Intel® Platform Innovation Framework for EFI Memory Subclass 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 memory data hub subclass of the Intel® Platform Innovation Framework for EFI (hereafter referred to as the "Framework"). This specification does the following:

- Describes the basic components of the memory data hub subclass and memory subclass data records
- Provides code definitions for type and record definitions for the memory subclass that are architecturally required by the *Intel® Platform Innovation Framework for EFI Architecture Specification*
- Provides examples of the data records for the memory subclass

This specification complies with the *System Management BIOS (SMBIOS) 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 specification describes a group of data records that have similar characteristics. The data records are records that will be input to the data hub to be consumed by one or more drivers.

This specification complies with the *Intel® Platform Innovation Framework for EFI Data Hub Specification* and it is assumed that the consumer of this specification is well versed in the concept of the data hub. This specification describes in more detail how to implement a driver that will log all the non-cache-memory-related data records and how to create drivers that will consume this data.

This specification also specifies the format of each data record. Each data record must be capable of being used by current and potential consumers of the data—in other words, the format should also be consumable by all potential agents. The unit of measurement, if applicable, should be the most common unit of measurement for the specific data record, and the range of values for a data record should be usable for the foreseeable future.

Scope

This specification is the contract for non-cache-memory-related data between the non-cache-memory data drivers and the non-cache-memory data consumers. The data referred to here is not the data contained in the memory but the data that describes the memory characteristics. This specification covers all data structures that are memory related, excluding caches, and includes all the different memory levels in the system. For multiprocessor (MP) systems, it includes all the data related to all processors in the system.

The data driver (memory driver in this case) does not have to declare all the record types listed in this specification but should declare only those data types that are applicable. If the data is not applicable (for example, if no error detection/correction memory exists in the system), the data consumer should make the necessary adjustment and not declare the associated data records (memory error information, etc.).

A specific record defines the semantic context of the data. All records related to memory are documented in this specification. The record type is numbered sequentially and a new record type can be appended to the end of the list. The record type in this specification defines the semantic context of the data. The data records in this specification comply to the **`EFI_SUBCLASS_TYPE1_HEADER`**.*`Version`* field of 1. Type **`EFI_SUBCLASS_TYPE1_HEADER`** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

Data records can be added or deleted as long as backward compatibility is maintained. If a data record needs to be updated, a new entry to this specification can be added. At some point when the objectives of this subclass are not accomplished or become inconsistent, an entirely new subclass with a new Globally Unique Identifier (GUID) will be introduced.

NOTE

The data records and structures sometimes are System Management BIOS (SMBIOS)–centric as the initial document is based on the SMBIOS requirements.

Error Data

This document does not list any memory error data. That information is in the *Intel® Platform Innovation Framework for EFI Status Codes Specification*.

Consumer versus Producer

It is possible that a producer of memory data is also a consumer and vice versa. An example is memory subunit comprised of multiple memory arrays. The producer of the subunit consumes the memory array information.

Compliancy Requirements

None of the record numbers described in this document are required by the *Extensible Firmware Interface Specification.* Some record number entries may be required by other industry-wide specifications such as the *System Management BIOS (SMBIOS) Reference Specification.*

Some **ENUM** definitions are controlled by the Distributed Management Task Force, Inc. (DMTF) organization. Refer to their Web site at www.dmtf.org/standards/dmi* and select the link to *Master MIF*.

Header Information

Memory Subclass Definition

The memory subclass belongs to the data class and is identified as the memory subclass by the GUID. See Memory Subclass Definition in Code Definitions for the definition.

Data Record Header

Each data record that is logged or read starts with a standard header of type **EFI_DATA_RECORD_HEADER**. The format of the header is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Specification*.

Data Class Header

Each data record that is a member of the data class starts with a standard header of type **EFI_SUBCLASS_TYPE1_HEADER**. The format of the header is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*. This header follows the data record header **EFI_DATA_RECORD_HEADER**, which is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Specification*.

The *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide* provides generic descriptions of the fields in **EFI_SUBCLASS_TYPE1_HEADER**. The following explanation further clarifies the descriptions for the memory subclass:

- The *Instance* is the instance number of memory in the system, as multiple memory subsystems can exist in the system.
- The *SubInstance* is the physical memory hierarchy level in the memory subsystem, as multiple physical memory hierarchy levels can exist in a memory subsystem.

Raw Data

The raw data follows the **EFI_SUBCLASS_TYPE1_HEADER** header and its definition is specific to the *RecordNumber*. The syntax of the raw data is defined in Data Record Number in Code Definitions. Type **EFI_SUBCLASS_TYPE1_HEADER** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

Data Record Number

The **EFI_SUBCLASS_TYPE1_HEADER** is followed by a data record. The data record format is specific to a *RecordNumber*. The sections in Data Record Number in Code Definitions define the data *RecordNumber*s for the memory subclass. The *RecordNumber*s are subdivided into different subsections on a per-SMBIOS type for ease of reference. See the Examples section for examples of these record numbers.

The **EFI_INTER_LINK_DATA** structure is used to link together multiple data hub items of the same subclass. An example is a memory array that is composed of memory modules. A possible use would be *Instance* for a memory region or array and *SubInstance* for a memory slot.

All generic macros are defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*. **STRING_REF** is defined in the *Intel® Platform Innovation Framework for EFI Human Interface Infrastructure Specification*.

