **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

OpCode

Must be set to **EFI_BOOT_SCRIPT_MEM_READ_WRITE_OPCODE**. Type **EFI_BOOT_SCRIPT_MEM_READ_WRITE_OPCODE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Width

The width of the memory operations. Enumerated in **EFI_BOOT_SCRIPT_WIDTH**. Type **EFI_BOOT_SCRIPT_WIDTH** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Address

The base address of the memory operations. *Address* needs alignment if required.

Data

A pointer to the data to be **OR**-ed.

DataMask

A pointer to the data mask to be **AND**-ed with the data read from the register.

Description

This function adds a memory read and write record into a specified boot script table. When the script is executed, the memory at *Address* is read, **AND**-ed with *DataMask*, and **OR**-ed with *Data*, and finally the result is written back.

Status Codes Returned

See "Status Codes Returned" in **Write()**.

EFI_BOOT_SCRIPT_PCI_CONFIG_WRITE_OPCODE

Summary

Adds a record for a PCI configuration space write operation into a specified boot script table.

Prototype

```
typedef
EFI_STATUS
 (EFIAPI *EFI_BOOT_SCRIPT_WRITE) (
  IN  struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL    *This,
  IN  UINT16                                   TableName,
  IN  UINT16                                   OpCode,
  IN  EFI_BOOT_SCRIPT_WIDTH                    Width,
  IN  UINT64                                   Address,
  IN  UINTN                                    Count,
  IN  VOID                                     *Buffer
  )
```

Parameters

This

A pointer to the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL** instance.

TableName

Name of the script table. Currently, the only meaningful value is **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**. Type **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

OpCode

Must be set to **EFI_BOOT_SCRIPT_PCI_CONFIG_WRITE_OPCODE**. Type **EFI_BOOT_SCRIPT_PCI_CONFIG_WRITE_OPCODE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Width

The width of the PCI operations. Enumerated in **EFI_BOOT_SCRIPT_WIDTH**. Type **EFI_BOOT_SCRIPT_WIDTH** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Address

The address within the PCI configuration space. See Table 12-1 in the *Extensible Firmware Interface Specification*, version 1.10, for the address format.

Count

The number of PCI operations to perform. The number of bytes moved is *Width* size * *Count*, starting at *Address*.

Buffer

The source buffer from which to write the data. The buffer size is *Width* size * *Count*.

Description

This function adds a PCI configuration space write record into a specified boot script table. When the script is executed, this operation writes the presaved value into the specified location in PCI configuration space.

Status Codes Returned

See "Status Codes Returned" in **Write()**.

EFI_BOOT_SCRIPT_PCI_CONFIG_READ_WRITE_OPCODE

Summary

Adds a record for a PCI configuration space modify operation into a specified boot script table.

Prototype

```
typedef
EFI_STATUS
 (EFIAPI *EFI_BOOT_SCRIPT_WRITE) (
  IN  struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL     *This,
  IN  UINT16                                    TableName,
  IN  UINT16                                    OpCode,
  IN  EFI_BOOT_SCRIPT_WIDTH                     Width,
  IN  UINT64                                    Address,
  IN  VOID                                      *Data,
  IN  VOID                                      *DataMask
  )
```

Parameters

This

A pointer to the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL** instance.

TableName

Name of the script table. Currently, the only meaningful value is **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**. Type **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

OpCode

Must be set to **EFI_BOOT_SCRIPT_PCI_CONFIG_READ_WRITE_OPCODE**. Type **EFI_BOOT_SCRIPT_PCI_CONFIG_READ_WRITE_OPCODE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Width

The width of the PCI operations. Enumerated in **EFI_BOOT_SCRIPT_WIDTH**. Type **EFI_BOOT_SCRIPT_WIDTH** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Address

The address within the PCI configuration space. See Table 12-1 in the *Extensible Firmware Interface Specification*, version 1.10, for the address format.

Data

A pointer to the data to be **OR**-ed. The size depends on *Width*.

DataMask

A pointer to the data mask to be **AND**-ed.

