Intel® Platform Innovation Framework for EFI Hand-Off Block (HOB) Specification

Draft for Review

Version 0.9

September 16, 2003

Intel order number xxxxxx-001

Revision History

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

Contents

Figures

Tables

1 Introduction

Overview

This specification defines the core code that is required for an implementation of Hand-Off Blocks (HOBs) in the Intel® Platform Innovation Framework for EFI (hereafter referred to as the "Framework"). A HOB is a binary data structure that passes system state information from the HOB producer phase to the HOB consumer phase in the Framework architecture. This HOB specification does the following:

- Describes the basic components of HOBs and the rules for constructing them
- Provides code definitions for the HOB data types and structures 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.

Pseudo-Code Conventions

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

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

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

Typographic Conventions

This document uses the typographic and illustrative conventions described below:

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

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

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

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

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

2 Design Discussion

Explanation of HOB Terms

Because HOBs are the key architectural mechanism that is used to hand off system information in the early preboot stages and because not all implementations of the Framework will use the Pre-EFI Initialization (PEI) and Driver Execution Environment (DXE) phases, this specification refrains from using the PEI and DXE nomenclature used in other Framework specifications.

Instead, this specification uses the following terms to refer to the phases that deal with HOBs:

- HOB producer phase
- HOB consumer phase

The *HOB producer phase* is the preboot phase in which HOBs and the HOB list are created. The *HOB consumer phase* is the preboot phase to which the HOB list is passed and then consumed.

If the Framework implementation incorporates the PEI and DXE, the HOB producer phase is the PEI phase and the HOB consumer phase is the DXE phase. The producer and consumer can change, however, depending on the implementation.

The following table translates the terminology used in this specification with that used in other Framework specifications.

Table 2-1. Translation of HOB Specification Terminology

Term Used in the HOB Specification	Term Used in Other Framework Specifications
HOB producer phase	PEI phase
HOB consumer phase	DXE phase
executable content in the HOB producer phase	Pre-EFI Initialization Module (PEIM)
hand-off into the HOB consumer phase	DXE Initial Program Load (IPL) PEIM or DXE IPL PEIM-to-PEIM Interface (PPI)
platform boot-policy phase	Boot Device Selection (BDS) phase

HOB Overview

The HOB producer phase provides a simple mechanism to allocate memory for data storage during the phase's execution. The data store is architecturally defined and described by HOBs. This data store is also passed to the HOB producer phase when it is invoked from the HOB producer phase.

The basic container of data storage is named a *Hand-Off Block,* or HOB. HOBs are allocated sequentially in memory that is available to executable content in the HOB producer phase. There are a series of services that facilitate HOB manipulation. The sequential list of HOBs in memory will be referred to as the *HOB list.*

See Code Definitions for definitions of the various HOB types and the semantics for creating them.

Example HOB Producer Phase Memory Map and Usage

The figure below shows an example of the HOB producer phase memory map and its usage. This map is a possible means by which to subdivide the region.

