Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide

Draft for Review

Version 0.9

April 1, 2004

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 design guidelines that are required for an implementation of a new data hub subclass in the Intel® Platform Innovation Framework for EFI (hereafter referred to as the "Framework"). This specification does the following:

- Describes basic interactions with the data hub
- Defines the rules and guidelines for creating a new data hub subclass
- Provides code definitions for the data record subclass header and common macros that are architecturally required by the *Intel® Platform Innovation Framework for EFI Architecture Specification*

This specification complies with the *System Management BIOS Reference Specification*, version 2.3.4.

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.

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

Overview

This design guide serves as a starting document for a new subclass document. It describes elements of data records that belong to a certain subclass and provides design guidelines when designing a new data hub record. This document will reference material in the *Intel® Platform Innovation Framework for EFI Data Hub Specification*. Not all data subclasses need to adhere to this design guide.

The following definitions are included in this design guide:

- Header information
- Common macro definition
- Syntax

All of the above definitions are common to all data records in the data class. Definitions that are specific to a subclass or to a specific data record will not be included in this specification. See Code Definitions for the definitions listed above.

Interactions with the Data Hub

As a repository of data, the data hub interacts with two types of agents:

- Data producers
- Data consumers

The data hub itself does not examine or otherwise interpret the deposited contents of producer-specific data. It merely provides storage, notification, and retrieval services to producers and consumers of data.

Records in the data hub constitute a database. Particular consumer interpretation—such as System Management BIOS (SMBIOS), for example—could be regarded as a specialized view of the database.

Example producers are device drivers depositing information about hardware—for example, a processor driver depositing information about system processors, a memory driver depositing information about system main RAM physical and logical topology and other provisions, and so on.

The following are example consumers:

- SMBIOS
- Setup
- Shell commands
- System management
- Other components and utilities

Rules and Guidelines

Rules

The following rules apply when creating a new data record or data subclass:

- A system is **not** required to have all the data records (`RecordType`) defined in a subclass specification.
- A certain record number can be declared multiple times because the header information will determine to which `Instance`, `SubInstance`, and `ProducerName` the data record refers.
- The data producer can describe the same data record (the same `RecordType`, `Instance`, `SubInstance`, and `ProducerName`) more than once if the previous data record needs to be updated.

The data hub driver will log all the records. The consumers need to consider that data producers are allowed to create a new data record with the same header information to update the previous data record.

Record Design Guidelines

Record Design Guidelines

This section provides guidelines that must be followed when designing a new data hub record. These guidelines apply retroactively. If existing code does not comply with the guidelines, the code must be changed to bring it into compliance.

NOTE

Not following these guidelines may result in unknown behavior.

Alignment

Fields in a data hub record should be aligned at their natural boundaries. For example, a field of type **`UINT64`** must be aligned at 8 bytes. Arrays of elements must be aligned only at a single element size. For example, a Unicode string with a character size of 16 bits must be aligned at 2 bytes.

This alignment could be achieved by properly arranging the fields' locations within the structure and/or by padding larger fields.

Provided that no packing is in effect (the **`#pragma pack()`** directive in C), the compiler might satisfactory align structured fields.

Note that this requirement precludes designing records that map one-to-one to some SMBIOS types. In such a case, the SMBIOS code would translate the data hub record to its own specific structures.

Content and Structure

The amount and granularity of information presented by a producer should be a sum of the data needs of all known, and possibly anticipated, consumers. The information may be contained in a single record or in multiple records. It is the consumer's responsibility to extract the correct record or set of records that will satisfy the consumer's needs for information.

It is possible and allowable that, while observing these guidelines, a data hub record structure can be designed to be one-to-one—i.e., identical, with some specific consumer structure. In such cases, the consumer could copy the data directly to its specific structures, and it is permissible that some of the data structures may look like a "copy" from other specifications.

Location of Record Declaration Files

Data hub data record structures should be self-contained and declared in declaration files (in *.h* files in the case of the C language) that are separate from producers and consumers.

These files should not be part of the consumer or producer code. They should be available at build time to all potential consumers, even if no relevant producer is part of, or even available at, the build time.

At the time of this document, the following is the location for these files: **`DXE\Guid\DataHubRecords`**.

At boot services runtime, a consumer would need to handle a potential situation where the data hub does not contain the requested records.

Conversely, the producer should not rely on any consumer code, yet it should be able to build and run without any consumer code being available at build and run times.

For example, a properly defined data hub record will not contain any references to the SMBIOS declarations.

3 Code Definitions

Introduction

This section contains the basic definitions of the data hub subclass and macros that are common to all data hub records. The following data types and macros are defined in this section:

- `EFI_SUBCLASS_TYPE1_HEADER`
- `EFI_EXP_BASE10_DATA`
- `EFI_EXP_BASE2_DATA`
- `EFI_INTER_LINK_DATA`

Subclass Header

EFI_SUBCLASS_TYPE1_HEADER

Summary

Each data record that is a member of some subclass starts with a standard header of type **EFI_SUBCLASS_TYPE1_HEADER**.

