EDK II Module Information (INF) File Specification

TABLE OF CONTENTS

EDK II Module Information (INF) File Specification

Revision 1.27

12/01/2020 05:32:02

Acknowledgements

Revision History

Revision	Revision History	Date
1.0	Initial release.	December 2007
1.1	Updated based on errata	August 2008
1,2	Updated based on enhancement requests	June 2009
1.21	Updated to support UEFI 2.3 and PI 1.2 specifications	March 2010
	Added new element, UEFI_HII_RESOURCE_SECTION to [Defines] section	
	Added new SMM_CORE module type	
	Updated for clarification	
	Permit NULL values in place of PCD default values	
	Updated to correct items listed in the errata document	
	Permit whitespace characters between token elements	
	Fixed BuildOptions separator between Family and the tool change	

	information to match the ":' implementation	
	Changes to appearance for readability	
	Moved EDK INF description from sections 2 and 3 to an Appendix	
	User feature requests:	
	Updated the description of the FeatureFlag Expression for all occurrences in Chapter 3 to be either a Shell style or a C style expression.	
1.22	Grammatical and formatting changes.	May 2010
1.22 w/	Updates:	December 2011
Errata A	Updated to support UEFI version 2.3.1 and updated spec release dates in Introduction	
	Clarify UEFI's PI Distribution Package Specification	
	Standardize Common data definitions for all specifications	
	Grammatical, formatting and spelling changes	
	Replaced "should" with wording saying that it is	
	"recommended"	
	Added PCI_COMPRESS definition in [Defines] section	
	Added the DPX_SOURCE statement back into the [Defines] section	
	Added VALID_ARCHITECTURES comment definition	
	back to the [Defines] section to formalize this comment which may be used by tools. This had been removed from the 1.22 spec as it was assumed that tools could determine valid architectures (other than ALL architectures) by the use of architectural modifiers in section tags.	
	Removed restriction about comments in the DPX_SOURCE file - C style comments are allowed	
	Updated DEPEX content for USER_DEFINED module types	
	Removed EDK content from EBNF in Chapter 3, as this chapter only describes the content for EDK II INF files; for clarity, moved EDK content from descriptions in Chapter 2 to Appendix A	
	Added EBNF for `<Extension>`	
	Added rules for how macros can be shared between sections	
	Update the EBNF for paths so that a macro can have a path that does not end with a file separator; also allow using a path and filename as a macro value; clarify that macros are only expanded in the EDK II INF files, never evaluated during the initial parsing of the file	
	Removed duplicate content and added the scoping rules for Macros, clarified MACRO summary; made the value optional so that a C flag macro can be specified without a value; require the "=" in a macro DEFINE statement	
	Removed references to system environment variables in the Macros section and removed table	
	Revised EBNF for PCD sections to allow more precise definitions	
	Specify how PCD values are obtained	
	Changed definition of a C Array to ensure that an empty array is not specified	

	Allow any non-zero value to be TRUE	
	Use separate EBNF for each PCD datum type, also explain the PCD usages; describe, in section 2, what sections are valid for binary only modules, and what sections are prohibited in binary only modules	
	Clarify that C data arrays must be byte arrays for PCD value fields; both C format and registry format GUID structures are not permitted in VOID* PCD value fields	
	The # character is optional for the header comment block in EDK INF files	
	Prohibit specifying something an a common section and in an architecturally specific section (something that is architecture specific cannot be common to all architectures)	
	Removed FFE from entries as they have no meaning, nothing changes - build does not break if they are there	
1.22 w/	Updates:	June 2012
Errata B	Section 1.3, page 5, Updated specs definition to include released errata	
	Section 3.8, page 67, Removed Value field for DynamicEx PCDs listed in a generated "As Built" INF file	
	Appendix F, page 120, Replaced invalid "FW" with "PE32" for file type of the binary image	
	Section 2.7, pages 25 & 26, Clarified binary file types are leaf sections, removed LIB, as EDK II build system does not support distribution of binary libraries	
	Section 2.7, page 25, Removed GUID encapsulation section keyword from the `[Binaries]` section	
	`<FileType>` definition - the binary file must be a leaf file type	
	Section 3.2, page 37, require `<Depex>` sections for `PEIM` ,	
	`DXE_DRIVER` , `DXE_RUNTIME_DRIVER` ,	
	`DXE_SAL_DRIVER` and `DXE_SMM_DRIVER`	
	Table 3, page 23 and Section 3.5, page 60, Removed references to `build_rule.txt` - this file is used by tools, no user editing is required	
	Section 3.15, page 84 & 85, Separated out the	
	`SUBTYPE_GUID` entry in the `[Binaries]` section `<FileType>` definition, as this entry requires a GUID value - also, added text to only allow unique	
	`SUBTYPE_GUID` `<GuidValue>` pairs per section	
	Section 3.2.1, pages 44 & 45, Fixed the DOS `<EOL>` character sequence	
	Section 3.11, page 76, Clarify what goes into a generated Binary INF file for Protocols	
	Added a generated binary INF in Appendix F	
	Cleanup of tables in Appendix G	
	Updated Example INF files in Appendix D and Appendix E	
	Section 3.4 Added description of ENTRY_POINT and	
	UNLOAD_IMAGE elements in the `[Defines]` section	
1.22 w/	Updates:	August 2013

