Intel® Platform Innovation Framework for EFI S3 Resume Boot Path Specification

Version 0.9

September 16, 2003

Intel order number xxxxxx-001

Revision History

Revision	Revision History	Date
0.9	First public release.	9/16/03

intel

Contents

Figures

1 Introduction

Overview

This specification defines the core code and services that are required for an implementation of the S3 resume boot path in the Intel® Platform Innovation Framework for EFI (hereafter referred to as the "Framework"). The *S3 resume boot path* is a special boot path that causes the Framework to take actions different from those in the normal boot path. In this special path, the Framework derives presaved data about the platform's configuration from persistent storage and configures the platform before jumping to the operating system's waking vector.

This specification does the following:

- Describes the basic components of the S3 resume boot path, how it relates to a normal boot path, and how it interacts with other Framework phases and code
- Provides code definitions for the S3-related protocols and PPIs that are architecturally required by the *Intel® Platform Innovation Framework for EFI Architecture Specification*

Goals

This Framework S3 resume boot path design has the following goals:

- **Extensibility:** The Framework S3 resume boot path should easily adapt to different platforms, such as Itanium®-based platforms and those based on 32-bit Intel® architecture (IA-32), by replacing only a few platform-specific modules.
- **High performance:** S3 restoration is most commonly used in notebook computers. Its performance is highly visible to end users.

Requirements

This Framework S3 resume boot path 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)
  - *Advanced Configuration and Power Interface Specification* (hereafter referred to as the "ACPI specification"), revision 2.0
- 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.

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

Assumptions

Multiple Phases of Platform Initialization

The Framework consists of two phases of platform initialization:

- Pre-EFI Initialization (PEI)
- Driver Execution Environment (DXE)

The PEI phase is responsible for initializing enough of the platform's resources to enable the execution of the DXE phase, which is where the majority of platform configuration is performed by different DXE drivers.

Initialization that is done in PEI is not necessarily preserved in DXE. In other words, a DXE driver can override the configuration settings that were derived from PEI. In light of this fact, the preboot platform state that the S3 resume boot path needs to restore is the DXE snapshot of the platform state, rather than the PEI snapshot of the platform state.

Process of Platform Initialization

Platform initialization can be viewed as a flow of the following:

- I/O operations
- Memory operations
- Accessing the PCI configuration space
- A collection of platform-specific actions that can be abstracted by Pre-EFI Initialization Module (PEIM) PEIM-to-PEIM Interfaces (PPIs)

The process of restoring hardware settings in different platforms involves different actions or even different instruction sets. These differences, however, can be abstracted behind PEIM PPIs.

Restoring the Platform

Restoring the Platform

Restoring the Platform

The goal of an S3 resume is to restore the platform to its preboot configuration. However, it is impossible to restore the platform in only one step, without going through all the Framework initialization phases, because the Framework is unable to have *a priori* knowledge of the following:

- Preboot configuration that is introduced by various PEIMs
- Drivers provided by different vendors

As a result, the Framework still needs to restore the platform in a phased fashion as it does in a normal boot path. The figure below shows the phases in an S3 resume boot path. See the following subsections for details of each phase.

Figure 2-1. Framework S3 Resume Boot Path

Phases in the S3 Resume Boot Path

SEC and the S3 Resume Boot Path

The Security (SEC) phase is the first architectural phase in the Framework. It builds the root of trust for the entire system. As such, the SEC phase remains intact in the S3 resume boot path.

PEI

PEI and the S3 Resume Boot Path

The PEI phase initializes the platform with the minimum configuration that is needed to enable the execution of the DXE phase. During the S3 resume boot path, the Framework still needs to restore the PEI portion of configuration.

Each PEIM is “boot path aware” in that the PEIM can call the appropriate PEI service to find out what the current boot path is. This awareness enables the platform to restore more efficiently because the same PEIM can save the configuration during a normal boot path and take advantage of that configuration in the S3 resume boot path. The figure below shows how the PEI phase works in a normal boot path and in an S3 resume boot path.