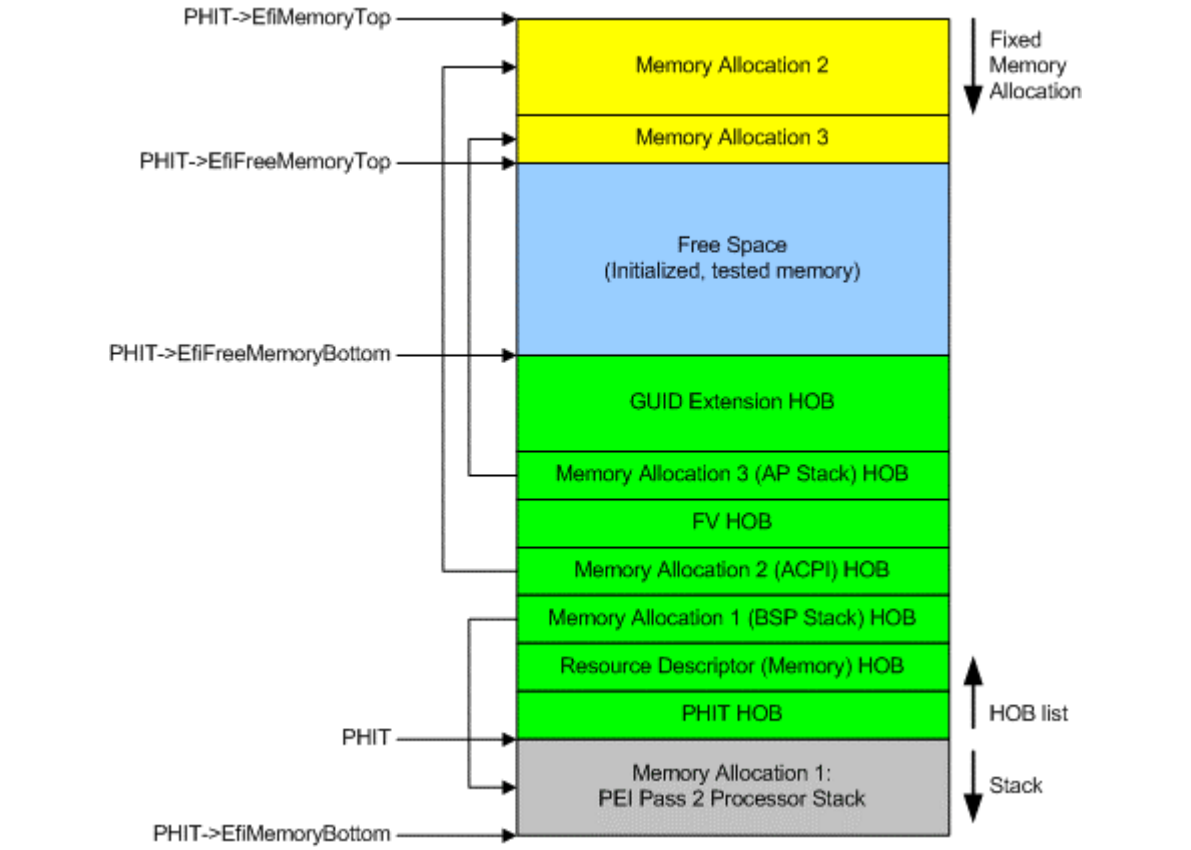

Figure 2-1. Example HOB Producer Phase Memory Map and Usage

HOB List

The first HOB in the HOB list must be the Phase Handoff Information Table (PHIT) HOB. The last HOB in the HOB list must be the end of list HOB.

Only HOB producer phase components are allowed to make additions or changes to HOBs. Once the HOB list is passed into the HOB consumer phase, it is effectively read only. The ramifications of a read-only HOB list is that handoff information, such as boot mode, must be handled in a distinguished fashion. For example, if the HOB consumer phase were to engender a recovery condition, it would not update the boot mode but instead would implement the action using a special type of reset call. The HOB list contains system state data at the time of HOB consumer–to–HOB producer handoff and does not represent the current system state during the HOB consumer phase.

Constructing the HOB List

Constructing the Initial HOB List

The HOB list is initially built by the HOB producer phase. The HOB list is created in memory that is present, initialized, and tested. Once the initial HOB list has been created, the physical memory cannot be remapped, interleaved, or otherwise moved by a subsequent software agent.

The HOB producer phase **must** build the following three HOBs in the initial HOB list before exposing the list to other modules:

- The PHIT HOB
- A memory allocation HOB describing where the boot-strap processor (BSP) stack for permanent memory is located

  or

  A memory allocation HOB describing where the BSP store for permanent memory is located (Itanium® processor family only)
- A resource descriptor HOB that describes a physical memory range encompassing the HOB producer phase memory range with its attributes set as present, initialized, and tested

The HOB list creator may build more HOBs into the initial HOB list, such as additional HOBs to describe other physical memory ranges. There can also be additional modules, which might include a HOB producer phase–specific HOB to record memory errors discovered during initialization.

When the HOB producer phase completes its list creation, it exposes a pointer to the PHIT HOB to other modules.

HOB Construction Rules

HOB construction must obey the following rules:

1. All HOBs must start with a HOB generic header. This requirement allows users to locate the HOBs in which they are interested while skipping the rest. See the **EFI_HOB_GENERIC_HEADER** definition.
2. HOBs may contain boot services data that is available during the HOB producer and consumer phases only until the HOB consumer phase is terminated.
3. HOBs may be relocated in system memory by the HOB consumer phase. HOBs must not contain pointers to other data in the HOB list, including that in other HOBs. The table must be able to be copied without requiring internal pointer adjustment.
4. All HOBs must be multiples of 8 bytes in length. This requirement meets the alignment restrictions of the Itanium® processor family.
5. The PHIT HOB must always begin on an 8-byte boundary. Due to this requirement and requirement #4 in this list, all HOBs will begin on an 8-byte boundary.
6. HOBs are added to the end of the HOB list. HOBs can only be added to the HOB list during the HOB producer phase, not the HOB consumer phase.
7. HOBs cannot be deleted. The generic HOB header of each HOB must describe the length of the HOB so that the next HOB can be found. A private GUIDed HOB may provide a mechanism to mark some or its entire contents invalid; however, this mechanism is beyond the scope of this document.

NOTE

The HOB list must be valid (i.e., no HOBs "under construction") when any HOB producer phase service is invoked. Another HOB producer phase component's function might walk the HOB list, and if a HOB header contains invalid data, it might cause unreliable operation.

Adding to the HOB List

To add a HOB to the HOB list, HOB consumer phase software must obtain a pointer to the PHIT HOB (start of the HOB list) and follow these steps:

1. Determine *NewHobSize*, where *NewHobSize* is the size in bytes of the HOB to be created.
2. Check free memory to ensure that there is enough free memory to allocate the new HOB. This test is performed by checking that *NewHobSize* <= PHIT->*EfiFreeMemoryTop* - PHIT->*EfiFreeMemoryBottom*).
3. Construct the HOB at PHIT->*EfiFreeMemoryBottom*.
4. Set PHIT->*EfiFreeMemoryBottom* = PHIT->*EfiFreeMemoryBottom* + *NewHobSize*.

3
Code Definitions

Introduction

This section contains the basic definitions of various HOBs. All HOBs consist of a generic header, **EFI_HOB_GENERIC_HEADER**, that specifies the type and length of the HOB. Each HOB has additional data beyond the generic header, according to the HOB type. The following data types and structures are defined in this section:

- **EFI_HOB_GENERIC_HEADER**
- **EFI_HOB_HANDOFF_INFO_TABLE**
- **EFI_HOB_MEMORY_ALLOCATION**
- **EFI_HOB_MEMORY_ALLOCATION_STACK**
- **EFI_HOB_MEMORY_ALLOCATION_BSP_STORE**
- **EFI_HOB_MEMORY_ALLOCATION_MODULE**
- **EFI_HOB_RESOURCE_DESCRIPTOR**
- **EFI_HOB_GUID_TYPE**
- **EFI_HOB_FIRMWARE_VOLUME**
- **EFI_HOB_CPU**
- **EFI_HOB_MEMORY_POOL**
- **EFI_HOB_CAPSULE_VOLUME**
- **EFI_HOB_TYPE_UNUSED**
- **EFI_HOB_TYPE_END_OF_HOB_LIST**

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

- **EFI_HOB_MEMORY_ALLOCATION_HEADER**
- **EFI_RESOURCE_TYPE**
- **EFI_RESOURCE_ATTRIBUTE_TYPE**

HOB Generic Header

EFI_HOB_GENERIC_HEADER

Summary

Describes the format and size of the data inside the HOB. All HOBs must contain this generic HOB header.

