Intel® Platform Innovation Framework for EFI SMBus PPI Specification

Draft for Review

Version 0.9

April 1, 2004

Intel order number xxxxxx-001

Revision History

Revision	Revision History	Date
0.9	First public release.	4/1/04

Contents

1 Introduction

Overview

This specification defines the core code and services that are required for an implementation of the System Management Bus (SMBus) PEIM-to-PEIM Interface (PPI) of the Intel® Platform Innovation Framework for EFI (hereafter referred to as the "Framework"). This PPI is used by other Pre-EFI Initialization Modules (PEIMs) to control an SMBus host controller.

This specification does the following:

- Describes the basic components of the PEI SMBus PPI
- Provides code definitions for the PEI SMBus PPI and SMBus-related type definitions that are architecturally required by the *Intel® Platform Innovation Framework for EFI Architecture Specification*

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.

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.

Protocol 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

Introduction

This document describes the System Management Bus (SMBus) PEIM-to-PEIM Interface (PPI). This document provides enough material to implement an SMBus Pre-EFI Initialization Module (PEIM) that can control transactions between an SMBus host controller and its slave devices.

The material that is contained in this document is designed to extend the *Intel® Platform Innovation Framework for EFI Architecture Specification* in a way that supports communication via the SMBus. These extensions are provided in the form of SMBus-specific protocols. This document provides the information that is required to implement an SMBus PEIM in the Pre-EFI Initialization (PEI) portion of system firmware.

A full understanding of the *EFI Specification* and the *SMBus Specification* is assumed throughout this document. See Related Information for the URL for the *SMBus Specification.*

Target Audience

This document is intended for the following readers:

- Original equipment manufacturers (OEMs) who will be creating Intel® architecture-based platforms that are intended to boot shrink-wrap operating systems.
- BIOS developers, either those who create general-purpose BIOS and other firmware products, or those who modify these products for use in Intel architecture-based products.
- Operating system developers who will be creating and/or adapting their shrink-wrap operating system products to run on Intel architecture-based platforms.

Related Information

The following publications and sources of information may be useful to you or are referred to by this specification. See Related Information from Intel in the master Framework help system for the URLs for EFI specifications and other documentation from Intel.

Industry Specifications

- *System Management Bus (SMBus) Specification*, version 2.0, SBS Implementer's Forum, August 3, 2000:
  http://www.smbus.org*
- *PCI Local Bus Specification*, revision 2.2, PCI Special Interest Group: See Industry Specifications in the master Framework help system for the URL for specifications from the PCI SIG.

SMBus PPI Terms

The following terms are used throughout this document to describe the model for constructing SMBus PPIs in the PEI environment. See the Glossary in the master Framework help system for explanations of Framework-specific terms.

PEC

Packet Error Code. It is similar to a checksum data of the data coming across the SMBus wire.

PEI

Pre-EFI Initialization.

PEIM

Pre-EFI Initialization Module.

PPI

PEIM-to-PEIM Interface.

SMBus

System Management Bus.

SMBus host controller

Provides a mechanism for the processor to initiate communications with SMBus slave devices. This controller can be connected to a main I/O bus such as PCI.

SMBus master device

Any device that initiates SMBus transactions and drives the clock.

SMBus PPI

A software interface that provides a method to control an SMBus host controller and access the data of the SMBus slave devices that are attached to it.

SMBus slave device

The target of an SMBus transaction, which is driven by some master.

UDID

Unique Device Identifier. A 128-bit value that a device uses during the Address Resolution Protocol (ARP) process to uniquely identify itself.

PEI SMBus PPI Overview

The PEI SMBus PPI is used by code, typically other PEIMs, that is running in the PEI environment to access data on an SMBus slave device via the SMBus host controller. In particular, functions for managing devices on SMBus buses are defined in this specification.

The interfaces that are provided in the **EFI_PEI_SMBUS_PPI** are for performing basic operations to an SMBus slave device. The system provides abstracted access to basic system resources to allow a PEIM to have a programmatic method to access these basic system resources. The main goal of this PPI is to provide an abstraction that simplifies the writing of PEIMs for SMBus slave devices. This goal is accomplished by providing a standard interface to the SMBus slave devices that does not require detailed knowledge about the particular hardware implementation or protocols of the SMBus.

See PEI SMBus PPI in Code Definitions for the definition of **EFI_PEI_SMBUS_PPI**. This PPI is produced by each of the SMBus host controllers in the system.