Errata C	Section 1.3, updated UDP - Errata version of the UEFI/PI Distribution Package Spec.	
	Section 2.7 and 3.15, added a binary file type of	
	`DISPOSABLE` which will not be processed by the EDK II tools.	
	Section 3.6, 3.8, clarify that the "As Built" INF file is always generated by the build system	
	Section 3.7, clarify that this section is required to list all dependent packages for PCDs listed in an "As Built" INF file	
	Section 3.8, clarify the types of PCDs that will be generated in "As Built" INF files	
	Section 3.3 Added Doxygen tags for Binary Header, Copyright from the Source INF file, containing the date of the last functional update to the source files is also the date that should be used for a Binary "As Built" INF file	
	Put the `BUILD_NUMBER` element back into the	
	`[Defines] section; this was inadvertently removed in Errata A`	
	Clarify that all entries are required within a Binary Header section.	
	Prohibit FeatureFlagExpressions for PCDs, GUIDs, Protocols and PPIs in the generated "As Built" INF files.	
	Fixed CRLF to be the correct hex bytes.	
	Reformatted the Header EBNF	
	Removed unused EBNF entry, `<ValPcds>`	
	Added Reference to EDK II Build Specification for PCD processing rules.	
	Remove sentences referring to lengths of PCD VOID* entries in section 2.14	
	Clarify that the Unicode format files are UCS-2LE encoded.	
1.22 w/	Updates:	March 2014
Errata D	Clarified that only [UserExtensions] sections with a UserId of TianoCore will be copied into the As Built INF generated by the EDK II build tools.	
	Clarify that [Depex] section tags must be unique.	
	Clarify the use of [Depex] sections in library modules.	
1.24	Updates:	August 2014
	Change revision number of this specification from 1.22 to	
	1.24	
	Update `INF_VERSION` to `0x00010017`	
	Added `MODULE_UNI_FILE` entry to the `[Defines]` section; this file must end with an extension of .uni, .UNI or .Uni	
	Added reserved `TianoCore` user extension for	
	"ExtraFiles"	
	Allow Space and Unicode characters in the directory path identified by the system environment variable,	
	`WORKSPACE`	
1.24 w/	Updates:	December

1.24 w/	Updates:	2014
Errata A	Revised ordering of the top level EBNF for an INF file to match the output of the Intel(R) UEFI Distribution Packaging Tool at the start of chapter 3.2	
	Updated specification dates in section 1.2 and added two new specs	
	Updated INF_VERSION to 0x00010018	
	Allow specifying the INF_VERSION value as a decimal value, such as 1.24.	
	Modified Section 2.14, allowing Feature Flag Expressions, removed expression syntax from the Common EBNF as it is now covered by its own specification.	
1.24 w/	Updates:	March 2015
Errata B	Update link to the EDK II Specifications, fixed the name of the Multi-String .UNI File Format Specification	
	Update usage, UNDEFINED, in Parameters sections for Guids, Protocols, PPIs and PCDs in chapter 3	
	Add clarification of the Event Types in chapter 3	
	Added UEFI PI PEI Boot Mode declarations in 3.2.5 to the list while keeping the synonyms that were already defined. Added descriptions as well.	
	Adding HOB type, UNUSED from the PI Specification	
1.24 w/	Updates:	August 2015
Errata C	Enable Feature Flag Expressions for entries in the INF file for Protocols, PPIs, GUID and PCD entries	
	Updated other sections to specify how the build system will evaluate the Feature Flag Expression	
	Prohibit using #include statements in UNI files specified in the MODULE_UNI_FILE entry	
1.25	Revised WORKSPACE wording for updated build system that can handle packages located outside of the WORKSPACE directory tree (refer to the TianoCore.org/EDKII website for additional instructions on setting up a development environment).	January 2016
1.26	Convert to GitBooks	May 2017
	#463 INF spec: document the LIB file type under the [Binaries] Section	
	#548 [INF spec] INF [LibraryClasses] section should not support ModuleType	
	#522 INF spec: add the clarification that PCD value may from build command	
1.27	Update version to 1.27	Mar 2018
	Add Flexible PCD value format support	

1 INTRODUCTION

This document describes the EDK II build information (INF) file format. This format supports the new build requirements of build EDK components and EDK II modules within the EDK II build infrastructure.

The EDK II Build Infrastructure supports creation of binary images that comply with Unified EFI (UEFI) 2.5 and UEFI Platform Infrastructure (PI) 1.4 specifications.