3
Code Definitions

Introduction

This section contains the basic definitions of the data record header fields that are specific to the memory subclass, as well as definitions of memory data records. The following data types and data records are defined in this section:

- **EFI_MEMORY_SUBCLASS**
- **EFI_MEMORY_SUBCLASS_VERSION**
- **EFI_MEMORY_SIZE_DATA**
- **EFI_MEMORY_ARRAY_LOCATION_DATA**
- **EFI_MEMORY_ARRAY_LINK_DATA**
- **EFI_MEMORY_ARRAY_START_ADDRESS_DATA**
- **EFI_MEMORY_DEVICE_START_ADDRESS_DATA**
- **EFI_MEMORY_CHANNEL_TYPE_DATA**
- **EFI_MEMORY_CHANNEL_DEVICE_DATA**

See the Examples section for examples using these data records.

Header Information

Memory Subclass Definition

EFI_MEMORY_SUBCLASS

Summary

The memory subclass belongs to the data class and is identified as the memory subclass by the GUID.

GUID

```
#define EFI_MEMORY_SUBCLASS_GUID \
{0x4E8F4EBB, 0x64B9, 0x4E05, 0x9B, 0x18, 0x4C, 0xFE, 0x49, 0x23,
0x50, 0x97}
```

Class

```
#define        EFI_MEMORY_SUBCLASS        EFI_DATA_CLASS_DATA
```

Description

Summary

The memory subclass belongs to the data class and is identified as the memory subclass by the GUID.

For this subclass, the values defined above are used as follows:

- **EFI_DATA_RECORD_HEADER**.*DataRecordGuid* = **EFI_MEMORY_SUBCLASS_GUID**, which is the GUID that is specific to the memory subclass.
- **EFI_DATA_RECORD_HEADER**.*DataRecordClass* = **EFI_DATA_CLASS_DATA**. The "class" may be equal to the GUID "class" or a superset of the GUID "class."

Type **EFI_DATA_CLASS_DATA** is defined in **EFI_DATA_RECORD_HEADER**, which is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Specification*.

Subclass Version

EFI_MEMORY_SUBCLASS_VERSION

Summary

Indicates the version of the memory subclass.

Prototype

```
#define EFI_MEMORY_SUBCLASS_VERSION   0x0100
```

Description

This value indicates the version of the memory subclass. It is used in **EFI_SUBCLASS_TYPE1_HEADER**.*Version*. Type **EFI_SUBCLASS_TYPE1_HEADER** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

Data Record Number

Legacy Memory

Legacy Memory

This section describes memory information from an operating system's point of view and is not part of the SMBIOS specification.

Memory Region Size

EFI_MEMORY_SIZE_DATA

Summary

This data record refers to the size of a memory region.

Prototype

```
typedef struct {
  UINT32                    ProcessorNumber;
  UINT16                    StartBusNumber;
  UINT16                    EndBusNumber;
  EFI_MEMORY_REGION_TYPE    MemoryRegionType;
  EFI_EXP_BASE2_DATA        MemorySize;
  EFI_PHYSICAL_ADDRESS      MemoryStartAddress;
} EFI_MEMORY_SIZE_DATA;
```

Parameters

ProcessorNumber

A zero-based value that indicates which processor(s) can access the memory region. A value of 0xFFFF indicates the region is accessible by all processors.

StartBusNumber

A zero-based value that indicates the starting bus that can access the memory region.

EndBusNumber

A zero-based value that indicates the ending bus that can access the memory region. A value of 0xFF for a PCI system indicates the region is accessible by all buses and is global in scope. An example of the *EndBusNumber* not being 0xFF is a system with two or more peer-to-host PCI bridges.

MemoryRegionType

The type of memory region from the operating system's point of view. *MemoryRegionType* values are equivalent to the legacy INT 15 AX = E820 BIOS command values. Type **EFI_MEMORY_REGION_TYPE** is defined in "Related Definitions" below.

MemorySize

The size of the memory region in bytes. Type **EFI_EXP_BASE2_DATA** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

MemoryStartAddress

The starting physical address of the memory region. Type **EFI_PHYSICAL_ADDRESS** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

Description

This data record refers to the size of a memory region. The regions that are described can refer to physical memory, memory-mapped I/O, or reserved BIOS memory regions. The unit of measurement of this data record is in bytes.

For this data record, **EFI_SUBCLASS_TYPE1_HEADER**.*RecordType* = **EFI_MEMORY_SIZE_RECORD_NUMBER**. Type **EFI_SUBCLASS_TYPE1_HEADER** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

Related Definitions

```
//*************************************************************
// Record number
//*************************************************************
#define EFI_MEMORY_SIZE_RECORD_NUMBER              0x00000001


//*************************************************************
// EFI_MEMORY_REGION_TYPE
//*************************************************************
typedef enum _EFI_MEMORY_REGION_TYPE {
  EfiMemoryRegionMemory                    = 0x01,
  EfiMemoryRegionReserved                  = 0x02,
  EfiMemoryRegionAcpi                      = 0x03,
  EfiMemoryRegionNvs                       = 0x04
} EFI_MEMORY_REGION_TYPE;
```

Physical Memory Array (Type 16)

EFI_MEMORY_ARRAY_LOCATION_DATA

Summary

This data record refers to the physical memory array.