Prototype

```
typedef struct _EFI_HOB_GENERIC_HEADER{
  UINT16  HobType;
  UINT16  HobLength;
  UINT32  Reserved;
} EFI_HOB_GENERIC_HEADER;
```

Parameters

HobType

Identifies the HOB data structure type. See "Related Definitions" below for the HOB types that are defined in this specification.

HobLength

The length in bytes of the HOB.

Reserved

For this version of the specification, this field must always be set to zero.

Description

All HOBs have a common header that is used for the following:

- Traversing to the next HOB
- Describing the format and size of the data inside the HOB

Related Definitions

The following values for *HobType* are defined by this specification.

```
//*******************************************************
// HobType values
//*******************************************************

#define EFI_HOB_TYPE_HANDOFF               0x0001
#define EFI_HOB_TYPE_MEMORY_ALLOCATION     0x0002
#define EFI_HOB_TYPE_RESOURCE_DESCRIPTOR   0x0003
#define EFI_HOB_TYPE_GUID_EXTENSION        0x0004
#define EFI_HOB_TYPE_FV                    0x0005
#define EFI_HOB_TYPE_CPU                   0x0006
#define EFI_HOB_TYPE_MEMORY_POOL           0x0007
#define EFI_HOB_TYPE_CV                    0x0008
#define EFI_HOB_TYPE_UNUSED                0xFFFE
#define EFI_HOB_TYPE_END_OF_HOB_LIST       0xffff
```

Other values for *HobType* are reserved for future use by this specification.

PHIT HOB

EFI_HOB_HANDOFF_INFO_TABLE (PHIT HOB)

Summary

Contains general state information used by the HOB producer phase. This HOB must be the first one in the HOB list.

Prototype

```
typedef struct _EFI_HOB_HANDOFF_INFO_TABLE {
  EFI_HOB_GENERIC_HEADER  Header;
  UINT32                  Version;
  EFI_BOOT_MODE           BootMode;
  EFI_PHYSICAL_ADDRESS    EfiMemoryTop;
  EFI_PHYSICAL_ADDRESS    EfiMemoryBottom;
  EFI_PHYSICAL_ADDRESS    EfiFreeMemoryTop;
  EFI_PHYSICAL_ADDRESS    EfiFreeMemoryBottom;
  EFI_PHYSICAL_ADDRESS    EfiEndOfHobList;
} EFI_HOB_HANDOFF_INFO_TABLE;
```

Parameters

Header

The HOB generic header. *Header.HobType* = **EFI_HOB_TYPE_HANDOFF**.

Version

The version number pertaining to the PHIT HOB definition. See "Related Definitions" below for the version number(s) defined by this specification. This value is 4 bytes in length to provide an 8-byte aligned entry when it is combined with the 4-byte *BootMode*.

BootMode

The system boot mode as determined during the HOB producer phase. Type **EFI_BOOT_MODE** is a **UINT32**; if the Framework implementation incorporates the PEI phase, the possible bit values are defined in the *Intel® Platform Innovation Framework for EFI Pre-EFI Initialization Core Interface Specification* (PEI CIS).

EfiMemoryTop

The highest address location of memory that is allocated for use by the HOB producer phase. This address must be 4 KB aligned to meet page restrictions of EFI. Type **EFI_PHYSICAL_ADDRESS** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

EfiMemoryBottom

The lowest address location of memory that is allocated for use by the HOB producer phase.

EfiFreeMemoryTop

The highest address location of free memory that is currently available for use by the HOB producer phase. This address must be 4 KB aligned to meet page restrictions of EFI.

EfiFreeMemoryBottom

The lowest address location of free memory that is available for use by the HOB producer phase.

EfiEndOfHobList

The end of the HOB list.

Description

The Phase Handoff Information Table (PHIT) HOB must be the first one in the HOB list. A pointer to this HOB is available to a HOB producer phase component through some service. This specification commonly refers to this HOB as the *PHIT HOB,* or sometimes the *handoff HOB.*

The HOB consumer phase reads the PHIT HOB during its initialization.

Related Definitions

```
//*******************************************************
// Version values
//*******************************************************

#define EFI_HOB_HANDOFF_TABLE_VERSION  0x0009
```

Memory Allocation HOB

Memory Allocation HOB

EFI_HOB_MEMORY_ALLOCATION

Summary

Describes all memory ranges used during the HOB producer phase that exist outside the HOB list. This HOB type describes how memory is used, not the physical attributes of memory.

Prototype

```
typedef struct _EFI_HOB_MEMORY_ALLOCATION {
  EFI_HOB_GENERIC_HEADER            Header;
  EFI_HOB_MEMORY_ALLOCATION_HEADER  AllocDescriptor;
  //
  // Additional data pertaining to the "Name" Guid memory
  // may go here.
  //
} EFI_HOB_MEMORY_ALLOCATION;
```

Parameters

Header

The HOB generic header. *Header.HobType* = **EFI_HOB_TYPE_MEMORY_ALLOCATION**.