Description

This function adds a PCI configuration read and write record into a specified boot script table. When the script is executed, the PCI configuration space location at *Address* is read, **AND**-ed with *DataMask*, and **OR**-ed with, and finally the result is written back.

Status Codes Returned

See "Status Codes Returned" in **Write()**.

EFI_BOOT_SCRIPT_SMBUS_EXECUTE_OPCODE

Summary

Adds a record for an SMBus command execution into a specified boot script table.

Prototype

```
typedef
EFI_STATUS
 (EFIAPI *EFI_BOOT_SCRIPT_WRITE) (
  IN struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL     *This,
  IN UINT16                                    TableName,
  IN UINT16                                    OpCode,
  IN EFI_SMBUS_DEVICE_ADDRESS                  SlaveAddress,
  IN EFI_SMBUS_DEVICE_COMMAND                  Command,
  IN EFI_SMBUS_OPERATION                       Operation,
  IN BOOLEAN                                   PecCheck,
  IN UINTN                                     *Length,
  IN VOID                                      *Buffer
  )
```

Parameters

This

A pointer to the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL** instance.

TableName

Name of the script table. Currently, the only meaningful value is **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**. Type **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

OpCode

Must be set to **EFI_BOOT_SCRIPT_SMBUS_EXECUTE_OPCODE**. Type **EFI_BOOT_SCRIPT_SMBUS_EXECUTE_OPCODE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

SlaveAddress

The SMBus address for the slave device that the operation is targeting. Type **EFI_SMBUS_DEVICE_ADDRESS** is defined in **EFI_PEI_SMBUS_PPI.Execute()** in the *Intel® Platform Innovation Framework for EFI SMBus PPI Specification*.

Command

The command that is transmitted by the SMBus host controller to the SMBus slave device. The interpretation is SMBus slave device specific. It can mean the offset to a list of functions inside an SMBus slave device. Type **EFI_SMBUS_DEVICE_COMMAND** is defined in **EFI_PEI_SMBUS_PPI.Execute()** in the *Intel® Platform Innovation Framework for EFI SMBus PPI Specification*.

Operation

Indicates which particular SMBus protocol it will use to execute the SMBus transactions. Type **EFI_SMBUS_OPERATION** is defined in **EFI_PEI_SMBUS_PPI.Execute()** in the *Intel® Platform Innovation Framework for EFI SMBus PPI Specification*.

PecCheck

Defines if Packet Error Code (PEC) checking is required for this operation.

Length

A pointer to signify the number of bytes that this operation will do.

Buffer

Contains the value of data to execute to the SMBUS slave device.

Description

This function adds an SMBus command execution record into a specified boot script table. When the script is executed, this operation executes a specified SMBus command.

(Note: The "Related Definitions" subsection and the type definitions for **EFI_SMBUS_DEVICE_ADDRESS**, **EFI_SMBUS_DEVICE_COMMAND**, *and* **EFI_SMBUS_OPERATION** *were deleted for the 0.91 version.)*

Status Codes Returned

See "Status Codes Returned" in **Write()**.

EFI_BOOT_SCRIPT_STALL_OPCODE

Summary

Adds a record for an execution stall on the processor into a specified boot script table.

Prototype

```
typedef
EFI_STATUS
 (EFIAPI *EFI_BOOT_SCRIPT_WRITE) (
  IN  struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL    *This,
  IN  UINT16                                   TableName,
  IN  UINT16                                   OpCode,
  IN  UINTN                                    Duration
  )
```

Parameters

This

A pointer to the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL** instance.

TableName

Name of the script table. Currently, the only meaningful value is **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**. Type **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

OpCode

Must be set to **EFI_BOOT_SCRIPT_STALL_OPCODE**. Type **EFI_BOOT_SCRIPT_STALL_OPCODE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

Duration

Duration in microseconds of the stall.

Description

This function adds a stall record into a specified boot script table. When the script is executed, this operation will stall the system for *Duration* number of microseconds.

Status Codes Returned

See "Status Codes Returned" in **Write()**.