Prototype

```
typedef struct {
  EFI_MEMORY_ARRAY_LOCATION       MemoryArrayLocation;
  EFI_MEMORY_ARRAY_USE            MemoryArrayUse;
  EFI_MEMORY_ERROR_CORRECTION     MemoryErrorCorrection;
  EFI_EXP_BASE2_DATA              MaximumMemoryCapacity;
  UINT16                          NumberMemoryDevices;
} EFI_MEMORY_ARRAY_LOCATION_DATA;
```

Parameters

MemoryArrayLocation

The physical location of the memory array. Type **EFI_MEMORY_ARRAY_LOCATION** is defined in "Related Definitions" below.

MemoryArrayUse

The memory array usage. Type **EFI_MEMORY_ARRAY_USE** is defined in "Related Definitions" below.

MemoryErrorCorrection

The primary error correction or detection supported by this memory array. Type **EFI_MEMORY_ERROR_CORRECTION** is defined in "Related Definitions" below.

MaximumMemoryCapacity

The maximum memory capacity size in kilobytes. If capacity is unknown, then values of *MaximumMemoryCapacity.Value* **= 0x00** and *MaximumMemoryCapacity.Exponent* **= 0x8000** are used. Type **EFI_EXP_BASE2_DATA** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

NumberMemoryDevices

The number of memory slots or sockets that are available for memory devices in this array.

Description

This data record refers to the physical memory array. This data record is a structure.

The type definition structure for **EFI_MEMORY_ARRAY_LOCATION_DATA** is in SMBIOS 2.3.4:

- Table 3.3.17.1, Type 16, Offset 0x4
- Table 3.3.17.2, Type 16, Offset 0x5
- Table 3.3.17.3, Type 16, with the following offsets:
  - Offset 0x6
  - Offset 0x7
  - Offset 0xB
  - Offset 0xD

For this data record, **EFI_SUBCLASS_TYPE1_HEADER**.*RecordType* = **EFI_MEMORY_ARRAY_LOCATION_RECORD_NUMBER**. Type **EFI_SUBCLASS_TYPE1_HEADER** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

Related Definitions

```
//*************************************************************
// Record number
//*************************************************************
#define EFI_MEMORY_ARRAY_LOCATION_RECORD_NUMBER    0x00000002


//*************************************************************
// EFI_MEMORY_ARRAY_LOCATION
//*************************************************************
typedef enum _EFI_MEMORY_ARRAY_LOCATION {
  EfiMemoryArrayLocationOther                = 0x01,
  EfiMemoryArrayLocationUnknown              = 0x02,
  EfiMemoryArrayLocationSystemBoard          = 0x03,
  EfiMemoryArrayLocationIsaAddonCard         = 0x04,
  EfiMemoryArrayLocationEisaAddonCard        = 0x05,
  EfiMemoryArrayLocationPciAddonCard         = 0x06,
  EfiMemoryArrayLocationMcaAddonCard         = 0x07,
  EfiMemoryArrayLocationPcmciaAddonCard      = 0x08,
  EfiMemoryArrayLocationProprietaryAddonCard = 0x09,
  EfiMemoryArrayLocationNuBus                = 0x0A,
  EfiMemoryArrayLocationPc98C20AddonCard     = 0xA0,
  EfiMemoryArrayLocationPc98C24AddonCard     = 0xA1,
  EfiMemoryArrayLocationPc98EAddonCard       = 0xA2,
  EfiMemoryArrayLocationPc98LocalBusAddonCard = 0xA3
} EFI_MEMORY_ARRAY_LOCATION;
```

```
//***************************************************************
// EFI_MEMORY_ARRAY_USE
//***************************************************************
typedef enum _EFI_MEMORY_ARRAY_USE {
  EfiMemoryArrayUseOther                = 0x01,
  EfiMemoryArrayUseUnknown              = 0x02,
  EfiMemoryArrayUseSystemMemory         = 0x03,
  EfiMemoryArrayUseVideoMemory          = 0x04,
  EfiMemoryArrayUseFlashMemory          = 0x05,
  EfiMemoryArrayUseNonVolatileRam       = 0x06,
  EfiMemoryArrayUseCacheMemory          = 0x07,
} EFI_MEMORY_ARRAY_USE;


//***************************************************************
// EFI_MEMORY_ERROR_CORRECTION
//***************************************************************
typedef enum _EFI_MEMORY_ERROR_CORRECTION {
  EfiMemoryErrorCorrectionOther         = 0x01,
  EfiMemoryErrorCorrectionUnknown       = 0x02,
  EfiMemoryErrorCorrectionNone          = 0x03,
  EfiMemoryErrorCorrectionParity        = 0x04,
  EfiMemoryErrorCorrectionSingleBitEcc  = 0x05,
  EfiMemoryErrorCorrectionMultiBitEcc   = 0x06,
  EfiMemoryErrorCorrectionCrc           = 0x07,
} EFI_MEMORY_ERROR_CORRECTION;
```

Memory Device (Type 17)

EFI_MEMORY_ARRAY_LINK_DATA

Summary

This data record describes a memory device.