AllocDescriptor

An instance of the **EFI_HOB_MEMORY_ALLOCATION_HEADER** that describes the various attributes of the logical memory allocation. The type field will be used for subsequent inclusion in the EFI memory map. Type **EFI_HOB_MEMORY_ALLOCATION_HEADER** is defined in "Related Definitions" below.

Description

The memory allocation HOB is used to describe memory usage outside the HOB list. The HOB consumer phase does not make assumptions about the contents of the memory that is allocated by the memory allocation HOB, and it will not move the data unless it has explicit knowledge of the memory allocation HOB's *Name* (**EFI_GUID**). Memory may be allocated in either the HOB producer phase memory area or other areas of present and initialized system memory.

The HOB consumer phase reads all memory allocation HOBs and allocates memory into the system memory map based on the following fields of **EFI_HOB_MEMORY_ALLOCATION_HEADER** of each memory allocation HOB:

- *MemoryBaseAddress*
- *MemoryLength*
- *MemoryType*

The HOB consumer phase does not parse the GUID-specific data identified by the Name field of each memory allocation HOB, except for a specific set of memory allocation HOBs that defined by this specification. A HOB consumer phase driver that corresponds to the specific Name GUIDed memory allocation HOB can parse the HOB list to find the specifically named memory allocation HOB and then manipulate the memory space as defined by the usage model for that GUID.

NOTE

Special design care should be taken to ensure that two HOB consumer phase components do not modify memory space that is described by a memory allocation HOB, because unpredictable behavior might result.

This specification defines a set of memory allocation HOBs that are architecturally used to allocate memory used by the HOB producer and consumer phases. Additionally, the following memory allocation HOBs are defined specifically for use by the final stage of the HOB producer phase to describe the processor state prior to handoff into the HOB consumer phase:

- BSP stack memory allocation HOB
- BSP store memory allocation HOB
- Memory allocation module HOB

Related Definitions

```
//*******************************************************
// EFI_HOB_MEMORY_ALLOCATION_HEADER
//*******************************************************

typedef struct _EFI_HOB_MEMORY_ALLOCATION_HEADER {
  EFI_GUID               Name;
  EFI_PHYSICAL_ADDRESS   MemoryBaseAddress;
  UINT64                 MemoryLength;
  EFI_MEMORY_TYPE        MemoryType;   // UINT32
  UINT8                  Reserved[4]; // Padding for Itanium®
                                      // processor family
} EFI_HOB_MEMORY_ALLOCATION_HEADER;
```

Name

A GUID that defines the memory allocation region's type and purpose, as well as other fields within the memory allocation HOB. This GUID is used to define the additional data within the HOB that may be present for the memory allocation HOB. Type **EFI_GUID** is defined in **InstallProtocolInterface()** in the *EFI 1.10 Specification.*

MemoryBaseAddress

The base address of memory allocated by this HOB. Type **EFI_PHYSICAL_ADDRESS** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

MemoryLength

The length in bytes of memory allocated by this HOB.

MemoryType

Defines the type of memory allocated by this HOB. The memory type definition follows the **EFI_MEMORY_TYPE** definition. Type **EFI_MEMORY_TYPE** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

Reserved

For this version of the specification, this field will always be set to zero.

NOTE

MemoryBaseAddress and MemoryLength must each have 4 KB granularity to meet the page size requirements of EFI.

Boot-Strap Processor (BSP) Stack Memory Allocation HOB

EFI_HOB_MEMORY_ALLOCATION_STACK

Summary

Describes the memory stack that is produced by the HOB producer phase and upon which all post-memory-installed executable content in the HOB producer phase is executing.

GUID

```
#define EFI_HOB_MEMORY_ALLOC_STACK_GUID  \
{0x4ed4bf27, 0x4092, 0x42e9, 0x80, 0x7d, 0x52, 0x7b, 0x1d, 0x0,
0xc9, 0xbd};
```

Prototype

```
typedef struct _EFI_HOB_MEMORY_ALLOCATION_STACK {
  EFI_HOB_GENERIC_HEADER            Header;
  EFI_HOB_MEMORY_ALLOCATION_HEADER  AllocDescriptor;
} EFI_HOB_MEMORY_ALLOCATION_STACK;
```

Parameters

Header

The HOB generic header. *Header.HobType* = **EFI_HOB_TYPE_MEMORY_ALLOCATION**.

AllocDescriptor

An instance of the **EFI_HOB_MEMORY_ALLOCATION_HEADER** that describes the various attributes of the logical memory allocation. The type field will be used for subsequent inclusion in the EFI memory map. Type **EFI_HOB_MEMORY_ALLOCATION_HEADER** is defined in **EFI_HOB_MEMORY_ALLOCATION**.

Description

This HOB describes the memory stack that is produced by the HOB producer phase and upon which all post-memory-installed executable content in the HOB producer phase is executing. It is necessary for the hand-off into the HOB consumer phase to know this information so that it can appropriately map this stack into its own execution environment and describe it in any subsequent memory maps.

The HOB consumer phase reads this HOB during its initialization. The HOB consumer phase may elect to move or relocate the BSP's stack to meet size and location requirements that are defined by the HOB consumer phase's implementation. Therefore, other HOB consumer phase components cannot rely on the BSP stack memory allocation HOB to describe where the BSP stack is located during execution of the HOB consumer phase.