3
Code Definitions

Introduction

This section contains the basic definitions for PEIMs and SMBus devices to use during the PEI phase. The following PPI is defined in this section:

- **EFI_PEI_SMBUS_PPI**

This section also contains the definitions for additional SMBus-related data types and structures that are subordinate to the structures in which they are called. All of the data structures below except for **EFI_PEI_SMBUS_NOTIFY_FUNCTION** can be used in the DXE phase as well. The following types or structures can be found in "Related Definitions" of the parent function definition:

- **EFI_SMBUS_DEVICE_ADDRESS**
- **EFI_SMBUS_DEVICE_COMMAND**
- **EFI_SMBUS_OPERATION**
- **EFI_SMBUS_UDID**
- **EFI_SMBUS_DEVICE_MAP**
- **EFI_PEI_SMBUS_NOTIFY_FUNCTION**

PEI SMBus PPI

EFI_PEI_SMBUS_PPI

Summary

Provides the basic I/O interfaces that a PEIM uses to access its SMBus controller and the slave devices attached to it.

GUID

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

PPI Interface Structure

```
typedef struct _EFI_PEI_SMBUS_PPI {
  EFI_PEI_SMBUS_PPI_EXECUTE_OPERATION  Execute;
  EFI_PEI_SMBUS_PPI_ARP_DEVICE         ArpDevice;
  EFI_PEI_SMBUS_PPI_GET_ARP_MAP        GetArpMap;
  EFI_PEI_SMBUS_PPI_NOTIFY             Notify;
} EFI_PEI_SMBUS_PPI;
```

Parameters

Execute

Executes the SMBus operation to an SMBus slave device. See the **Execute()** function description.

ArpDevice

Allows an SMBus 2.0 device(s) to be Address Resolution Protocol (ARP). See the **ArpDevice()** function description.

GetArpMap

Allows a PEIM to retrieve the address that was allocated by the SMBus host controller during enumeration/ARP. See the **GetArpMap()** function description.

Notify

Allows a driver to register for a callback to the SMBus host controller driver when the bus issues a notification to the bus controller PEIM. See the **Notify()** function description.

Description

The **EFI_PEI_SMBUS_PPI** provides the basic I/O interfaces that are used to abstract accesses to SMBus host controllers. There is one **EFI_PEI_SMBUS_PPI** instance for each SMBus host controller in a system. A PEIM that wishes to manage an SMBus slave device in a system will have to retrieve the **EFI_PEI_SMBUS_PPI** instance that is associated with its SMBus host controller.

EFI_PEI_SMBUS_PPI.Execute()

Summary

Executes an SMBus operation to an SMBus controller. Returns when either the command has been executed or an error is encountered in doing the operation.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_PEI_SMBUS_PPI_EXECUTE_OPERATION) (
  IN     EFI_PEI_SERVICES               **PeiServices,
  IN     EFI_PEI_SMBUS_PPI              *This,
  IN     EFI_SMBUS_DEVICE_ADDRESS       SlaveAddress,
  IN     EFI_SMBUS_DEVICE_COMMAND       Command,
  IN     EFI_SMBUS_OPERATION            Operation,
  IN     BOOLEAN                        PecCheck,
  IN OUT UINTN                          *Length,
  IN OUT VOID                           *Buffer
);
```

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_SMBUS_PPI** instance.

SlaveAddress

The SMBUS hardware address to which the SMBUS device is preassigned or allocated. Type **EFI_SMBUS_DEVICE_ADDRESS** is defined in "Related Definitions" below.

Command

This command is transmitted by the SMBus host controller to the SMBus slave device and the interpretation is SMBus slave device specific. It can mean the offset to a list of functions inside an SMBus slave device. Not all operations or slave devices support this command's registers. Type **EFI_SMBUS_DEVICE_COMMAND** is defined in "Related Definitions" below.

Operation

Signifies which particular SMBus hardware protocol instance that it will use to execute the SMBus transactions. This SMBus hardware protocol is defined by the *SMBus Specification* and is not related to EFI. Type **EFI_SMBUS_OPERATION** is defined in "Related Definitions" below.

PecCheck

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

Length

Signifies the number of bytes that this operation will do. The maximum number of bytes can be revision specific and operation specific. This parameter will contain the actual number of bytes that are executed for this operation. Not all operations require this argument.

Buffer

Contains the value of data to execute to the SMBus slave device. Not all operations require this argument. The length of this buffer is identified by *Length*.

Description

The **Execute()** function provides a standard way to execute an operation as defined in the *System Management Bus (SMBus) Specification*. The resulting transaction will be either that the SMBus slave devices accept this transaction or that this function returns with error.

Related Definitions

```
//*******************************************************
// EFI_SMBUS_DEVICE_ADDRESS
//*******************************************************
typedef struct _EFI_SMBUS_DEVICE_ADDRESS {
  UINTN   SmbusDeviceAddress:7;
} EFI_SMBUS_DEVICE_ADDRESS;
```

SmbusDeviceAddress

The SMBUS hardware address to which the SMBUS device is preassigned or allocated.