Prototype

```
typedef struct {
  STRING_REF                MemoryDeviceLocator;
  STRING_REF                MemoryBankLocator;
  STRING_REF                MemoryManufacturer;
  STRING_REF                MemorySerialNumber;
  STRING_REF                MemoryAssetTag;
  STRING_REF                MemoryPartNumber;
  EFI_INTER_LINK_DATA       MemoryArrayLink;
  EFI_INTER_LINK_DATA       MemorySubArrayLink;
  UINT16                    MemoryTotalWidth;
  UINT16                    MemoryDataWidth;
  EFI_EXP_BASE2_DATA        MemoryDeviceSize;
  EFI_MEMORY_FORM_FACTOR    MemoryFormFactor;
  UINT8                     MemoryDeviceSet;
  EFI_MEMORY_ARRAY_TYPE     MemoryType;
  EFI_MEMORY_TYPE_DETAIL    MemoryTypeDetail;
  EFI_EXP_BASE10_DATA       MemoryTypeSpeed;
  EFI_MEMORY_STATE          MemoryState;
} EFI_MEMORY_ARRAY_LINK_DATA;
```

Parameters

MemoryDeviceLocator

A string that identifies the physically labeled socket or board position where the memory device is located. Type **STRING_REF** is defined in **EFI_HII_PROTOCOL.NewString()** in the *Intel® Platform Innovation Framework for EFI Human Interface Infrastructure Specification*.

MemoryBankLocator

A string denoting the physically labeled bank where the memory device is located.

MemoryManufacturer

A string denoting the memory manufacturer.

MemorySerialNumber

A string denoting the serial number of the memory device.

MemoryAssetTag

The asset tag of the memory device.

MemoryPartNumber

A string denoting the part number of the memory device.

MemoryArrayLink

A link to a memory array structure set. See Physical Memory Array (Type 16) for memory array structures. Type **EFI_INTER_LINK_DATA** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

MemorySubArrayLink

A link to a memory array structure set. See Physical Memory Array (Type 16) for memory array structures.

MemoryTotalWidth

The total width in bits of this memory device. If there are no error correcting bits, then the total width equals the data width. If the width is unknown, then set the field to 0xFFFF.

MemoryDataWidth

The data width in bits of the memory device. A data width of 0x00 and a total width of 0x08 indicate that the device is used solely for error correction.

MemoryDeviceSize

The size in bytes of the memory device. A value of 0x00 denotes that no device is installed, while a value of all *F*s denotes that the size is not known. Type **EFI_EXP_BASE2_DATA** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide.*

MemoryFormFactor

The form factor of the memory device. Type **EFI_MEMORY_FORM_FACTOR** is defined in "Related Definitions" below.

MemoryDeviceSet

A memory device set that must be populated with all devices of the same type and size. A value of 0x00 indicates that the device is not part of any set. A value of 0xFF indicates that the attribute is unknown. Any other value denotes the set number.

MemoryType

The memory type in the socket. Type **EFI_MEMORY_ARRAY_TYPE** is defined in "Related Definitions" below.

MemoryTypeDetail

The memory type details. Type **EFI_MEMORY_TYPE_DETAIL** is defined in "Related Definitions" below.

MemoryTypeSpeed

The memory speed in megahertz (MHz). A value of 0x00 denotes that the speed is unknown. Type **EFI_EXP_BASE10_DATA** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide.*

MemoryState

The memory state. Type **EFI_MEMORY_STATE** is defined in "Related Definitions" below.

Description

This data record describes a memory device. This data record is a structure.

The type definition structure for **EFI_MEMORY_ARRAY_LINK_DATA** is in SMBIOS 2.3.4:

- Table 3.3.18, Type 17, with the following offsets:
  - Offset 0x4
  - Offset 0x6
  - Offset 0x8
  - Offset 0xA
  - Offset 0xC
- Table 3.3.18.1, Type 17, with the following offsets:
  - Offset 0xE
  - Offset 0xF
  - Offset 0x10
  - Offset 0x11
- Table 3.3.18.2, Type 17, Offset 0x12
- Table 3.3.18.3, Type 17, with the following offsets:
  - Offset 0x13
  - Offset 0x15
  - Offset 0x17
  - Offset 0x18
  - Offset 0x19
  - Offset 0x1A

For this data record, **EFI_SUBCLASS_TYPE1_HEADER**.*RecordType* = **EFI_MEMORY_ARRAY_LINK_RECORD_NUMBER**. Type **EFI_SUBCLASS_TYPE1_HEADER** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