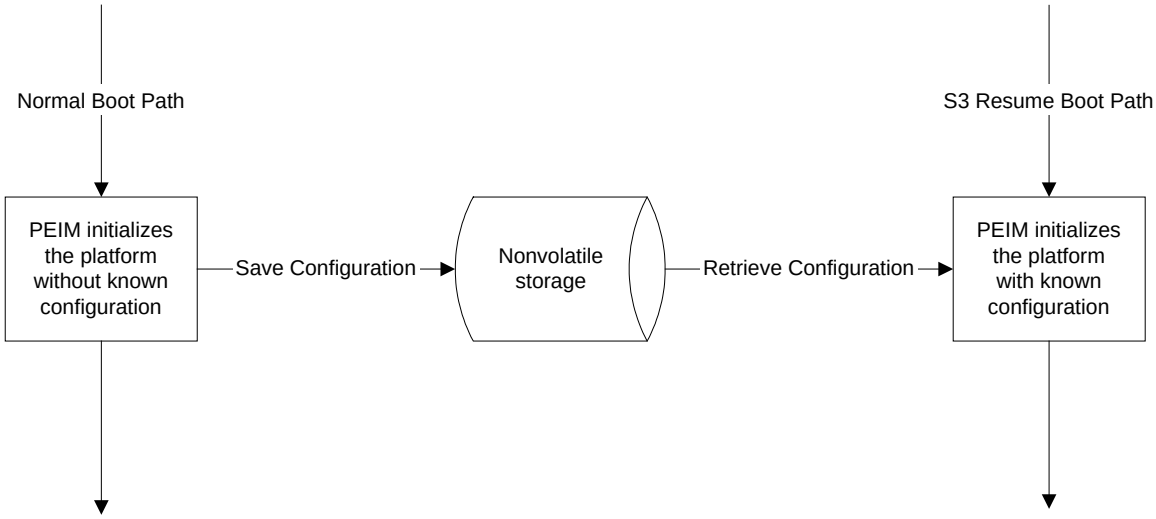

Figure 2-2. PEI Phase in S3 Resume Boot Path

Saving Configuration Data in PEI

There are different ways to save configuration data, such as the firmware volume variable, for the PEI phase in nonvolatile storage (NVS). One way is to save the data directly in the PEI phase. However, if the PEI phase does not implement the capability to write to a firmware volume, a PEIM can choose to pass the configuration data to the DXE phase using a Hand-Off Block (HOB). The PEIM's DXE counterpart or another appropriate DXE component can then save the configuration data. The figure below illustrates this mechanism to save the configuration data. See the *Intel® Platform Innovation Framework for EFI Hand-Off Block (HOB) Specification* for more details on HOBs.

To achieve higher performance, it is recommended to implement the latter mechanism because code running in the PEI phase is more time consuming than code running in the DXE phase. Note that the way to save the configuration data during the PEI phase is outside the scope of this document.