This version of the specification clarifies (or enables) existing content, no new content has been introduced.

1.1 Overview

This document describes the format of EDK II INF files that has the following requirements:

Compatible

Backward compatibility with the existing INF file formats. Changes made to this specification must not require changes to existing INF files.

Simplified platform build and configuration

Simplify the build setup and configuration for a given platform. The process of adding EDK and EDK II firmware components to a firmware volume on any given platform was also simplified.

Distributing Modules

Enable easy distribution of modules, both in source and binary form. Individual modules may be compiled and distributed in binary form, which may be integrated into a platform image, or into an option ROM image.

1.2 Terms

The following terms are used throughout this document to describe varying aspects of input localization:

BaseTools

The BaseTools are the tools required for an EDK II build.

BDS

Framework Boot Device Selection phase.

BNF

BNF is an acronym for "Backus Naur Form." John Backus and Peter Naur introduced for the first time a formal notation to describe the syntax of a given language.

Component

An executable image. Components defined in this specification support one of the defined module types.

DEC

EDK II Package Declaration File. This file declares information about what is provided in the package. An EDK II package is a collection of like content.

DEPEX

Module dependency expressions that describe runtime process restrictions.

Dist

This refers to a distribution package that conforms to the UEFI Platform Initialization Distribution Package Specification.

DSC

EDK II Platform Description File. This file describes what and how modules, libraries and components are to be built, as well as defining library instances which will be used when linking EDK II modules.

DXE

Framework Driver Execution Environment phase.

DXE SAL

A special class of DXE module that produces SAL Runtime Services. DXE SAL modules differ from DXE Runtime modules in that the DXE Runtime modules support Virtual mode OS calls at OS runtime and DXE SAL modules support intermixing Virtual or Physical mode OS calls.

DXE SMM

A special class of DXE module that is loaded into the System Management Mode memory.

DXE Runtime

Special class of DXE module that provides Runtime Services

EBNF

Extended "Backus-Naur Form" meta-syntax notation with the following additional constructs: square brackets "[...]" surround optional items, suffix "*" for a sequence of zero or more of an item, suffix "+" for one or more of an item, suffix "?" for zero or one of an item, curly braces "{...}" enclosing a list of alternatives, and super/subscripts indicating between n and m occurrences.

EDK

Extensible Firmware Interface Development Kit, the original implementation of the Intel(R) Platform Innovation Framework for EFI Specifications developed in 2007.

EDK II

EFI Development Kit, version II that provides updated firmware module layouts and custom tools, superseding the original EDK.

EDK Compatibility Package (ECP)

The EDK Compatibility Package (ECP) provides libraries that will permit using most existing EDK drivers with the EDK II build environment and EDK II platforms.

EFI

Generic term that refers to one of the versions of the EFI specification: EFI 1.02, EFI 1.10 or any of the UEFI specifications.

FDF

EDK II Flash definition file. This file is used to define the content and binary image layouts for firmware images, update capsules and PCI option ROMs.

FLASH

This term is used throughout this document to describe one of the following:

- An image that is loaded into a hardware device on a platform - traditional ROM image
- An image that is loaded into an Option ROM device on an add-in card
- A bootable image that is installed on removable, bootable media, such as a Floppy, CD-ROM or USB storage device.
- An image that is contains update information that will be processed by OS Runtime services to interact with EFI Runtime services to update a traditional ROM image.
- A UEFI application that can be accessed during boot (at an EFI Shell Prompt), prior to hand-off to the OS Loader.

Foundation

The set of code and interfaces that holds implementations of EFI together.

Framework

Intel(R) Platform Innovation Framework for EFI consists of the Foundation, plus other modular components that characterize the portability surface for modular components designed to work on any implementation of the Tiano architecture.

GUID

Globally Unique Identifier. A 128-bit value used to name entities uniquely. A unique GUID can be generated by an individual without the help of a centralized authority. This allows the generation of names that will never conflict, even among multiple, unrelated parties. GUID values can be registry format (8-4-4-4-12) or C data structure format.

GUID also refers to an API named by a GUID.

HII

Human Interface Infrastructure. This generally refers to the database that contains string, font, and IFR information along with other pieces that use one of the database components.

HOB

Hand-off blocks are key architectural mechanisms that are used to hand off system information in the early pre-boot stages.

IFR

Internal Forms Representation. This is the binary encoding that is used for the representation of user interface pages.

INF

EDK II Module Information File. This file describes how the module is coded. For EDK, this file describes how the component or library is coded as well as providing some basic build information.

- Source INF - An EDK II Module Information file that contains content in a [Sources] section and it does not contain a [Binaries] section. If the [Binaries] section is empty or the only entries in the [Binaries] section are of type DISPOSABLE, then the [Binaries] section is ignored.
- Binary INF - An EDK II Module Information file that has a [Binaries] section and does not contain a [Sources] section or the [Sources] section is empty.
- Mixed INF - An EDK II Module Information file that contains content in both [Sources] and [Binaries] sections and there are entries in the [Binaries] section are not of type DISPOSABLE
- AsBuilt INF - An EDK II Module Information file generated by the EDK II build system when building source content (listed in a [Sources] section).