Related Definitions

```
//*******************************************************
// Record number
//*******************************************************
#define EFI_MEMORY_ARRAY_LINK_RECORD_NUMBER 0x00000003
```

```
//*************************************************************
// EFI_MEMORY_FORM_FACTOR
//*************************************************************
typedef enum _EFI_MEMORY_FORM_FACTOR {
  EfiMemoryFormFactorOther                  = 0x01,
  EfiMemoryFormFactorUnknown                = 0x02,
  EfiMemoryFormFactorSimm                   = 0x03,
  EfiMemoryFormFactorSip                    = 0x04,
  EfiMemoryFormFactorChip                   = 0x05,
  EfiMemoryFormFactorDip                    = 0x06,
  EfiMemoryFormFactorZip                    = 0x07,
  EfiMemoryFormFactorProprietaryCard        = 0x08,
  EfiMemoryFormFactorDimm                   = 0x09,
  EfiMemoryFormFactorTsop                   = 0x0A,
  EfiMemoryFormFactorRowOfChips             = 0x0B,
  EfiMemoryFormFactorRimm                   = 0x0C,
  EfiMemoryFormFactorSodimm                 = 0x0D,
  EfiMemoryFormFactorSrimm                  = 0x0E
} EFI_MEMORY_FORM_FACTOR;


//*************************************************************
// EFI_MEMORY_ARRAY_TYPE
//*************************************************************
typedef enum _EFI_MEMORY_ARRAY_TYPE {
  EfiMemoryTypeOther       = 0x01,
  EfiMemoryTypeUnknown     = 0x02,
  EfiMemoryTypeDram        = 0x03,
  EfiMemoryTypeEdram       = 0x04,
  EfiMemoryTypeVram        = 0x05,
  EfiMemoryTypeSram        = 0x06,
  EfiMemoryTypeRam         = 0x07,
  EfiMemoryTypeRom         = 0x08,
  EfiMemoryTypeFlash       = 0x09,
  EfiMemoryTypeEeprom      = 0x0A,
  EfiMemoryTypeFeprom      = 0x0B,
  EfiMemoryTypeEprom       = 0x0C,
  EfiMemoryTypeCdram       = 0x0D,
  EfiMemoryType3Dram       = 0x0E,
  EfiMemoryTypeSdram       = 0x0F,
  EfiMemoryTypeSgram       = 0x10,
  EfiMemoryTypeRdram       = 0x11,
  EfiMemoryTypeDdr         = 0x12
} EFI_MEMORY_ARRAY_TYPE;
```

```
//*************************************************************
// EFI_MEMORY_TYPE_DETAIL
//*************************************************************
typedef enum {
  UINT32 Reserved          :1;
  UINT32 Other             :1;
  UINT32 Unknown           :1;
  UINT32 FastPaged         :1;
  UINT32 StaticColumn      :1;
  UINT32 PseudoStatic      :1;
  UINT32 Rambus            :1;
  UINT32 Synchronous       :1;
  UINT32 Cmos              :1;
  UINT32 Edo               :1;
  UINT32 WindowDram        :1;
  UINT32 CacheDram         :1;
  UINT32 Nonvolatile       :1;
  UINT32 Reserved1         :19;
} EFI_MEMORY_TYPE_DETAIL ;


//*************************************************************
// EFI_MEMORY_STATE
//*************************************************************
typedef enum {
  EfiMemoryStateEnabled = 0,
  EfiMemoryStateUnknown = 1,
  EfiMemoryStateUnsupported = 2,
  EfiMemoryStateError = 3,
  EfiMemoryStateAbsent = 4,
  EfiMemoryStateDisabled = 5
} EFI_MEMORY_STATE;
```

Memory Array Mapped Address (Type 19)

EFI_MEMORY_ARRAY_START_ADDRESS_DATA

Summary

This data record refers to a specified physical memory array associated with a given memory range.

Prototype

```
typedef struct {
  EFI_PHYSICAL_ADDRESS          MemoryArrayStartAddress;
  EFI_PHYSICAL_ADDRESS          MemoryArrayEndAddress;
  EFI_INTER_LINK_DATA           PhysicalMemoryArrayLink;
  UINT16                        MemoryArrayPartitionWidth;
} EFI_MEMORY_ARRAY_START_ADDRESS_DATA;
```

Parameters

MemoryArrayStartAddress

The starting physical address in bytes of memory mapped to a specified physical memory array. Type **EFI_PHYSICAL_ADDRESS** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

MemoryArrayEndAddress

The last physical address in bytes of memory mapped to a specified physical memory array.

PhysicalMemoryArrayLink

See Physical Memory Array (Type 16) for physical memory array structures. Type **EFI_INTER_LINK_DATA** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

MemoryArrayPartitionWidth

The number of memory devices that form a single row of memory for the address partition.

Description

This data record refers to a specified physical memory array associated with a given memory range. This data record is a structure.

The type definition structure for **EFI_MEMORY_ARRAY_START_ADDRESS_DATA** is in SMBIOS 2.3.4, Table 3.3.20, Type 19, with the following offsets:

- Offset 0x4
- Offset 0x8
- Offset 0xC
- Offset 0xE

For this data record, **EFI_SUBCLASS_TYPE1_HEADER**.*RecordType* = **EFI_MEMORY_ARRAY_START_ADDRESS_RECORD_NUMBER**. Type **EFI_SUBCLASS_TYPE1_HEADER** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

Related Definitions

```
//*************************************************************
// Record number
//*************************************************************
#define EFI_MEMORY_ARRAY_START_ADDRESS_RECORD_NUMBER 0x00000004
```

Memory Device Mapped Address (Type 20)

EFI_MEMORY_DEVICE_START_ADDRESS_DATA

Summary

This data record refers to a physical memory device that is associated with a given memory range.

Prototype

```
typedef struct {
  EFI_PHYSICAL_ADDRESS      MemoryDeviceStartAddress;
  EFI_PHYSICAL_ADDRESS      MemoryDeviceEndAddress;
  EFI_INTER_LINK_DATA       PhysicalMemoryDeviceLink;
  EFI_INTER_LINK_DATA       PhysicalMemoryArrayLink;
  UINT8                     MemoryDevicePartitionRowPosition;
  UINT8                     MemoryDeviceInterleavePosition;
  UINT8                     MemoryDeviceInterleaveDataDepth;
} EFI_MEMORY_DEVICE_START_ADDRESS_DATA;
```

Parameters

MemoryDeviceStartAddress

The starting physical address that is associated with the device. Type **EFI_PHYSICAL_ADDRESS** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

MemoryDeviceEndAddress

The ending physical address that is associated with the device.

PhysicalMemoryDeviceLink

A link to the memory device data structure. See Memory Device (Type 17). Type **EFI_INTER_LINK_DATA** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

PhysicalMemoryArrayLink

A link to the memory array data structure. See Physical Memory Array (Type 16).

MemoryDevicePartitionRowPosition

The position of the memory device in a row. A value of 0x00 is reserved and a value of 0xFF indicates that the position is unknown.

MemoryDeviceInterleavePosition

The position of the device in an interleave.