Prototype

```
typedef struct {
  UINT32      Version;
  UINT32      HeaderSize;
  UINT16      Instance;
  UINT16      SubInstance;
  UINT32      RecordType;
} EFI_SUBCLASS_TYPE1_HEADER;
```

Parameters

Version

The version of the specification to which a specific subclass data record adheres.

HeaderSize

The size in bytes of this data class header.

Instance

The instance number of the subclass with the same *ProducerName*. This number is applicable in cases where multiple subclass instances that were produced by the same driver exist in the system. This entry is 1 based; 0 means *Reserved* and -1 means *Not Applicable.* All data consumer drivers should be able to handle all the possible values of *Instance*, including *Not Applicable* and *Reserved*.

SubInstance

The instance number of the *RecordType* for the same *Instance*. This number is applicable in cases where multiple instances of the *RecordType* exist for a specific *Instance*. This entry is 1 based; 0 means *Reserved* and -1 means *Not Applicable.* All data consumer drivers should be able to handle all the possible values of *SubInstance*, including *Not Applicable* and *Reserved*.

RecordType

The record number for the data record being specified. The numbering scheme and definition is defined in the specific subclass specification.

Description

Each data record that is a member of some subclass starts with a standard header of type **`EFI_SUBCLASS_TYPE1_HEADER`**. This header is only a guideline and applicable only to a data subclass that is producing SMBIOS data records. A subclass can start with a different header if needed.

`Instance` and *`SubInstance`* may not be applicable in some cases. Also, certain *`RecordType`*s may not be applicable for specific implementations of a subclass.

Macros

Macros

The following definitions are common among data records and are defined here for reference.

EFI_EXP_BASE10_DATA

Summary

Provides a calculation for base-10 representations.

Prototype

```
typedef struct {
  INT16      Value;
  INT16      Exponent;
} EFI_EXP_BASE10_DATA;
```

Parameters

Value

The **INT16** number by which to multiply the base-10 representation. For example, if **Value = 3**, the calculation would be (**3 * 10^*Exponent***).

Exponent

The **INT16** number by which to raise the base-10 calculation. For example, when used with this structure:

- **Exponent = 3** means 10^3 (kilo).
- **Exponent = 6** means 10^6 (mega).
- **Exponent = -3** means 10^{-3} (milli).

Description

This macro provides a calculation for base-10 representations. *Value* and *Exponent* are each **INT16**. It is signed to cover negative values and is 16 bits wide (15 bits for data and 1 bit for the sign).

EFI_EXP_BASE2_DATA

Summary

Provides a calculation for base-2 representations.

Prototype

```
typedef struct {
  UINT16     Value;
  UINT16     Exponent;
} EFI_EXP_BASE2_DATA;
```

Parameters

Value

The **INT16** number by which to multiply the base-2 representation. For example, if **Value = 3**, the calculation would be (**3 * 2^***Exponent*).

Exponent

The **INT16** number by which to raise the base-2 calculation. For example, when used with this structure:

- **Exponent = 10** means 2^{10} (1024, kilo).
- **Exponent = 20** means 2^{20} (1048576, mega).

Description

This macro provides a calculation for base-2 representations. *Value* and *Exponent* are each **INT16**. It is 16 bits wide and is unsigned to mean nonnegative values.

EFI_INTER_LINK_DATA

Summary

Links data records in the same subclass.

Prototype

```
typedef struct {
  EFI_GUID        ProducerName;
  UINT16          Instance;
  UINT16          SubInstance;
} EFI_INTER_LINK_DATA;
```

Parameters

ProducerName

An **EFI_GUID** that identifies the component that produced this data record. Type **EFI_GUID** is defined in **InstallProtocolInterface()** in the *EFI 1.10 Specification.*

Instance

The instance number of the subclass with the same *ProducerName*. This number is applicable in cases where multiple subclass instances that were produced by the same driver exist in the system. This entry is 1 based; 0 means *Reserved* and -1 means *Not Applicable*. All data consumer drivers should be able to handle all the possible values of *Instance*, including *Not Applicable* and *Reserved*.

SubInstance

The instance number of the *RecordType* for the same *Instance*. This number is applicable in cases where multiple instances of the *RecordType* exist for a specific *Instance*. This entry is 1 based; 0 means *Reserved* and -1 means *Not Applicable*. All data consumer drivers should be able to handle all the possible values of *SubInstance*, including *Not Applicable* and *Reserved*.

Description

This structure is used to link data records in the same subclasses. A data record is defined as a link to another data record in the same subclass using this structure.

STRING_REF

Summary

Defines the string to be manipulated.

Description

A string reference (**`STRING_REF`**) is a **`UINT16`** value defining a string to be manipulated. See the *Intel® Platform Innovation Framework for EFI Human Interface Infrastructure Specification* for the definition.