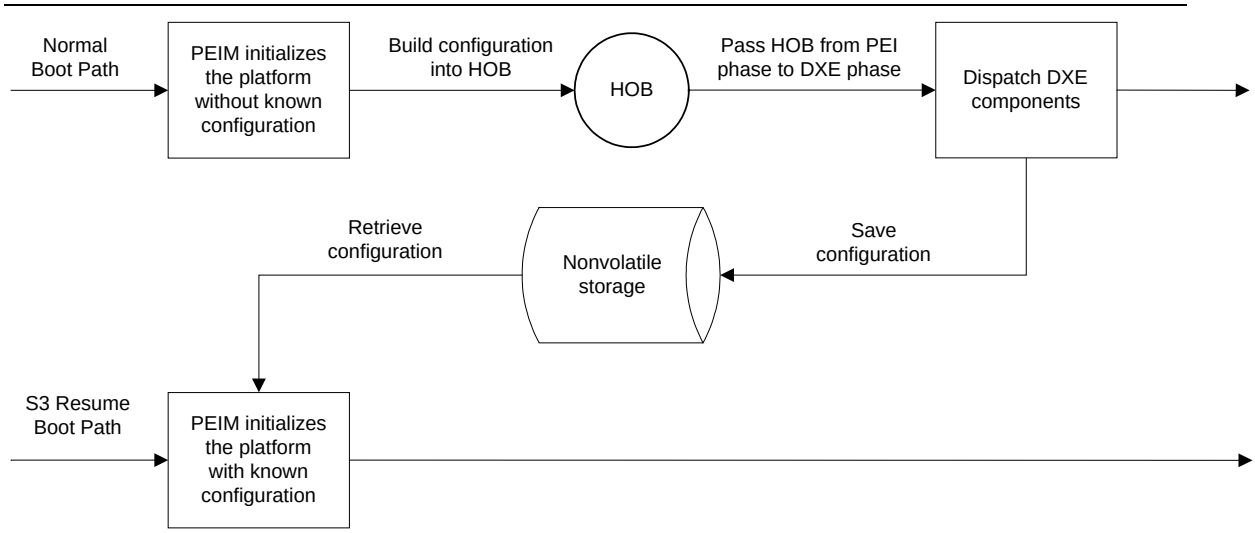

Figure 2-3. Configuration Save for PEI Phase

DXE

DXE and the S3 Resume Boot Path

In the DXE phase during a normal boot path, various DXE drivers collectively bring the platform to the preboot state. However, bringing DXE into the S3 resume boot path and making a DXE driver boot-path aware is very risky for the following reasons:

- The DXE phase hosts numerous services, which makes it rather large.
- Loading DXE from flash is very time consuming.

Even if DXE could be relocated into NVS during a normal boot, the large amount of memory that DXE consumes and the complexity of executing the DXE phase do not justify doing so.

Instead, the Framework provides a boot script that lets the S3 resume boot path avoid the DXE phase altogether, which helps to maximize optimum performance. During a normal boot, DXE drivers record the platform's configuration in the boot script, which is saved in NVS. During the S3 resume boot path, a boot script engine executes the script, thereby restoring the configuration.

The ACPI specification only requires the BIOS to restore chipset and processor configuration. The chipset configuration can be viewed as a series of memory, I/O, and PCI configuration operations, which DXE drivers record in the Framework boot script. During an S3 resume, a boot script engine executes the boot script to restore the chipset settings. Processor configuration involves the following:

- Basic setup for System Management Mode (SMM)
- Microcode updates
- Processor-specific initialization
- Processor cache setting

DXE drivers register arbitrary code in the boot script to restore processor configuration. During the S3 resume boot path, the boot script engine can jump to execute the registered code to restore all processor-related configurations.

Framework Boot Script

Framework Boot Script

The Framework boot script is a script that is used to describe a set of common operations to initialize the platform. Such common operations include I/O, memory, PCI, and System Management Bus (SMBus).

There can be multiple boot script tables in a certain platform. It is assumed that these tables sufficiently describe the actions that need to be taken to restore the preboot platform state.

A PEIM publishes **`EFI_PEI_BOOT_SCRIPT_EXECUTER_PPI`**, which is a PPI that allows the boot script to be executed. Executing the Framework boot script relies on multiple PPIs, including the following, which need to be published before executing the Framework boot script:

- **`EFI_PEI_CPU_IO_PPI`**
- **`EFI_PEI_PCI_CFG_PPI`**
- **`EFI_PEI_SMBUS_PPI`**
- **`EFI_PEI_STALL_PPI`**

See the *Intel® Platform Innovation Framework for EFI Boot Script Specification* for more details on these dependencies.

Boot Script Save Protocol

The **`EFI_BOOT_SCRIPT_SAVE_PROTOCOL`** provides standard APIs to manipulate the Framework boot script table. See the *Intel® Platform Innovation Framework for EFI Boot Script Specification* for more information.