MemoryDeviceInterleaveDataDepth

The maximum number of consecutive rows from the device that are accessed in a single interleave transfer. A value of 0x00 indicates that the device is not interleaved and a value of 0xFF indicates that the interleave configuration is unknown.

Description

This data record refers to a physical memory device that is associated with a given memory range. This data record is a structure.

The type definition structure for **EFI_MEMORY_DEVICE_START_ADDRESS_DATA** is in SMBIOS 2.3.4, Table 3.3.21, Type 20, with the following offsets:

- Offset 0x4
- Offset 0x8
- Offset 0xC
- Offset 0xE
- Offset 0x10
- Offset 0x11
- Offset 0x12

For this data record, **EFI_SUBCLASS_TYPE1_HEADER**.*RecordType* = **EFI_MEMORY_DEVICE_START_ADDRESS_RECORD_NUMBER**. Type **EFI_SUBCLASS_TYPE1_HEADER** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

Related Definitions

```
//*************************************************************
// Record number
//*************************************************************
#define EFI_MEMORY_DEVICE_START_ADDRESS_RECORD_NUMBER 0x00000005
```

Memory Channel (Type 37)

Memory Channel Type

EFI_MEMORY_CHANNEL_TYPE_DATA

Summary

This data record refers the type of memory that is associated with the channel.

Prototype

```
typedef struct {
  EFI_MEMORY_CHANNEL_TYPE        MemoryChannelType;
  UINT8                          MemoryChannelMaximumLoad;
  UINT8                          MemoryChannelDeviceCount;
} EFI_MEMORY_CHANNEL_TYPE_DATA;
```

Parameters

MemoryChannelType

The type of memory that is associated with the channel. Type **EFI_MEMORY_CHANNEL_TYPE** is defined in "Related Definitions" below.

MemoryChannelMaximumLoad

The maximum load that is supported by the channel.

MemoryChannelDeviceCount

The number of memory devices on this channel.

Description

This data record refers the type of memory that is associated with the channel. This data record is a structure.

The type definition structure for **EFI_MEMORY_CHANNEL_TYPE_DATA** is in SMBIOS 2.3.4, Table 3.3.38, Type 37, with the following offsets:

- Offset 0x4
- Offset 0x5
- Offset 0x6

For this data record, **EFI_SUBCLASS_TYPE1_HEADER**.*RecordType* = **EFI_MEMORY_CHANNEL_TYPE_RECORD_NUMBER**. Type **EFI_SUBCLASS_TYPE1_HEADER** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

Related Definitions

```
//*****************************************************************
// Record number
//*****************************************************************
#define EFI_MEMORY_CHANNEL_TYPE_RECORD_NUMBER      0x000000006


//*****************************************************************
// EFI_MEMORY_CHANNEL_TYPE
//*****************************************************************
typedef enum _EFI_MEMORY_CHANNEL_TYPE {
  EfiMemoryChannelTypeOther                    = 1,
  EfiMemoryChannelTypeUnknown                  = 2,
  EfiMemoryChannelTypeRambus                   = 3,
  EfiMemoryChannelTypeSyncLink                 = 4
} EFI_MEMORY_CHANNEL_TYPE;
```

Memory Channel Device

EFI_MEMORY_CHANNEL_DEVICE_DATA

Summary

This data record refers to the memory device that is associated with the memory channel.

Prototype

```
typedef struct _EFI_MEMORY_CHANNEL_DEVICE_DATA {
  UINT8                    DeviceId;
  EFI_INTER_LINK_DATA      DeviceLink;
  UINT8                    MemoryChannelDeviceLoad;
} EFI_MEMORY_CHANNEL_DEVICE_DATA;
```

Parameters

DeviceId

A number between one and *MemoryChannelDeviceCount* plus an arbitrary base. See Memory Channel Type.

DeviceLink

The *Link* of the associated memory device. See Memory Device (Type 17) for memory devices. Type **EFI_INTER_LINK_DATA** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

MemoryChannelDeviceLoad

The number of load units that this device consumes.

Description

This data record refers to the memory device that is associated with the memory channel. This data record is a structure.

The type definition structure for **EFI_MEMORY_CHANNEL_DEVICE_DATA** is in SMBIOS 2.3.4, Table 3.3.38, Type 37, with the following offsets:

- Offset 0x7
- Offset 0x8

For this data record, **EFI_SUBCLASS_TYPE1_HEADER**.*RecordType* = **EFI_MEMORY_CHANNEL_DEVICE_RECORD_NUMBER**. Type **EFI_SUBCLASS_TYPE1_HEADER** is defined in the *Intel® Platform Innovation Framework for EFI Data Hub Subclass Design Guide*.