The BSP stack memory allocation HOB without any additional qualification describes either of the following:

- The stack that is currently consumed by the BSP
- The processor that is currently executing the HOB producer phase and its executable content

The model for the Framework and the HOB producer phase is that of a single-threaded execution environment, so it is this single, distinguished thread of control whose environment is described by this HOB. The Itanium® processor family has the additional requirement of having to describe the value of the **BSPSTORE (AR18)** ("Backing Store Pointer Store") register, which holds the successive location in memory where the Itanium processor family Register Stack Engine (RSE) will spill its values.

In addition, Itanium®-based systems feature a system architecture where all processors come out of reset and execute the reset path concurrently. As such, the stack resources that are consumed by these alternate agents need to be described even though they are not responsible for executing the main thread of control through the HOB producer and consumer phases.

Boot-Strap Processor (BSP) BSPSTORE Memory Allocation HOB

EFI_HOB_MEMORY_ALLOCATION_BSP_STORE

NOTE

This HOB is valid for the Itanium® processor family only.

Summary

Defines the location of the boot-strap processor (BSP) BSPStore ("Backing Store Pointer Store") register overflow store.

GUID

```
#define EFI_HOB_MEMORY_ALLOC_BSP_STORE_GUID  \
{0x564b33cd, 0xc92a, 0x4593, 0x90, 0xbf, 0x24, 0x73, 0xe4, 0x3c,
 0x63, 0x22};
```

Prototype

```
typedef struct _EFI_HOB_MEMORY_ALLOCATION_BSP_STORE {
  EFI_HOB_GENERIC_HEADER            Header;
  EFI_HOB_MEMORY_ALLOCATION_HEADER  AllocDescriptor;
} EFI_HOB_MEMORY_ALLOCATION_BSP_STORE;
```

Parameters

Header

The HOB generic header. *Header.HobType* = **EFI_HOB_TYPE_MEMORY_ALLOCATION**.

AllocDescriptor

An instance of the **EFI_HOB_MEMORY_ALLOCATION_HEADER** that describes the various attributes of the logical memory allocation. The type field will be used for subsequent inclusion in the EFI memory map. Type **EFI_HOB_MEMORY_ALLOCATION_HEADER** is defined in the HOB type **EFI_HOB_MEMORY_ALLOCATION**.

Description

The HOB consumer phase reads this HOB during its initialization. The HOB consumer phase may elect to move or relocate the BSP's register store to meet size and location requirements that are defined by the HOB consumer phase's implementation. Therefore, other HOB consumer phase components cannot rely on the BSP store memory allocation HOB to describe where the BSP store is located during execution of the HOB consumer phase.

This HOB is valid for the Itanium processor family only.

Memory Allocation Module HOB

EFI_HOB_MEMORY_ALLOCATION_MODULE

Summary

Defines the location and entry point of the HOB consumer phase.

GUID

```
#define EFI_HOB_MEMORY_ALLOC_MODULE_GUID  \
{0xf8e21975, 0x899, 0x4f58, 0xa4, 0xbe, 0x55, 0x25, 0xa9, 0xc6,
0xd7, 0x7a}
```

Prototype

```
typedef struct {
  EFI_HOB_GENERIC_HEADER             Header;
  EFI_HOB_MEMORY_ALLOCATION_HEADER   MemoryAllocationHeader;
  EFI_GUID                           ModuleName;
  EFI_PHYSICAL_ADDRESS               EntryPoint;
} EFI_HOB_MEMORY_ALLOCATION_MODULE;
```

Parameters

Header

The HOB generic header. *Header.HobType =* **EFI_HOB_TYPE_MEMORY_ALLOCATION**.

MemoryAllocationHeader

An instance of the **EFI_HOB_MEMORY_ALLOCATION_HEADER** that describes the various attributes of the logical memory allocation. The type field will be used for subsequent inclusion in the EFI memory map. Type **EFI_HOB_MEMORY_ALLOCATION_HEADER** is defined in the HOB type **EFI_HOB_MEMORY_ALLOCATION**.

ModuleName

The GUID specifying the values of the firmware file system name that contains the HOB consumer phase component. Type **EFI_GUID** is defined in **InstallProtocolInterface()** in the *EFI 1.10 Specification.*

EntryPoint

The address of the memory-mapped firmware volume that contains the HOB consumer phase firmware file. Type **EFI_PHYSICAL_ADDRESS** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

Description

The HOB consumer phase reads the memory allocation module HOB during its initialization. This HOB describes the memory location of the HOB consumer phase. The HOB consumer phase should use the information to create the image handle for the HOB consumer phase.

Resource Descriptor HOB

EFI_HOB_RESOURCE_DESCRIPTOR

Summary

Describes the resource properties of all fixed, nonrelocatable resource ranges found on the processor host bus during the HOB producer phase.

Prototype

```
typedef struct _EFI_HOB_RESOURCE_DESCRIPTOR {
  EFI_HOB_GENERIC_HEADER       Header;
  EFI_GUID                     Owner;
  EFI_RESOURCE_TYPE            ResourceType;
  EFI_RESOURCE_ATTRIBUTE_TYPE  ResourceAttribute;
  EFI_PHYSICAL_ADDRESS         PhysicalStart;
  UINT64                       ResourceLength;
} EFI_HOB_RESOURCE_DESCRIPTOR;
```

Parameters

Header

The HOB generic header. *Header.HobType* = **EFI_HOB_TYPE_RESOURCE_DESCRIPTOR**.

Owner

A GUID representing the owner of the resource. This GUID is used by HOB consumer phase components to correlate device ownership of a resource.

ResourceType

Resource type enumeration as defined by **EFI_RESOURCE_TYPE**. Type **EFI_RESOURCE_TYPE** is defined in "Related Definitions" below.