S3 Resume PPI and DXE IPL PPI

The DXE Initial Program Load (IPL) PPI is architecturally the last PPI that is executed in the PEI phase. It is also made aware of the exact boot path that the Framework is currently using. As a result, the DXE IPL PPI is the place to initiate a process to restore the preboot platform state and then jump to the operating system (OS) waking vector. The DXE phase will not be loaded as it was in a normal boot.

When booting with the S3 resume boot path, the DXE IPL PPI will locate the S3 Resume PPI. If it is found, the DXE IPL PPI will transfer control to the S3 Resume PPI. The S3 Resume PPI is responsible for restoring the platform configuration to the preboot state.

The S3 Resume PPI must execute a Framework boot script table to restore the platform configuration. It also needs to locate the OS waking vector and transfer its control to the OS to complete the S3 resume. However, during all these operations, only the memory range that is reserved for an S3 resume can be available for use. Any memory usage that is out of this range may perturb the context of system memory and corrupt the S3 resume.

To execute the Framework boot script and restore the platform configuration, the S3 Resume PPI needs to know the following:

- The memory address of the S3 boot script table or the name of the firmware volume file that contains the S3 boot script table
- The Root System Description Table (RSDT) pointing to the start of the ACPI table that it needs to find the OS waking vector
- Reserved memory range that can be used in the S3 resume boot path

This information can be made available through EFI firmware variables or nonperturbed memory.

The figure below shows the process flow of the DXE IPL PPI and S3 Resume PPI in the S3 resume boot path.

See Code Definitions for the definition of **`EFI_PEI_S3_RESUME_PPI`**. See the *Intel® Platform Innovation Framework for EFI Pre-EFI Initialization Core Interface Specification* (PEI CIS) for the definition of the DXE IPL PPI.