EFI_BOOT_SCRIPT_DISPATCH_OPCODE

Summary

Adds a record for dispatching specified arbitrary code into a specified boot script table.

Prototype

```
typedef
EFI_STATUS
 (EFIAPI *EFI_BOOT_SCRIPT_WRITE) (
  IN  struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL   *This,
  IN  UINT16                                  TableName,
  IN  UINT16                                  OpCode,
  IN  EFI_PHYSICAL_ADDRESS                    EntryPoint
  )
```

Parameters

This

A pointer to the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL** instance.

TableName

Name of the script table. Currently, the only meaningful value is **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**. Type **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

OpCode

Must be set to **EFI_BOOT_SCRIPT_DISPATCH_OPCODE**. Type **EFI_BOOT_SCRIPT_DISPATCH_OPCODE** is defined in "Related Definitions" in **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()**.

EntryPoint

Entry point of the code to be dispatched. Type **EFI_PHYSICAL_ADDRESS** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

Description

This function adds a dispatch record into a specified boot script table, with which it can run the arbitrary code that is specified. This script can be used to initialize the processor. When the script is executed, the script incurs jumping to the entry point to execute the arbitrary code. After the execution is returned, it goes on executing the next opcode in the table. If the codes to be dispatched have dependencies on other PPIs or codes, the caller should guarantee that all dependencies are sufficient before dispatching the codes.

Status Codes Returned

See "Status Codes Returned" in **Write()**.

EFI_BOOT_SCRIPT_SAVE_PROTOCOL.CloseTable()

Summary

Closes the specified script table.

Prototype

```
typedef
EFI_STATUS
 (EFIAPI *EFI_BOOT_SCRIPT_CLOSE_TABLE) (
  IN  struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL    *This,
  IN  UINT16                                   TableName,
  OUT EFI_PHYSICAL_ADDRESS                     *Address
);
```

Parameters

This

A pointer to the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL** instance.

TableName

Name of the table.

Address

A pointer to the physical address where the table begins. Type **EFI_PHYSICAL_ADDRESS** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

Description

This function closes the specified boot script table and returns the base address of the table. It allocates a new pool to duplicate all the boot scripts in the specified table. Once this function is called, the specified table will be destroyed after it is copied into the allocated pool. As a result, any attempts to add a script record into a closed table will cause a new table to be created. The base address of the allocated pool will be returned in *Address*. After using the boot script table, the caller is responsible for freeing the pool that is allocated by this function. If the boot script table, such as **EFI_ACPI_S3_RESUME_SCRIPT_TABLE**, is required to be stored in a nonperturbed memory region, the caller should copy the table into the nonperturbed memory region by itself.

Status Codes Returned

EFI_SUCCESS	The table was successfully returned.
EFI_NOT_FOUND	The specified table was not created previously.
EFI_OUT_OF_RESOURCES	Memory is insufficient to hold the reorganized boot script table.

Boot Script Executer

EFI_PEI_BOOT_SCRIPT_EXECUTER_PPI

Summary

This PPI produces functions to interpret and execute the Framework boot script table.

GUID

```
#define EFI_PEI_BOOT_SCRIPT_EXECUTER_PPI_GUID  \
  { 0xabd42895, 0x78cf, 0x4872, 0x84, 0x44, 0x1b, 0x5c, 0x18,
  0x0b, 0xfb, 0xff }
```

PPI Interface Structure

```
typedef struct _EFI_PEI_BOOT_SCRIPT_EXECUTER_PPI {
  EFI_PEI_BOOT_SCRIPT_EXECUTE                Execute;
} EFI_PEI_BOOT_SCRIPT_EXECUTER_PPI;
```

Parameters

Execute

Executes a boot script table. See the **Execute()** function description.

Description

This PPI is published by a PEIM upon dispatch and provides an execution engine for the Framework boot script. This PEIM should be platform neutral and have no specific knowledge of platform instructions and other information. The ability to interpret the boot script depends on the abundance of other PPIs that are available. For example, if the script requests an SMBus command execution, the PEIM looks for a relevant PPI that is available to execute it, rather than executing it by issuing the native IA-32 instruction.