ResourceAttribute

Resource attributes as defined by **EFI_RESOURCE_ATTRIBUTE_TYPE**. Type **EFI_RESOURCE_ATTRIBUTE_TYPE** is defined in "Related Definitions" below.

PhysicalStart

Physical start address of the resource region. Type **EFI_PHYSICAL_ADDRESS** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

ResourceLength

Number of bytes of the resource region.

Description

The resource descriptor HOB describes the resource properties of all fixed, nonrelocatable resource ranges found on the processor host bus during the HOB producer phase. This HOB type does not describe how memory is used but instead describes the attributes of the physical memory present.

The HOB consumer phase reads all resource descriptor HOBs when it established the initial Global Coherency Domain (GCD) map. The minimum requirement for the HOB producer phase is that executable content in the HOB producer phase report one of the following:

- The resources that are necessary to start the HOB consumer phase
- The fixed resources that are not captured by HOB consumer phase driver components that were started prior to the dynamic system configuration performed by the platform boot-policy phase

For example, executable content in the HOB producer phase should report any physical memory found during the HOB producer phase. Another example is reporting the Boot Firmware Volume (BFV) that contains firmware volume(s). Executable content in the HOB producer phase does not need to report fixed system resources such as I/O port 70h/71h (real-time clock) because these fixed resources can be allocated from the GCD by a platform-specific chipset driver loading in the HOB consumer phase prior to the platform boot-policy phase, for example.

Current thinking is that the GCD does not track the HOB's *Owner* GUID, so a HOB consumer phase component that assumes ownership of a device's resource must deallocate the resource initialized by the HOB producer phase from the GCD before attempting to assign the devices resource to itself in the HOB consumer phase.

Related Definitions

There can only be a single *ResourceType* field, characterized as follows.

```
//*******************************************************
// EFI_RESOURCE_TYPE
//*******************************************************

typedef UINT32 EFI_RESOURCE_TYPE;

#define EFI_RESOURCE_SYSTEM_MEMORY           0x00000000
#define EFI_RESOURCE_MEMORY_MAPPED_IO        0x00000001
#define EFI_RESOURCE_IO                      0x00000002
#define EFI_RESOURCE_FIRMWARE_DEVICE         0x00000003
#define EFI_RESOURCE_MEMORY_MAPPED_IO_PORT   0x00000004
#define EFI_RESOURCE_MEMORY_RESERVED         0x00000005
#define EFI_RESOURCE_IO_RESERVED             0x00000006
#define EFI_RESOURCE_MAX_MEMORY_TYPE         0x00000007
```

The following table describes the fields listed in the above definition.

EFI_RESOURCE_SYSTEM_MEMORY	Memory that persists out of the HOB producer phase.
EFI_RESOURCE_MEMORY_MAPPED_IO	Memory-mapped I/O that is programmed in the HOB producer phase.
EFI_RESOURCE_IO	Processor I/O space.
EFI_RESOURCE_FIRMWARE_DEVICE	Memory-mapped firmware devices.
EFI_RESOURCE_MEMORY_MAPPED_IO_PORT	Memory that is decoded to produce I/O cycles.
EFI_RESOURCE_MEMORY_RESERVED	Reserved memory address space.

EFI_RESOURCE_IO_RESERVED	Reserved I/O address space.
EFI_RESOURCE_MAX_MEMORY_TYPE	Any reported HOB value of this type or greater should be deemed illegal. This value could increase with successive revisions of this specification, so the “illegality” will also be based upon the revision field of the PHIT HOB.

The *ResourceAttribute* field is characterized as follows:

```
//*******************************************************
// EFI_RESOURCE_ATTRIBUTE_TYPE
//*******************************************************

typedef UINT32 EFI_RESOURCE_ATTRIBUTE_TYPE;


// These types can be ORed together as needed.
//
// The first three enumerations describe settings
//
#define EFI_RESOURCE_ATTRIBUTE_PRESENT      0x00000001
#define EFI_RESOURCE_ATTRIBUTE_INITIALIZED  0x00000002
#define EFI_RESOURCE_ATTRIBUTE_TESTED       0x00000004

//
// The rest of the settings describe capabilities
//
#define EFI_RESOURCE_ATTRIBUTE_SINGLE_BIT_ECC    0x00000008
#define EFI_RESOURCE_ATTRIBUTE_MULTIPLE_BIT_ECC  0x00000010
#define EFI_RESOURCE_ATTRIBUTE_ECC_RESERVED_1    0x00000020
#define EFI_RESOURCE_ATTRIBUTE_ECC_RESERVED_2    0x00000040
#define EFI_RESOURCE_ATTRIBUTE_READ_PROTECTED    0x00000080
#define EFI_RESOURCE_ATTRIBUTE_WRITE_PROTECTED   0x00000100
#define EFI_RESOURCE_ATTRIBUTE_EXECUTION_PROTECTED
                                                 0x00000200
#define EFI_RESOURCE_ATTRIBUTE_UNCACHEABLE       0x00000400
#define EFI_RESOURCE_ATTRIBUTE_WRITE_COMBINEABLE 0x00000800
#define EFI_RESOURCE_ATTRIBUTE_WRITE_THROUGH_CACHEABLE
                                                 0x00001000
#define EFI_RESOURCE_ATTRIBUTE_WRITE_BACK_CACHEABLE
                                                 0x00002000
#define EFI_RESOURCE_ATTRIBUTE_16_BIT_IO         0x00004000
#define EFI_RESOURCE_ATTRIBUTE_32_BIT_IO         0x00008000
#define EFI_RESOURCE_ATTRIBUTE_64_BIT_IO         0x00010000
#define EFI_RESOURCE_ATTRIBUTE_UNCACHED_EXPORTED 0x00020000
```

The following table describes the fields listed in the above definition.