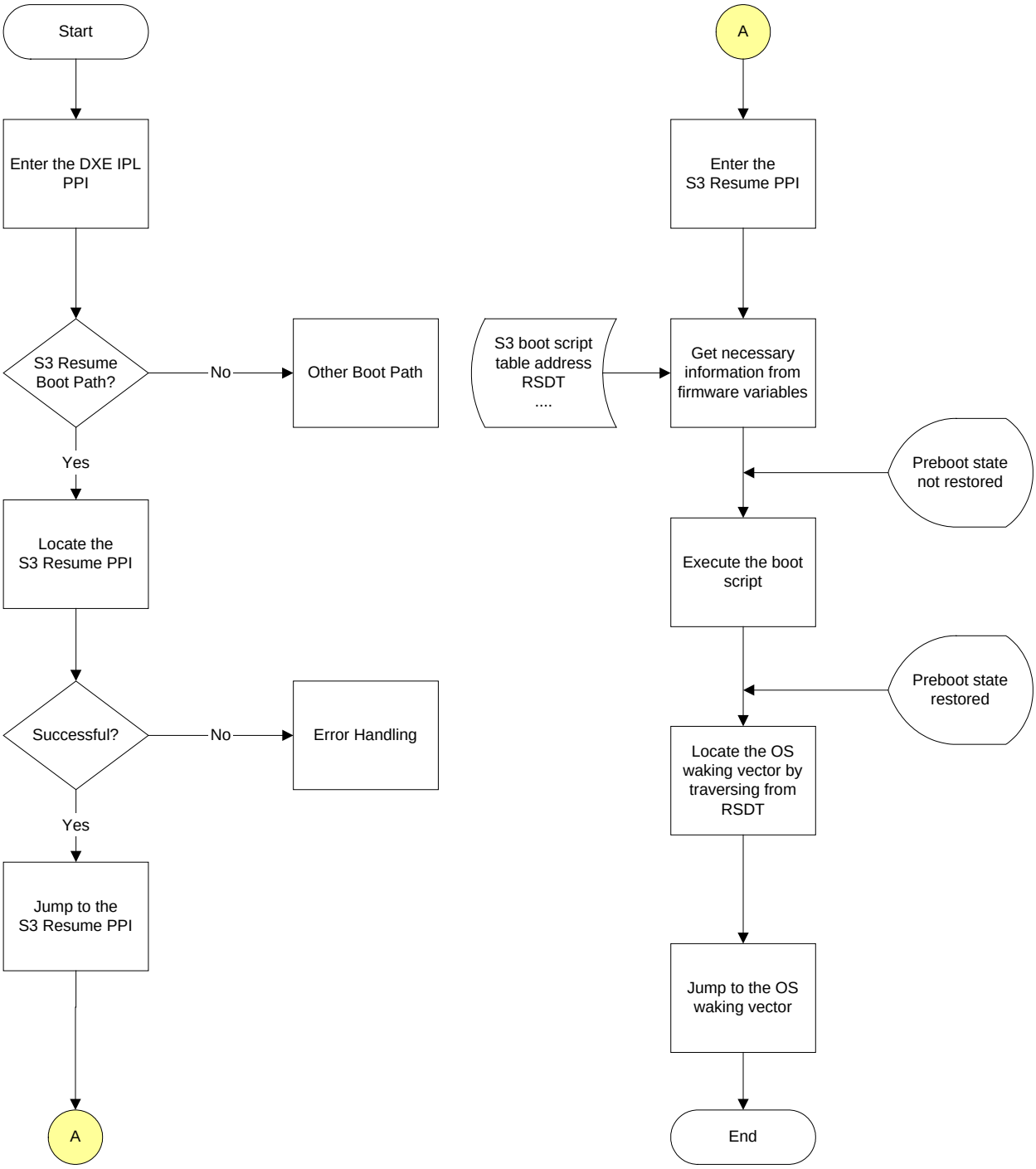

Figure 2-4. DXE IPL PPI and S3 Resume PPI in S3 Resume Boot Path

3
Code Definitions

Introduction

This section contains definitions for the following protocols, PPIs, and functions that are involved in the S3 resume boot path:

- **EFI_ACPI_S3_SAVE_PROTOCOL**
- **EFI_PEI_S3_RESUME_PPI**

ACPI S3 Save Protocol

EFI_ACPI_S3_SAVE_PROTOCOL

Summary

This protocol is used to prepare all information that is needed for the S3 resume boot path. This protocol is not required for all platforms.

GUID

```
#define EFI_ACPI_S3_SAVE_GUID \
   { 0x125f2de1, 0xfb85, 0x440c, 0xa5, 0x4c, 0x4d, 0x99, 0x35,
   0x8a, 0x8d, 0x38 }
```

Protocol Interface Structure

```
typedef struct _EFI_ACPI_S3_SAVE_PROTOCOL{
  EFI_ACPI_GET_LEGACY_MEMORY_SIZE      GetLegacyMemorySize;
  EFI_ACPI_S3_SAVE                     S3Save;
} EFI_ACPI_S3_SAVE_PROTOCOL;
```

Parameters

GetLegacyMemorySize

Gets the size of legacy memory below 1 MB that is required for S3 resume. See the **GetLegacyMemorySize()** function description.

S3Save

Prepare all information for an S3 resume. See the **S3Save()** function description.

Description

The **EFI_ACPI_S3_SAVE_PROTOCOL** is responsible for preparing all the information that the Framework needs to restore the platform's preboot state during an S3 resume boot. This information can include the following:

- The Framework boot script table, containing all necessary operations to initialize the platform
- ACPI table information, such as RSDT, through which the OS waking vector can be located
- Range of reserved memory that can be used on the S3 resume boot path

This protocol can be used after the Framework makes sure that the boot process is complete and that no hardware has been left unconfigured. It is implementation specific where to call this protocol to save all the information.