Library Class

A library class defines the API or interface set for a library. The consumer of the library is coded to the library class definition. Library classes are defined via a library class .h file that is published by a package. See the EDK 2.0 Module Development Environment Library Specification for some example library classes.

Library Instance

An implementation of one or more library classes. See the EDK 2.0 Module Development Environment Library Specification for a list of sample library instances.

Module

A module is either an executable image or a library instance. For a list of module types supported by this package, see module type.

Module Type

All libraries and components belong to one of the following module types: `BASE` , `SEC` , `PEI_CORE` , `PEIM` , `DXE_CORE` , `DXE_DRIVER` , `DXE_RUNTIME_DRIVER` , `DXE_SMM_DRIVER` , `DXE_SAL_DRIVER` , `UEFI_DRIVER` , or `UEFI_APPLICATION` . These definitions provide a framework that is consistent with a similar set of requirements. A module that is of module type `BASE` , depends only on headers and libraries provided in the MDE, while a module that is of module type DXE_DRIVER depends on common DXE components. For a definition of the various module types, see module type. The EDK II build system also permits modules of type `USER_DEFINED` . These modules will not be processed by the EDK II Build system.

Package

A package is a container. It can hold a collection of files for any given set of modules. Packages may be described as containing zero or more of any of the following:

- source modules, containing all source files and descriptions of a module
- binary modules, containing EFI Sections or a Framework File System and a description file specific to linking and binary editing of features and attributes specified in a Platform Configuration Database (PCD,)
- mixed modules, with both binary and source modules

Multiple modules can be combined into a package, and multiple packages can be combined into a single package.

PCD

Platform Configuration Database.

PEI

Pre-EFI Initialization Phase.

PEIM

An API named by a GUID.

PPI

A PEIM-to-PEIM Interface that is named by a GUID.

Protocol

An API named by a GUID.

Runtime Services

Interfaces that provide access to underlying platform-specific hardware that might be useful during OS runtime, such as time and date services. These services become active during the boot process but also persist after the OS loader terminates boot services.

SAL

System Abstraction Layer. A firmware interface specification used on Intel(R) Itanium(R) Processor based systems.

SEC

Security Phase is the code in the Framework that contains the processor reset vector and launches PEI. This phase is separate from PEI because some security schemes require ownership of the reset vector.

SKU

Stock Keeping Unit.

SMM

System Management Mode. A generic term for the execution mode entered when a CPU detects an SMI. The firmware, in response to the interrupt type, will gain control in physical mode. For this document, "SMM" describes the operational regime for IA32 and x64 processors that share the OS-transparent characteristics.

UEFI Application

An application that follows the UEFI specification. The only difference between a UEFI application and a UEFI driver is that an application is unloaded from memory when it exits regardless of return status, while a driver that returns a successful return status is not unloaded when its entry point exits.

UEFI Driver

A driver that follows the UEFI specification.

UEFI Specification Version 2.5

Current UEFI version.

UEFI Platform Initialization Distribution Package Specification Version 1.0

The current version of this specification includes Errata B.

UEFI Platform Initialization Specification 1.4

Current version of the UEFI PI specification.

Unified EFI Forum

A non-profit collaborative trade organization formed to promote and manage the UEFI standard. For more information, see http://www.uefi.org.

VFR

Visual Forms Representation.

VPD

Vital Product Data that is read-only binary configuration data, typically located within a region of a flash part. This data would typically be updated as part of the firmware build, post firmware build (via patching tools), through automation on a manufacturing line as the 'FLASH' parts are programmed or through special tools.

1.3 Related Information

The following publications and sources of information may be useful to you or are referred to by this specification:

- Unified Extensible Firmware Interface Specification, Version 2.5, Unified EFI, Inc, 2015, http://www.uefi.org.
- UEFI Platform Initialization Specification, Version 1.4, Unified EFI, Inc., 2015, http://www.uefi.org.
- UEFI Platform Initialization Distribution Package Specification, Version 1.0 with Errata B, Unified EFI, Inc., 2014, http://www.uefi.org.
- Intel(R) Platform Innovation Framework for EFI Specifications, Intel, 2007, http://www.intel.com/technology/framework/.
- http://tianocore.sourceforge.net/wiki/EDK_II_Specifications
  - EDK II Module Writers Guide, Intel, 2010.
  - EDK II User Manual, Intel, 2010.
  - EDK II C Coding Standard, Intel, 2015.
  - EDK II Build Specification, Intel, 2016.
  - EDK II DEC File Specification, Intel, 2016.
  - EDK II DSC Specification, Intel, 2016.
  - EDK II FDF Specification, Intel, 2016.
  - Multi-String UNI File Format Specification, Intel, 2016.
  - EDK II Expression Syntax Specification, Intel, 2015.
  - VFR Programming Language, Intel, 2015.
  - UEFI Packaging Tool (UEFIPT) Quick Start, Intel, 2015.
  - EDK II Platform Configuration Database Infrastructure Description, Intel, 2009.
- INI file, Wikipedia, http://en.wikipedia.org/wiki/INI_file.
- C Now - C Programming Information, Langston University, Tulsa Oklahoma, J.H. Young, 1999-2011, http://c.comsci.us/syntax/expression/ebnf.html.