EFI_RESOURCE_ATTRIBUTE_ PRESENT	Physical memory attribute: The memory region exists.
EFI_RESOURCE_ATTRIBUTE_ INITIALIZED	Physical memory attribute: The memory region has been initialized.
EFI_RESOURCE_ATTRIBUTE_ TESTED	Physical memory attribute: The memory region has been tested.
EFI_RESOURCE_ATTRIBUTE_SINGLE_ BIT_ECC	Physical memory attribute: The memory region supports single-bit ECC.
EFI_RESOURCE_ATTRIBUTE_ MULTIPLE_BIT_ECC	Physical memory attribute: The memory region supports multibit ECC.
EFI_RESOURCE_ATTRIBUTE_ECC_ RESERVED_1	Physical memory attribute: The memory region supports reserved ECC.
EFI_RESOURCE_ATTRIBUTE_ECC_ RESERVED_2	Physical memory attribute: The memory region supports reserved ECC.
EFI_RESOURCE_ATTRIBUTE_READ_ PROTECTED	Physical memory protection attribute: The memory region is read protected.
EFI_RESOURCE_ATTRIBUTE_WRITE_ PROTECTED	Physical memory protection attribute: The memory region is write protected.
EFI_RESOURCE_ATTRIBUTE_ EXECUTION_PROTECTED	Physical memory protection attribute: The memory region is execution protected.
EFI_RESOURCE_ATTRIBUTE_ UNCACHEABLE	Memory cacheability attribute: The memory does not support caching.
EFI_RESOURCE_ATTRIBUTE_WRITE_ THROUGH_CACHEABLE	Memory cacheability attribute: The memory supports being programmed with a write-through cacheable attribute.
EFI_RESOURCE_ATTRIBUTE_WRITE_ COMBINEABLE	Memory cacheability attribute: The memory supports a write-combining attribute.
EFI_RESOURCE_ATTRIBUTE_WRITE_ BACK_CACHEABLE	Memory cacheability attribute: The memory region supports being configured as cacheable with a write-back policy. Reads and writes that hit in the cache do not propagate to main memory. Dirty data is written back to main memory when a new cache line is allocated.
EFI_RESOURCE_ATTRIBUTE_16_ BIT_IO	Memory physical attribute: The memory supports 16-bit I/O.
EFI_RESOURCE_ATTRIBUTE_32_ BIT_IO	Memory physical attribute: The memory supports 32-bit I/O.

EFI_RESOURCE_ATTRIBUTE_64_ BIT_IO	Memory physical attribute: The memory supports 64-bit I/O.
EFI_RESOURCE_ATTRIBUTE_ UNCACHED_EXPORTED	Memory cacheability attribute: The memory region is uncacheable and exported and supports the fetch and add semaphore mechanism.

The table below specifies the resource attributes applicable to each resource type.

Table 3-1. HOB Producer Phase Resource Types

EFI_RESOURCE_ATTRIBUTE_TYPE	HOB Producer Phase System Memory	HOB Producer Phase Memory-Mapped I/O	HOB Producer Phase I/O
Present	X		
Initialized	X		
Tested	X		
SingleBitEcc	X		
MultipleBitEcc	X		
EccReserved1	X		
EccReserved2	X		
ReadProtected	X	X	
WriteProtected	X	X	
ExecutionProtected	X		
Uncacheable	X	X	
WriteThroughCacheable	X	X	
WriteCombineable	X	X	
WriteBackCacheable	X	X	
16bitIO			X
32bitIO			X
64bitIO			X
UncachedExported	X	X	

GUID Extension HOB

EFI_HOB_GUID_TYPE

Summary

Allows writers of executable content in the HOB producer phase to maintain and manage HOBs whose types are not included in this specification. Specifically, writers of executable content in the HOB producer phase can generate a GUID and name their own HOB entries using this module-specific value.

Prototype

```
typedef struct _EFI_HOB_GUID_TYPE {
  EFI_HOB_GENERIC_HEADER  Header;
  EFI_GUID                Name;

  //
  // Guid specific data goes here
  //
} EFI_HOB_GUID_TYPE;
```

Parameters

Header

The HOB generic header. *Header.HobType* = **EFI_HOB_TYPE_GUID_EXTENSION**.

Name

A GUID that defines the contents of this HOB. Type **EFI_GUID** is defined in **InstallProtocolInterface()** in the *EFI 1.10 Specification*.

Description

The GUID extension HOB allows writers of executable content in the HOB producer phase to create their own HOB definitions using a GUID. This HOB type should be used by all executable content in the HOB producer phase to define implementation-specific data areas that are not architectural. This HOB type may also pass implementation-specific data from executable content in the HOB producer phase to drivers in the HOB consumer phase.

A HOB consumer phase component such as a HOB consumer phase driver will read the GUID extension HOB during the HOB consumer phase. The HOB consumer phase component must inherently know the GUID for the GUID extension HOB for which it is scanning the HOB list. This knowledge establishes a contract on the HOB's definition and usage between the executable content in the HOB producer phase and the HOB consumer phase driver.

Firmware Volume HOB

EFI_HOB_FIRMWARE_VOLUME

Summary

Details the location of firmware volumes that contain firmware files.