Related Definitions

```
//*************************************************************
// Record number
//*************************************************************
#define EFI_MEMORY_CHANNEL_DEVICE_RECORD_NUMBER       0x00000007
```

4 Examples

Legacy Examples

Memory Region Size

Memory Region Size: Example 1

Data Example: Two processors each have private memory of 1 MB, while the rest of the 64 MB memory is global. Memory is available on all PCI buses and is system memory. One agent having the GUID of global memory usage produces all data. Note that 1 MB base-2 has an exponent of 20 decimal or 0x14.

```
MemoryRegion0.ProducerName = GUID of global memory usage;
MemoryRegion0.Instance = 0x01;
MemoryRegion0.ProcessorNumber = 0x00;
MemoryRegion0.StartBusNumber = 0x00;
MemoryRegion0.EndBusNumber = 0xFF;
MemoryRegion0.RegionType = EfiMemoryRegionMemory;
MemoryRegion0.MemorySize.Value = 0x01;
MemoryRegion0.MemorySize.Exponent = 0x14;
MemoryRegion0.MemoryStartAddress = 0x00;

MemoryRegion1.ProducerName = GUID of global memory usage;
MemoryRegion1.Instance = 0x02;
MemoryRegion1.ProcessorNumber = 0x00;
MemoryRegion1.StartBusNumber = 0x00;
MemoryRegion1.EndBusNumber = 0xFF;
MemoryRegion1.MemoryRegionType = EfiMemoryRegionMemory;
MemoryRegion1.MemorySize.Value = 0x01;
MemoryRegion1.MemorySize.Exponent = 0x14;
MemoryRegion1.MemoryStartAddress = 0x00;

MemoryRegion2.ProducerName = GUID of global memory usage;
MemoryRegion2.Instance = 0x03;
MemoryRegion2.ProcessorNumber = 0xFFFF;
MemoryRegion2.StartBusNumber = 0x00;
MemoryRegion2.EndBusNumber = 0xFF;
MemoryRegion2.MemoryRegionType = EfiMemoryRegionMemory;
MemoryRegion2.MemorySize.Value = 0x3F;
MemoryRegion2.MemorySize.Exponent = 0x14;
MemoryRegion2.MemoryStartAddress = 0x100000;
```

Memory Region Size: Example 2

Data Example: There are two peer-to-host bridges. The first peer-to-host bridge has bus 0 through 0x3F and the second peer-to-host bridge has the remaining buses. Each peer-to-host bridge has 32 MB of memory. Memory is available to all processors and is system memory.

```
MemoryRegion0.ProducerName = GUID of global memory usage;
MemoryRegion0.Instance = 0x01;
MemoryRegion0.ProcessorNumber = 0xFFFF;
MemoryRegion0.StartBusNumber = 0x00;
MemoryRegion0.EndBusNumber = 0x3F;
MemoryRegion0.MemoryRegionType = EfiMemoryRegionMemory;
MemoryRegion0.MemorySize.Value = 0x20;
MemoryRegion0.MemorySize.Exponent = 0x14;
MemoryRegion0.MemoryStartAddress = 0x00;

MemoryRegion1.ProducerName = GUID of global memory usage;
MemoryRegion1.Instance = 0x02;
MemoryRegion1.ProcessorNumber = 0xFFFF;
MemoryRegion1.StartBusNumber = 0x40;
MemoryRegion1.EndBusNumber = 0xFF;
MemoryRegion1.MemoryRegionType = EfiMemoryRegionMemory;
MemoryRegion1.MemorySize.Value = 0x20;
MemoryRegion1.MemorySize.Exponent = 0x14;
MemoryRegion1.MemoryStartAddress = 0x2000000;
```

Physical Memory Array Examples

Memory Array Location

Data Example: The memory array is on the system board.

```
MemoryRegion3.Instance = 0x01;
MemoryRegion3.MemoryArrayLocation =
EfiMemoryArrayLocationSystemBoard;
```

Memory Array Use

Data Example: The memory array is system memory.

```
MemoryArrayUse = EfiMemoryArrayUseSystemMemory;
```

Memory Error Correction

Data Example: Memory error correction is multibit error correction code (ECC).

```
MemoryErrorCorrection = EfiMemoryErrorCorrectionMuliBitEcc;
```

Maximum Memory Capacity

Data Example: The maximum memory capacity is 1 GB.

```
MaximumMemoryCapacity.Value = 0x1;
MaximumMemoryCapacity.Exponent = 0x1E;
```

Number Memory Devices

Data Example: The memory module has six memory slots

```
MemoryRegion3.Instance = 0x01;
MemoryRegion3.SubInstance = 0x00;
MemoryRegion3.NumberMemoryDevices = 0x06;
```

Memory Device Examples

Memory Device Locator

Data Example: The memory device locator is C4R2.

```
MemoryDeviceLocator = String reference to “C4R2”,0;
```

Memory Bank Locator

Data Example: The memory bank is C4.

```
MemoryBankLocator = String reference to “C4”,0;
```

Memory Manufacturer

Data Example: The memory manufacturer is Samsung.

```
MemoryManufacturer = String reference to “Samsung”,0;
```

Memory Serial Number

Data Example: The memory serial number is 9Q3005.

```
MemorySerialNumber = String reference to “9Q3005”,0;
```

Memory Asset Tag

Data Example: The asset tag is 1234.

```
MemoryAssetTag = String reference to “1234”,0;
```

Memory Part Number

Data Example: The memory part number is 987654.

```
MemoryPartNumber = String reference to  “987654”,0;
```

Memory Array Link

Data Example: The memory device is part of a memory array that has an *Instance* of 0x1. It is in slot four and therefore has a *SubInstance* of 0x04. Note that the array has no *SubInstance*.

```
MemoryRegion3.Instance = 0x01
MemoryRegion3.SubInstance = 0x04
MemoryRegion3.MemoryArrayLink.Instance = 0x01;
MemoryRegion3.MemoryArrayLink.SubInstance = 0x00;
```

Memory Subarray Link

Data Example: *Instance* 0x03 refers to an array of memory slots. Each slot is defined by the *SubInstance*. *MemorySubArrayLink* for slot four refers to a new array (the memory stick in the slot) with *Instance* 0x27. The *SubInstance* refers to the memory stick side. *MemorySubArrayLink* for the second side refers to a new array (the side) with *Instance* 0x53. The *SubInstance* refers to the memory bank. *MemorySubArrayLink* refers to a new array (the third bank) with *Instance* 0x9A. The *SubInstance* refers to a fifth memory chip.