1.4 Target Audience

Those performing UEFI development and support for platforms and distributable modules.

1.5 Conventions Used in this Document

This document uses typographic and illustrative conventions described below.

1.5.1 Data Structure Descriptions

Intel(R) processors based on 32 bit Intel(R) architecture (IA 32) are "little endian" machines. This distinction means that the low-order byte of a multi byte data item in memory is at the lowest address, while the high-order byte is at the highest address. Processors of the Intel(R) Itanium(R) 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:

Summary

A brief description of the data structure.

Prototype

An EBNF-type declaration for the data structure.

Parameters

Explanation of some terms used in the prototype.

Example

Sample data structure using the prototype.

1.5.2 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 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 UEFI Specification.

1.5.3 Typographic Conventions

This document uses the typographic and illustrative conventions described below:

Typographic Convention	Typographic convention description
Plain text	The normal text typeface is used for the vast majority of the descriptive text in a specification.

Plain text (blue)	Any plain text that is underlined and in blue indicates an active link to the crossreference. Click on the word to follow the hyperlink.
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`	Words in a `Bold Monospace` typeface that is underlined and in blue indicate an active hyper link to the code definition for that function or type definition. Click on the word to follow the hyper link.
`$(VAR)`	This symbol VAR defined by the utility or input files.
Italic Bold	In code or in text, words in ***Italic Bold*** indicate placeholder names for variable information that must be supplied (i.e., arguments).

Note: Due to management and file size considerations, only the first occurrence of the reference on each page is an active link. Subsequent references on the same page will not be actively linked to the definition and will use the standard, non-underlined **BOLD Monospace** typeface. Find the first instance of the name (in the underlined **BOLD Monospace** typeface) on the page and click on the word to jump to the function or type definition.

The following typographic conventions are used in this document to illustrate the Extended Backus-Naur Form.

[item]	Square brackets denote the enclosed item is optional.
`{item}`	Curly braces denote a choice or selection item, only one of which may occur on a given line.
`<item>`	Angle brackets denote a name for an item.
`(range-range)`	Parenthesis with characters and dash characters denote ranges of values, for example, (a-zA-Z0-9) indicates a single alphanumeric character, while (0-9) indicates a single digit.
"item"	Characters within quotation marks are the exact content of an item, as they must appear in the output text file.
`?`	The question mark denotes zero or one occurrences of an item.
`*`	The star character denotes zero or more occurrences of an item.
`+`	The plus character denotes one or more occurrences of an item.
`item{n}`	A superscript number, n, is the number occurrences of the item that must be used. Example: (0-9)8 indicates that there must be exactly eight digits, so 01234567 is valid, while 1234567 is not valid.
`item{n,}`	A superscript number, n, within curly braces followed by a comma "," indicates the minimum number of occurrences of the item, with no maximum number of occurrences.
`item{,n}`	A superscript number, n, within curly brackets, preceded by a comma ","indicates a maximum number of occurrences of the item.
`item{n,m}`	A super script number, n, followed by a comma "," and a number, m, indicates that the number of occurrences can be from n to m occurrences of the item, inclusive.

2 INF OVERVIEW

This section of the document describes the decisions regarding the format of the EDK II module INF files. The INF files are used by EDK II utilities that parse build meta-data files (INF, DEC, DSC and FDF files) to generate `AutoGen.c` and `AutoGen.h` and Makefile/GNU makefile files for the EDK II build infrastructure.

The EDK II INF meta-data file describes properties of a module, how it is coded, what it provides, what it depends on, architecture specific items, features, etc. regarding the module. INF files generated during a build (that allow distribution of binary modules) describe how the module was compiled, linked and what platform configuration database items (PCDs) are exposed. Binary distribution of EDK II modules allows original device manufactures (ODMs) to distribute proprietary drivers, without distributing source code, for inclusion in a firmware image.

EDK II modules may be located in sub-directories of a package (a collection of related objects.) If a module is "a Library", creating the module directory in the "Library" subdirectory of a package is strongly recommended. An "Include" package subdirectory may also be required. Header files for modules that define a library class must be placed in the Include/Library directory using the Library Class Name for the file name.