```
//*******************************************************
// EFI_SMBUS_DEVICE_COMMAND
//*******************************************************
typedef UINTN EFI_SMBUS_DEVICE_COMMAND;
```

```
//*******************************************************
// EFI_SMBUS_OPERATION
//*******************************************************
typedef enum _EFI_SMBUS_OPERATION {
  EfiSmbusQuickRead,
  EfiSmbusQuickWrite,
  EfiSmbusReceiveByte,
  EfiSmbusSendByte,
  EfiSmbusReadByte,
  EfiSmbusWriteByte,
  EfiSmbusReadWord,
  EfiSmbusWriteWord,
  EfiSmbusReadBlock,
  EfiSmbusWriteBlock,
  EfiSmbusProcessCall,
  EfiSmbusBWBRProcessCall
} EFI_SMBUS_OPERATION;
```

See the *SMBus Specification* for descriptions of the fields in the above definition.

Status Codes Returned

EFI_SUCCESS	The last data that was returned from the access matched the poll exit criteria.
EFI_CRC_ERROR	The checksum is not correct (PEC is incorrect)
EFI_TIMEOUT	*Timeout* expired before the operation was completed. *Timeout* is determined by the SMBus host controller device.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
EFI_DEVICE_ERROR	The request was not completed because a failure reflected in the Host Status Register bit. Device errors are a result of a transaction collision, illegal command field, unclaimed cycle (host initiated), or bus errors (collisions).
EFI_INVALID_PARAMETER	*Operation* is not defined in **EFI_SMBUS_OPERATION**.
EFI_INVALID_PARAMETER	*Length*/*Buffer* is **NULL** for operations except for **EfiSmbusQuickRead** and **EfiSmbusQuickWrite**. *Length* is outside the range of valid values.
EFI_UNSUPPORTED	The SMBus operation or PEC is not supported.
EFI_BUFFER_TOO_SMALL	*Buffer* is not sufficient for this operation.

EFI_PEI_SMBUS_PPI.ArpDevice()

Summary

Sets the SMBus slave device addresses for the device with a given unique ID or enumerates the entire bus.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_PEI_SMBUS_PPI_ARP_DEVICE) (
  IN     EFI_PEI_SERVICES               **PeiServices,
  IN     EFI_PEI_SMBUS_PPI              *This,
  IN     BOOLEAN                        ArpAll,
  IN     EFI_SMBUS_UDID                 *SmbusUdid, OPTIONAL
  IN OUT EFI_SMBUS_DEVICE_ADDRESS       *SlaveAddress  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_SMBUS_PPI** instance.

ArpAll

A Boolean expression that indicates if the host drivers need to enumerate all the devices or enumerate only the device that is identified by *SmbusUdid*. If *ArpAll* is **TRUE**, *SmbusUdid* and *SlaveAddress* are optional. If *ArpAll* is **FALSE**, *ArpDevice* will enumerate *SmbusUdid* and the address will be at *SlaveAddress*.

SmbusUdid

The targeted SMBus Unique Device Identifier (UDID). The UDID may not exist for SMBus devices with fixed addresses. Type **EFI_SMBUS_UDID** is defined in "Related Definitions" below.

SlaveAddress

The new SMBus address for the slave device for which the operation is targeted. Type **EFI_SMBUS_DEVICE_ADDRESS** is defined in **EFI_PEI_SMBUS_PPI.Execute()**.

Description

The **ArpDevice()** function enumerates the entire bus or enumerates a specific device that is identified by *SmbusUdid*.

Related Definitions

```
//*******************************************************
// EFI_SMBUS_UDID
//*******************************************************
typedef struct _EFI_SMBUS_UDID {
  UINT32 VendorSpecificId;
  UINT16 SubsystemDeviceId;
  UINT16 SubsystemVendorId;
  UINT16 Interface;
  UINT16 DeviceId;
  UINT16 VendorId;
  UINT8  VendorRevision;
  UINT8  DeviceCapabilities;
} EFI_SMBUS_UDID;
```

VendorSpecificId

A unique number per device.

SubsystemDeviceId

Identifies a specific interface, implementation, or device. The subsystem ID is defined by the party that is identified by the *SubsystemVendorId* field.

SubsystemVendorId

This field may hold a value that is derived from any of several sources:

- The device manufacturer’s ID as assigned by the SBS Implementer's Forum or the PCI SIG.
- The device OEM’s ID as assigned by the SBS Implementer's Forum or the PCI SIG.
- A value that, in combination with the *SubsystemDeviceId*, can be used to identify an organization or industry group that has defined a particular common device interface specification.

Interface

Identifies the protocol layer interfaces that are supported over the SMBus connection by the device. For example, Alert Standard Format (ASF) and IPMI.

DeviceId

The device ID as assigned by the device manufacturer (identified by the *VendorId* field).

VendorId

The device manufacturer’s ID as assigned by the SBS Implementer's Forum or the PCI SIG.

VendorRevision

UDID version number and a silicon revision identification.

DeviceCapabilities

Describes the device’s capabilities.

Status Codes Returned

EFI_SUCCESS	The SMBus slave device address was set.
EFI_INVALID_PARAMETER	*SlaveAddress* is **NULL**.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
EFI_TIMEOUT	The SMBus slave device did not respond.
EFI_DEVICE_ERROR	The request was not completed because the transaction failed. Device errors are a result of a transaction collision, illegal command field, or unclaimed cycle (host initiated).

EFI_PEI_SMBUS_PPI.GetArpMap()

Summary

Returns a pointer to the Address Resolution Protocol (ARP) map that contains the ID/address pair of the slave devices that were enumerated by the SMBus host controller driver.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_PEI_SMBUS_PPI_GET_ARP_MAP) (
  IN     EFI_PEI_SERVICES              **PeiServices,
  IN     EFI_PEI_SMBUS_PPI             *This,
  IN OUT UINTN                         *Length,
  IN OUT EFI_SMBUS_DEVICE_MAP          **SmbusDeviceMap
);
```