The hierarchy looks like the following:

Memory Subsystem: **`Instance = 0x03`**

Memory Slot: **`SubInstance = 0x04; Instance = 0x027`**

Memory Stick Side: **`SubInstance = 0x02; Instance = 0x53`**

Memory Bank: **`SubInstance = 0x03; Instance 0x9A`**

Memory Total Width

Data Example: Total memory width is 36 decimal or 0x24. The width is 32 bits data plus 4 bits parity.

```
MemoryTotalWidth = 0x24;
```

Memory Data Width

Data Example: Memory data width is 32 bits decimal or 0x20.

```
MemoryDataWidth = 0x20;
```

Memory Device Size

Data Example: The maximum memory capacity is 1 MB.

```
MemoryDeviceSize.Value = 0x1;
MemoryDeviceSize.Exponent = 0x0014;
```

Memory Form Factor

Data Example: The memory form factor is dual inline memory module (DIMM).

```
MemoryFormFactor = EfiMemoryFormFactorDimm;
```

Memory Device Set

Data Example: Memory is part of device set 2.

```
MemoryDeviceSet = 0x2;
```

Memory Type

Data Example: The memory type ID is DRAM.

```
MemoryType = EfiMemoryTypeDram;
```

Memory Type Detail

Data Example: The memory type detail is fast-paged memory.

```
MemoryTypeDetail = 01000b;
```

Memory Type Speed

Data Example: Memory speed is 133 MHz decimal or 0x85.

```
MemoryTypeSpeed.Value = 0x85;
MemoryTypeSpeed.Exponent = 0x00;
```

Memory State

Data Example: The memory state is enabled.

```
MemoryState = EfiMemoryStateEnabled;
```

Memory Array Mapped Address Examples

Memory Array Start Address

Data Example: The memory array start address is 1 GB.

```
MemoryArrayStartAddress = 0x4000 0000;
```

Memory Array End Address

Data Example: The memory array is 1 GB in length and starts at 1 GB.

```
MemoryArrayEndAddress = 0x7FFF FC00;
```

Physical Memory Array Link

Data Example: Memory is part of the physical array *Instance* 0x01.

```
MemoryRegion3.Instance = 0x01;
MemoryRegion3.SubInstance = 0x00;
MemoryRegion3.PhysicalMemoryArrayLink.Instance = 0x01;
MemoryRegion3.PhysicalMemoryArrayLink.SubInstance = 0x00;
```

Memory Array Partition Width

Data Example: The memory array consists of 36 decimal devices or 0x24.

```
MemoryArrayPartitionWidth = 0x24;
```

Memory Device Mapped Address Examples

Memory Device Start Address

Data Example: The memory array start address is 1 GB.

```
MemoryArrayStartAddress = 0x4000 0000;
```

Memory Device End Address

Data Example: The memory array is 1 GB in length and starts at 1 GB.

```
MemoryArrayEndAddress = 0x7FFF FC00;
```

Physical Memory Device Link

Data Example: Memory is part of the physical memory device *Instance* 0x01 and *SubInstance* 0x04 (slot four). Also see the examples in Memory Device Examples.

```
MemoryRegion3.Instance = 0x01;
MemoryRegion3.SubInstance = 0x04;
MemoryRegion3.PhysicalMemoryDeviceLink.Instance = 0x01;
MemoryRegion3.PhysicalMemoryDeviceLink.SubInstance = 0x04;
```

Physical Memory Array Link

Data Example: Memory is part of the physical array Instance 0x01. Also see the examples in Physical Memory Array Examples.

```
MemoryRegion3.Instance = 0x01;
MemoryRegion3.SubInstance = 0x04;
MemoryRegion3.PhysicalMemoryArrayLink.Instance = 0x01;
MemoryRegion3.PhysicalMemoryArrayLink.SubInstance = 0x04;
```

Memory Device Partition Row Position

Data Example: Memory is the fourth device.

```
MemoryDevicePartitionRowPosition = 0x4;
```

Memory Device Interleave Position

Data Example: Memory is the second device in interleave.

```
MemoryDeviceInterleavePosition = 0x2;
```

Memory Device Interleave Data Depth

Data Example: Memory accesses two rows for an interleave transaction.

```
MemoryDeviceInterleaveDataDepth = 0x2;
```

Memory Channel Examples

Memory Channel Type

Data Example: The memory channel is Rambus.

```
MemoryChannelType = EfiMemoryChannelTypeRamBus;
```

Memory Channel Maximum Load

Data Example: The memory channel maximum load is 144 decimal units or 0x90.

```
MemoryChannelMaximumLoad = 0x90;
```

Memory Channel Device Count

Data Example: The memory device count is 36 decimal or 0x24.

```
MemoryChannelDeviceCount = 0x0000 0000 0000 0400;
```

Memory Channel Device Load

Data Example: Each memory device has a load of four units and is associated with the physical device *Instance* 0x427. This memory device is the third in this channel.

```
MemoryRegion3.Instance = 0x27;
MemoryRegion3.DeviceId = 0x3;
MemoryRegion3.DeviceLink.Instance = 0x427;
MemoryRegion3.MemoryChannelDeviceLoad = 0x04;
```