The Include directory and sub-directories contain header files that define either a library class API or pre-defined ("industry standard") data elements. One and only one header file defines the library class API. Multiple library instances can "produce" the functionality of a library class. The use of library class API headers allows for platform integrators to select a library instance that is suitable for their platform. This usage model frees the driver developer from coding a module to specific library instances. Libraries are really nothing more than modules with pre-defined APIs.

Each module may have one or more INF files that can be used by tools to generate images. Specifically, the EDK Compatibility Package will contain two INF files for any module that contains assembly code. Since the ECP can be used with existing EDK tools (which is only supported by Microsoft and Intel Windows based tools,) a separate INF file to support the multiple tool chain capability of the EDK II build system must be provided for the modules that contain assembly code. The EDK II ECP will use the `basename_edk2.inf` for the filename of the EDK II build system compatible INF files for non-Windows based tool chains, and use just the `basename.inf` for the filename of EDK only INF files used by the EDK build system.

Note: Path and Filename elements within the INF are case-sensitive in order to support building on UNIX style operating systems.

Note: GUID values are used during runtime to uniquely map the C names of PROTOCOLS, PPIS, PCDS and other variable names.

Note: This document uses a backslash "\" to indicate that a line that cannot be displayed in this document on a single line. Within the DSC specification, each entry must appear on a single line.

Note: The total path and file name length is limited by the operating system and third party tools. It is recommended that for EDK II builds that the WORKSPACE directory be either a directory under a subst drive in Windows (s:/build as an example) or be located in either the /opt directory or in the user's

/home/username directory for Linux and OS/X.

2.1 Processing Overview

Each module or component INF file is broken out into sections in a manner similar to the other build meta-data files. However, the intent of a module's INF file is to define the source files, libraries, and definitions relevant to building the module, creating binary files that are either raw binary files or PE32/PE32+/coff format files. The different sections are described in detail in this chapter. In general, the original EDK parsing utilities read each line from the `[Libraries]` or `[Components]` sections of the build description (DSC) file, process the INF file on a line to generate a makefile, and then continued with the next line. EDK II parsing utilities are token based, which permits an element to span multiple lines. The EDK II utilities check both EDK and EDK II INF files, and, if required, generate C code files based on the content of the EDK II INF. Refer to the EDK II Build Specification for more information regarding these autogenerated files.

One major difference between EDK and EDK II is support for non-Microsoft development environments. Because modules may be distributed to developers that use these environments, both source code and the meta-data files need to be UNIX*/ GCC clean. One little known fact regarding the Microsoft tools and operating systems is their ability to process the forward slash "/" character as a directory separator.

All EDK II INF files MUST use this forward slash character for all directory paths specified.

2.2 Information File General Rules

This section covers the format for the EDK II module INF files. While the EDK code base and tools treated libraries completely separate from modules, the EDK II code base and tools process modules, with libraries being considered a module that produces a library class.

2.2.1 Section Entries

To simplify parsing, the EDK II meta-data files continue using the INI format. This style was introduced for EDK meta-data files, when only the Windows tool chains were supported. It was decided that for compatibility purposes, that INI format would continue to be used. EDK II formats differ from the defacto format in that the semicolon ";" character cannot be used to indicate a comment.

Leading and trailing space/tab characters must be ignored.

Duplicate section names must be merged by tools.

This description file consists of sections delineated by section tags enclosed within square `[]` brackets. Section tag entries are case-insensitive. The different sections and their usage are described below. The text of a given section can be used for multiple section names by separating the section names with a comma. For example:

`[Sources.X64, Sources.IPF]`

The content below each section heading is processed by the parsing utilities in the order that they occur in the file. The precedence for processing these architecture section tags is from right to left, with sections defining an architecture having a higher precedence than a section which uses common (or no architecture extension) as the architecture.

Note: Content within a section IS case sensitive. IA32, Ia32 and ia32 within a section are processed as separate items. (Refer to Naming Conventions below for more information on directory and/or file naming.)

Sections are terminated by the start of another section or the end of the file.

Duplicate sections (two sections with identical section tags) will be merged by tools, with the second section appended to the first.

If architectural modifiers are used in the section tag, the section is merged by tools with content from common sections (if specified) with the architectural section appended to the first, into an architectural section. For example, given the following:

```
[Sources]
  ACommonFile.c

[Sources.IA32]
  BforIa32.c

[Sources.X64]
  CforX64.c
```

After the first pass of the tools, when building the module for IA32, the source files will logically be:

```
[Sources.IA32]
  ACommonFile.c
  BforIa32.c
```

When building the module for X64, the source files will logically be:

```
[Sources.X64]
  ACommonFile.c
  CforX64.c
```

The `[Defines]` section tag prohibits use of architectural modifiers. All other sections can specify architectural modifiers.

2.2.2 Comments

The hash `#` character indicates comments in the Module Information (INF) file. In line comments terminate the processing of a line. In line comments must be placed at the end of the line, and may not be placed within the section (`[` , `]`) tags.