Prototype

```
typedef struct {
  EFI_HOB_GENERIC_HEADER  Header;
  EFI_PHYSICAL_ADDRESS    BaseAddress;
  UINT64                  Length;
} EFI_HOB_FIRMWARE_VOLUME;
```

Parameters

Header

The HOB generic header. *Header.HobType* = **EFI_HOB_TYPE_FV**.

BaseAddress

The physical memory-mapped base address of the firmware volume. Type **EFI_PHYSICAL_ADDRESS** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

Length

The length in bytes of the firmware volume.

Description

The firmware volume HOB details the location of firmware volumes that contain firmware files. It includes a base address and length. In particular, the HOB consumer phase will use these HOBs to discover drivers to execute and the hand-off into the HOB consumer phase will use this HOB to discover the location of the HOB consumer phase firmware file.

The firmware volume HOB is produced in the following ways:

- By the executable content in the HOB producer phase in the Boot Firmware Volume (BFV) that understands the size and layout of the firmware volume(s) that are present in the platform.
- By a module that has loaded a firmware volume from some media into memory. The firmware volume HOB details this memory location.

Firmware volumes that described by the firmware volume HOB must have a firmware volume header whose definition matches that described in the *Intel® Platform Innovation Framework for EFI Firmware Volume Block Specification.*

The HOB consumer phase consumes all firmware volume HOBs that are presented by the HOB producer phase for use by its read-only support for the Framework Firmware File System (FFS). The HOB producer phase is required to describe any firmware volumes that may contain the HOB consumer phase or platform drivers that are required to discover other firmware volumes.

CPU HOB

EFI_HOB_CPU

Summary

Describes processor information, such as address space and I/O space capabilities.

Prototype

```
typedef struct _EFI_HOB_CPU {
  EFI_HOB_GENERIC_HEADER  Header;
  UINT8                   SizeOfMemorySpace;
  UINT8                   SizeOfIoSpace;
  UINT8                   Reserved[6];
} EFI_HOB_CPU;
```

Parameters

Header

The HOB generic header. *Header.HobType* = **EFI_HOB_TYPE_CPU**.

SizeOfMemorySpace

Identifies the maximum physical memory addressability of the processor.

SizeOfIoSpace

Identifies the maximum physical I/O addressability of the processor.

Reserved

For this version of the specification, this field will always be set to zero.

Description

The CPU HOB is produced by the processor executable content in the HOB producer phase. It describes processor information, such as address space and I/O space capabilities. The HOB consumer phase consumes this information to describe the extent of the GCD capabilities.

Memory Pool HOB

EFI_HOB_MEMORY_POOL

Summary

Describes pool memory allocations.

Prototype

```
typedef struct _EFI_HOB_MEMORY_POOL {
  EFI_HOB_GENERIC_HEADER       Header;
} EFI_HOB_MEMORY_POOL;
```

Parameters

Header

The HOB generic header. *Header.HobType* = **EFI_HOB_TYPE_MEMORY_POOL**.

Description

The memory pool HOB is produced by the HOB producer phase and describes pool memory allocations. The HOB consumer phase should be able to ignore these HOBs. The purpose of this HOB is to allow for the HOB producer phase to have a simple memory allocation mechanism within the HOB list. The size of the memory allocation is stipulated by the *HobLength* field in **EFI_HOB_GENERIC_HEADER**.

Capsule Volume HOB

EFI_HOB_CAPSULE_VOLUME

Summary

Details the location of capsule volumes that contain firmware files.

Prototype

```
typedef struct {
  EFI_HOB_GENERIC_HEADER     Header;
  EFI_PHYSICAL_ADDRESS       BaseAddress;
  UINT64                     Length;
} EFI_HOB_CAPSULE_VOLUME;
```

Parameters

Header

The HOB generic header. *Header.HobType* = **EFI_HOB_TYPE_CV**.

BaseAddress

The physical memory-mapped base address of the capsule volume. This value is set to point to the base of the contiguous memory of the capsule(s). Type **EFI_PHYSICAL_ADDRESS** is defined in **AllocatePages()** in the *EFI 1.10 Specification.*

Length

The length of the contiguous memory in bytes.

Description

The capsule volume HOB details the location of capsule volumes that contain firmware files. It includes a base address and length.

The capsule volume HOB is the same format as the firmware volume HOB but describes the contents of a capsule update. The HOB producer phase would typically discover and aggregate the memory contents of the capsule update and create the capsule volume HOB. Subsequently, an agent in the HOB consumer phase would discover the capsule volume HOB and effect the appropriate action, such as the update itself, based on the contents. See the *Intel® Platform Innovation Framework for EFI Capsule Specification* for more information on capsule update process.

Unused HOB

EFI_HOB_TYPE_UNUSED

Summary

Indicates that the contents of the HOB can be ignored.

Prototype

```
#define EFI_HOB_TYPE_UNUSED             0xFFFE
```

Description

This HOB type means that the contents of the HOB can be ignored. This type is necessary to support the simple, allocate-only architecture of HOBs that have no delete service. The consumer of the HOB list should ignore HOB entries with this type field.

An agent that wishes to make a HOB entry ignorable should set its type to the prototype defined above.

End of HOB List HOB

EFI_HOB_TYPE_END_OF_HOB_LIST

Summary

Indicates the end of the HOB list. This HOB must be the last one in the HOB list.

Prototype

```
#define EFI_HOB_TYPE_END_OF_HOB_LIST      0xffff
```

Description

This HOB type indicates the end of the HOB list. This HOB type must be the last HOB type in the HOB list and terminates the HOB list. A HOB list should be considered ill formed if it does not have a final HOB of type **EFI_HOB_TYPE_END_OF_HOB_LIST**.