In the case of an EFI-aware OS, **ExitBootServices()** can be a choice to provide this hook. The currently executing EFI OS loader image calls **ExitBootServices()** to terminate all boot services. After **ExitBootServices()** successfully completes, the loader becomes responsible for the continued operation of the system.

On a normal boot, **ExitBootServices()** checks if the platform supports S3 by looking for **EFI_ACPI_S3_SAVE_PROTOCOL**. If the protocol exists, **ExitBootServices()** will assume that the target platform supports an S3 resume and then call **EFI_ACPI_S3_SAVE_PROTOCOL** to save the S3 resume information. The entire Framework boot script table will then be generated, assuming the platform currently is in the preboot state.

EFI_ACPI_S3_SAVE_PROTOCOL.GetLegacyMemorySize()

Summary

Gets the size of legacy memory below 1 MB that is required during S3 resume.

Prototype

```
typedef
EFI_STATUS
EFI_BOOTSERVICE
(EFIAPI *EFI_ACPI_GET_LEGACY_MEMORY_SIZE) (
  IN struct _EFI_ACPI_S3_SAVE_PROTOCOL   *This,
  OUT UINTN                              *Size
);
```

Parameters

This

A pointer to the **EFI_ACPI_S3_SAVE_PROTOCOL** instance.

Size

The returned size of legacy memory below 1 MB.

Description

This function returns the size of the legacy memory below 1 MB that is required during an S3 resume. Before the Framework-based firmware transfers control to the OS, it has to transition from flat mode into real mode in case the OS supplies only a real-mode waking vector. This transition requires a certain amount of legacy memory below 1 MB. After getting the size of legacy memory below 1 MB, the caller is responsible for allocating the legacy memory below 1 MB according to the size that is returned. The specific implementation of allocating the legacy memory is out of the scope of this specification.

If the returned value of *Size* equals 0, the firmware does not support jumping to a real-mode OS waking vector.

Status Codes Returned

EFI_SUCCESS	Size is successfully returned.
EFI_INVALID_PARAMETER	The pointer *Size* is **NULL**.

EFI_ACPI_S3_SAVE_PROTOCOL.S3Save()

Summary

Prepares all information that is needed in the S3 resume boot path.

Prototype

```
typedef
EFI_STATUS
EFI_BOOTSERVICE
(EFIAPI *EFI_ACPI_S3_SAVE) (
  IN struct _EFI_ACPI_S3_SAVE_PROTOCOL    *This,
  IN VOID                                 *LegacyMemoryAddress
);
```

Parameters

This

A pointer to the **EFI_ACPI_S3_SAVE_PROTOCOL** instance.

LegacyMemoryAddress

The base of legacy memory. It is allocated by the caller according the size returned by the previously call to **GetLegacyMemorySize()**.

Description

This function is used to do the following:

- Prepare all information that is needed in the S3 resume boot path. This information can include the following:
  - Framework boot script table
  - RSDT pointer
  - Reserved memory for the S3 resume
- Get the minimum memory length below 1 MB that is required for the S3 resume boot path.

If *LegacyMemoryAddress* is **NULL**, the firmware will be unable to jump into a real-mode waking vector. However, it might still be able to jump into a flat-mode waking vector as long as the OS provides a flat-mode waking vector. It is the caller's responsibility to ensure the *LegacyMemoryAddress* is valid. If the *LegacyMemoryAddress* is higher than 1 MB, **EFI_INVALID_PARAMETER** will be returned.

This function will store all the information in NVS on a normal boot path. On an S3 boot path, this information will be retrieved to initialize the platform and finally transfer its control to OS. The caller should ensure that all the information is available at that moment. Especially for the Framework boot script table, this function will internally call the **EFI_BOOT_SCRIPT_SAVE_PROTOCOL.CloseTable()** function to get **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** and copy it to NVS automatically. Therefore, the caller should add all necessary boot script records into **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** before calling **CloseTable()**.