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_SMBUS_PPI** instance.

Length

Size of the buffer that contains the SMBus device map.

SmbusDeviceMap

The pointer to the device map as enumerated by the SMBus controller driver. Type **EFI_SMBUS_DEVICE_MAP** is defined in "Related Definitions" below.

Description

The **GetArpMap()** function returns the mapping of all the SMBus devices that are enumerated by the SMBus host driver.

Related Definitions

```
//*******************************************************
// EFI_SMBUS_DEVICE_MAP
//*******************************************************
typedef struct _EFI_SMBUS_DEVICE_MAP {
  EFI_SMBUS_DEVICE_ADDRESS  SmbusDeviceAddress;
  EFI_SMBUS_UDID            SmbusDeviceUdid;
} EFI_SMBUS_DEVICE_MAP;
```

SmbusDeviceAddress

The SMBUS hardware address to which the SMBUS device is preassigned or allocated. Type **EFI_SMBUS_DEVICE_ADDRESS** is defined in **EFI_PEI_SMBUS_PPI.Execute()**.

SmbusDeviceUdid

The SMBUS Unique Device Identifier (UDID) as defined in **EFI_SMBUS_UDID**. Type **EFI_SMBUS_UDID** is defined in **EFI_PEI_SMBUS_PPI.ArpDevice()**.

Status Codes Returned

EFI_SUCCESS	The device map was returned correctly in the buffer.

EFI_PEI_SMBUS_PPI.Notify()

Summary

Allows a device driver to register for a callback when the bus driver detects a state that it needs to propagate to other PEIMs that are registered for a callback.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_PEI_SMBUS_PPI_NOTIFY) (
  IN     EFI_PEI_SERVICES                **PeiServices,
  IN     EFI_PEI_SMBUS_PPI               *This,
  IN     EFI_SMBUS_DEVICE_ADDRESS        SlaveAddress,
  IN     UINTN                           Data,
  IN     EFI_PEI_SMBUS_NOTIFY_FUNCTION   NotifyFunction
);
```

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_SMBUS_PPI** instance.

SlaveAddress

Address that the host controller detects as sending a message and calls all the registered functions. Type **EFI_SMBUS_DEVICE_ADDRESS** is defined in **EFI_PEI_SMBUS_PPI.Execute()**.

Data

Data that the host controller detects as sending a message and calls all the registered functions.

NotifyFunction

The function to call when the bus driver detects the *SlaveAddress* and *Data* pair. Type **EFI_PEI_SMBUS_NOTIFY_FUNCTION** is defined in "Related Definitions" below.

Description

The **Notify()** function registers all the callback functions to allow the bus driver to call these functions when the *SlaveAddress*/*Data* pair happens.

Related Definitions

```
//*******************************************************
// EFI_PEI_SMBUS_NOTIFY_FUNCTION
//*******************************************************
typedef
EFI_STATUS
(EFIAPI *EFI_PEI_SMBUS_NOTIFY_FUNCTION) (
  IN      EFI_PEI_SERVICES            **PeiServices,
  IN      EFI_PEI_SMBUS_PPI           *SmbusPpi,
  IN      EFI_SMBUS_DEVICE_ADDRESS    SlaveAddress,
  IN      UINTN                       Data
);
```

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).

SmbusPpi

A pointer to the **EFI_PEI_SMBUS_PPI** instance.

SlaveAddress

The SMBUS hardware address to which the SMBUS device is preassigned or allocated. Type **EFI_SMBUS_DEVICE_ADDRESS** is defined in **EFI_PEI_SMBUS_PPI.Execute()**.

Data

Data of the SMBus host notify command that the caller wants to be called.

Status Codes Returned

EFI_SUCCESS	*NotifyFunction* has been registered.