The table below lists the PPIs on which each opcode directly depends.

Table 3-1. Opcode PPI Dependencies

Boot Script Opcode	Dependency
EFI_BOOT_SCRIPT_IO_WRITE_OPCODE	EFI_PEI_CPU_IO_PPI
EFI_BOOT_SCRIPT_IO_READ_WRITE_OPCODE	EFI_PEI_CPU_IO_PPI
EFI_BOOT_SCRIPT_MEM_WRITE_OPCODE	EFI_PEI_CPU_IO_PPI
EFI_BOOT_SCRIPT_MEM_READ_WRITE_OPCODE	EFI_PEI_CPU_IO_PPI
EFI_BOOT_SCRIPT_PCI_CONFIG_WRITE_OPCODE	EFI_PEI_PCI_CFG_PPI
EFI_BOOT_SCRIPT_PCI_CONFIG_READ_WRITE_OPCODE	EFI_PEI_PCI_CFG_PPI
EFI_BOOT_SCRIPT_SMBUS_EXECUTE_OPCODE	EFI_PEI_SMBUS_PPI
EFI_BOOT_SCRIPT_STALL_OPCODE	EFI_PEI_STALL_PPI
EFI_BOOT_SCRIPT_DISPATCH_OPCODE	Uncertain

For more information on these dependencies, please see the following specifications:

- Definitions of **EFI_PEI_CPU_IO_PPI**, **EFI_PEI_PCI_CFG_PPI**, and **EFI_PEI_STALL_PPI**:
  Intel® Platform Innovation Framework for EFI Pre-EFI Initialization Core Interface Specification (PEI CIS)
- Definition of **EFI_PEI_SMBUS_PPI**: *Intel® Platform Innovation Framework for EFI SMBus PPI Specification*

EFI_PEI_BOOT_SCRIPT_EXECUTER_PPI.Execute()

Summary

Executes the Framework boot script table.

Prototype

```
typedef
EFI_STATUS
(EFIAPI * EFI_PEI_BOOT_SCRIPT_EXECUTE) (
  IN  EFI_PEI_SERVICES                        **PeiServices,
  IN  struct _EFI_PEI_BOOT_SCRIPT_EXECUTER_PPI  *This,
  IN  EFI_PHYSICAL_ADDRESS                    Address
  IN  EFI_GUID                                *FvFile OPTIONAL
);
```

Parameters

PeiServices

A pointer to the system PEI Services Table. Type **EFI_PEI_SERVICES** is defined in the *Intel® Platform Innovation Framework for EFI Pre-EFI Initialization Core Interface Specification* (PEI CIS).

This

A pointer to the **EFI_PEI_BOOT_SCRIPT_EXECUTER_PPI** instance.

Address

The physical memory address where the table is stored. It must be zero if the table to be executed is stored in a firmware volume file. Type **EFI_PHYSICAL_ADDRESS** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

FvFile

The firmware volume file name that contains the table to be executed. It must be **NULL** if the table to be executed is stored in physical memory. Type **EFI_GUID** is defined in **InstallProtocolInterface()** in the *EFI 1.10 Specification.*

Description

This function executes a Framework boot script table. The boot scripts are recorded using the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.Write()** function, and the entire table can be retrieved using the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.CloseTable()** function. This function interprets every boot script in the table upon the opcode and performs the operation that the boot script requires by calling the appropriate PPI. The caller should ensure that all dependent PPIs and codes to be dispatched are available in memory before calling this function.

Status Codes Returned

EFI_SUCCESS	The boot script table was executed successfully.
EFI_INVALID_PARAMETER	*Address* is zero and *FvFile* is **NULL**.
EFI_NOT_FOUND	The file name specified in *FvFile* cannot be found.
EFI_UNSUPPORTED	The format of the boot script table is invalid.
EFI_UNSUPPORTED	An unsupported opcode occurred in the table.
EFI_UNSUPPORTED	There were opcode execution errors, such as an insufficient dependency.