See the *Intel® Platform Innovation Framework for EFI Boot Script Specification* for definitions of the following:

- **`EFI_BOOT_SCRIPT_SAVE_PROTOCOL.CloseTable()`**
- **`EFI_ACPI_S3_RESUME_SCRIPT_TABLE`**

Status Codes Returned

EFI_SUCCESS	All information was saved successfully.
EFI_INVALID_PARAMETER	The memory range is not located below 1 MB.
EFI_OUT_OF_RESOURCES	Resources were insufficient to save all the information.
EFI_NOT_FOUND	Some necessary information cannot be found.

S3 Resume PPI

EFI_PEI_S3_RESUME_PPI

Summary

This PPI accomplishes the firmware S3 resume boot path and transfers control to OS.

GUID

```
#define EFI_PEI_S3_RESUME_PPI_GUID  \
{ 0x4426CCB2, 0xE684, 0x4a8a, 0xAE, 0x40, 0x20, 0xD4, 0xB0, 0x25,
0xB7, 0x10}
```

PPI Interface Structure

```
typedef struct _EFI_PEI_S3_RESUME_PPI {
  EFI_PEI_S3_RESUME_PPI_RESTORE_CONFIG        S3RestoreConfig;
} EFI_PEI_S3_RESUME_PPI;
```

Parameters

S3RestoreConfig

Restores the platform to its preboot configuration for an S3 resume and jumps to the OS waking vector. See the **S3RestoreConfig()** function description.

Description

This PPI is published by the S3 resume PEIM and can be used on the S3 resume boot path to restore the platform to its preboot configuration and transfer control to OS. The information that is required for an S3 resume can be saved during the normal boot path using **EFI_ACPI_S3_SAVE_PROTOCOL**. This presaved information can then be restored in the S3 resume boot path using **EFI_PEI_S3_RESUME_PPI**. Architecturally, the S3 resume PEIM is the last PEIM to be dispatched in the S3 resume boot path.

Before using this PPI, the caller has to ensure the necessary information for the S3 resume, such as the following, is available for the S3 resume boot path:

- **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** script table. Type **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** is defined in the *Intel® Platform Innovation Framework for EFI Boot Script Specification.*
- OS waking vector.
- The reserved memory range to be used for the S3 resume.

Otherwise, the S3 resume boot path may fail.

EFI_PEI_S3_RESUME_PPI.S3RestoreConfig()

Summary

Restores the platform to its preboot configuration for an S3 resume and jumps to the OS waking vector.

Prototype

```
typedef
EFI_STATUS
(EFIAPI * EFI_PEI_S3_RESUME_PPI_RESTORE_CONFIG) (
  IN EFI_PEI_SERVICES   **PeiServices
);
```

Parameters

PeiServices

Pointer to the 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).

Description

This function will restore the platform to its preboot configuration that was prestored in **EFI_ACPI_S3_RESUME_SCRIPT_TABLE** and transfer control to OS waking vector.

Upon invocation, this function is responsible for locating the following information before jumping to OS waking vector:

- ACPI table
- S3 resume boot script table
- Any other information that it needs

All this necessary information should have been previously prepared by the **EFI_ACPI_S3_SAVE_PROTOCOL.S3Save()** function on a normal boot path. The **S3RestoreConfig()** function then executes the prestored boot script table by calling **EFI_PEI_BOOT_SCRIPT_EXECUTER_PPI.Execute()** and transitions the platform to the preboot state. Finally, this function transfers control to the OS waking vector. If the OS supports only a real-mode waking vector, this function will switch from flat mode to real mode before jumping to the waking vector.

If all platform preboot configurations are successfully restored and all other necessary information is ready, this function will never return and instead will directly jump to the OS waking vector. If this function returns, it indicates that the attempt to resume from the ACPI S3 sleep state failed.

Status Codes Returned

EFI_ABORTED	Execution of the S3 resume boot script table failed.
EFI_NOT_FOUND	Some necessary information that is used for the S3 resume boot path could not be located.