Only `gPkgTSGuid.PcdFoo|TRUE|BOOLEAN|0x00000015` in the following example is processed by tools; the remainder of the line is ignored:

`gPkgTSGuid.PcdFoo|TRUE|BOOLEAN|0x00000015 # EFI_FOO_MEMORY`

Note: Blank lines and lines that start with the hash # character must be ignored by build tools.

Hash characters appearing within a quoted string are permitted, with the string being processed as a single entity. The following example must handle the quoted string as single element by tools.

`UI = "# Copyright 2007, No Such, LTD. All rights reserved."`

Comments are terminated by the end of line.

2.2.3 Valid Entries

Processing of the line is terminated if a comment is encountered.

Processing of a line is terminated by the end of the line.

Items in quotation marks are treated as a single token and have the highest precedence. All expressions must be written using in-fix notation (operators are written between the operands). Parenthesis surrounding groups of operands and operators must be used to determine the order in which operations are to be performed. All other processing occurs from left to right.

In the following example, B - C is processed first, then result is added to A followed by adding 2; finally 3 is added to the result.

(A + (B - C) + 2) + 3

In the next example, A + B is processed first, then C + D is processed and finally the two results are added.

(A + B) + (C + D)

Space and tab characters are permitted around field separators.

2.2.4 Naming Conventions

The EDK II build infrastructure is supported under Microsoft Windows, Linux* and MAC OS/X operating systems. All directory and file names must be treated as case sensitive because of multiple environment support.

- The use of special characters in directory names and file names is restricted to the dash, underscore, and period characters, respectively "-", "_", and ".".
- Period characters must not be followed by another period character. File and Directory names must not start with "./", "." or "..".
- Space characters must never be used in the directory path specified by the system environment variable, `WORKSPACE`.
- Directory names and file names within the `WORKSPACE` directory tree must not contain space characters.
- Directory Names must only contain alphanumeric, dash, underscore and period characters (two sequential period characters, ".." are not permitted); it is recommended that the name start with an alpha character.
- All files (except those listed in the Packages sections) must reside in the directory containing the INF file or in sub-directories of the directory containing the INF file.
- Additionally, all EDK II directories that are architecturally dependent must use a name with only the first character capitalized. Ia32, Ipf, X64 and Ebc are valid architectural directory names. IA32, IPF and EBC are not acceptable directory names, and may cause build breaks. From a build tools perspective, IA32 is not equivalent to Ia32 or ia32.
- Absolute paths are not permitted in EDK II INF files. All paths specified are relative to an EDK II package directory (defined as a directory containing a DEC file) or relative to the directory containing the INF file.

The build tools must be able to process the tool definitions file: `tools_def.txt` (describing the location and flags for compiler and user defined tools), which may contain space characters in paths on Windows* systems.

Note: The toolsdef.txt file and `[BuildOptions]` sections are the places that permit the use of space characters in a directory path.

The EDK II Coding Style specification covers naming conventions for use within C Code files, and as well as specifying the rules for directory and file names. This section is meant to highlight those rules as they apply to the content of the INF files.

Architecture keywords (`IA32` , `IPF` , `X64` and `EBC`) are used by build tools and in metadata files for describing alternate threads for processing of files. These keywords must not be used for describing directory paths. Additionally, directory names with architectural names (Ia32, Ipf, X64 and Ebc) do not automatically cause the build tools or meta-data files to follow these alternate paths. Directories and Architectural Keywords are similar in name only.

All directory paths within EDK II INF files must use the forward slash "/" character to separate directories as well as directories from filenames. Example:

`C:/Work/Edk2/edksetup.bat`

File names must also follow the same naming convention required for directories. No space characters are permitted. The special characters permitted in directory names are the only special characters permitted in file names.

2.2.5 !include Statements

The `!include` statement are NOT permitted in the INF files.

2.2.6 Macro Statements

Use of MACRO statements in the EDK II INF files is limited to local usage only; global or external macros are not permitted. This decision was made in order to support UEFI's PI Distribution Package Specification requirements.

Macro statements are permitted in the EDK II INF files. Macro statements assign a Value to a Variable Name, and are only valid during the processing of the INF specifying the value. If a value is not specified, then the MACRO has a value of zero.

Token names (reserved words defined in the EDK II meta-data file specifications) cannot be used as macro names. As an example, using PLATFORM_NAME as a macro name is not permitted, as it is a token defined in the DSC file's `[Defines]` section.

Any defined MACRO definitions will be expanded by tools when they encounter the entry in the section except when the macro is within double quotation marks in build options sections. The expectation is that these macros will be expanded by scripting tools such as make or nmake.

Macros can be used to define a path, a filename, any combination of path and file names or content that will appear in the right side of a statement in the `[BuildOptions]` section. Macros for paths and files can be defined and used in `[Defines]` , `[LibraryClasses]` , `[Sources]` , `[Binaries]` , and `[Packages]` sections.

Macro Definition statements that appear within a section of the file (other than the `[Defines]` section) are scoped to the section they are defined in. If the Macro statement is within the `[Defines]` section, then the Macro is common to the entire file, with local definitions taking precedence (if the same MACRO name is redefined in subsequent sections, then the MACRO value is local to only that section.)

In the following example, the MACRO, IFMP is used to fit a long directory/filename pair on to a single line.:

`DEFINE IFMP = IntelFrameworkModulePackage`

Using the macro, for example, in a `[Packages]` section, looks like:

`$(IFMP)/IntelFrameworkModulePackage.dec`

Macros are evaluated where they are used in statements, not where they are defined. It is recommended that tools break the build and report an error if an expression cannot be evaluated.

Macros used in build flags (in `[BuildOptions]` sections) that are encapsulated by quotation marks are not expanded by tools, and do not need to be local to the INF file. The expectation is that macros in the quoted values will be expanded by external build scripting tools, such as `nmake` or `gmake` ; they will not be expanded by the build tools.

The macro statements are positional, in that only statements following a macro definition are permitted to use the macro - a macro cannot be used before it has been defined.

Macros defined in common sections may be used in the architecturally modified sections of the same section type. Macros defined in architectural sections cannot be used in other architectural sections, nor can they be used in the common section. Section modifiers in addition to the architectural modifier follow the same rules as architectural modifiers.

Within the EDK II INF File, macros are expanded (except within quotes), not evaluated, during the parsing of the file.

Example

```
[LibraryClasses.common]
```

```
  DEFINE MDE = MdePkg/Library
  BaseLib|$(MDE)/BaseLib.inf

[LibraryClasses.X64, LibraryClasses.IA32]
  # Can use $(MDE), cannot use $(MDEMEM)
  DEFINE PERF = PerformancePkg/Library
  TimerLib|$(PERF)/DxeTscTimerLib/DxeTscTimerLib.inf

[LibraryClasses.X64.PEIM]
  # Can use $(MDE) and $(PERF)
  DEFINE MDEMEM = $(MDE)/PeiMemoryAllocationLib
  MemoryAllocationLib|$(MDEMEM)/PeiMemoryAllocationLib.inf

[LibraryClasses.IPF]
  # Cannot use $(PERF) or $(MDEMEM)
  # Can use $(MDE) from the common section
  PalLib|$(MDE)/UefiPalLib/UefiPalLib.inf
  TimerLib|$(MDE)/BaseTimerLibNullTemplate/BaseTimerLibNullTemplate.inf
```

In the previous example, the directory and filename for a library instance is the recommended instance and may not be the actual library linked to the module, as the platform integrator may choose a different library instance to satisfy a library class dependency.

2.2.7 Conditional Directive Statements (!if...)

Conditional statements are NOT permitted in the EDK II INF files.

2.2.8 Expressions

Expressions are supported in specific statements within the EDK II INF files. The expression syntax is defined in the EDK II Expression Syntax Specification.

2.3 EDK II INF Format

The remainder of this chapter describes the EDK II INF file format.

Note: EDK II accommodates distribution of binary modules, so in addition to handling standard module builds, the INF can also specify information about a binary module.

EDK II INF files may be created by package installation tools using the UEFI Distribution Package description files that accompany a distribution package.

All content (except section tag names) within the EDK II INF file is case-sensitive. All newly created EDK II INF files must be written to be case-sensitive.

2.4 [Defines] Section

This is a required section.

The `[Defines]` section of EDK II INF files is used to define variable assignments that can be used in later build steps. The INF_VERSION of existing INF files does not need to be updated unless content in the file has been updated to match new content specified by this revision of the specification.

Architectural modifiers are not permitted in the [Defines] section.

The parsing utilities process any local symbol assignments defined in this section. The

EDK II parsing utilities will use some of this section's information for generating `AutoGen.c` and `AutoGen.h` files. Note that the sections are processed in the order listed in the INF file, and later assignments of these local symbols override previous assignments.

`[Defines]`

The format for entries in this section is:

`Name = Value`

The following is an example of a driver's `[Defines]` section.

```
[Defines]
  INF_VERSION     = 0x0001001B
  BASE_NAME       = DxeIpl
  FILE_GUID       = 86D70125-BAA3-4296-A62F-602BEBBB9081
  VERSION_STRING  = 1.0
  MODULE_TYPE     = PEIM
  ENTRY_POINT     = PeimInitializeDxeIpl
  MODULE_UNI_FILE = DxeIpl.uni
```

The following is an example of a library's `[Defines]` section.

```
[Defines]
  INF_VERSION    = 1.27
  BASE_NAME      = BaseMemoryLib
  FILE_GUID      = fd44e603-002a-4b29-9f5f-529e815b6165
  MODULE_TYPE    = BASE
  VERSION_STRING = 1.0
  LIBRARY_CLASS  = BaseMemoryLib
```

Drivers may expose library functionality, such as a `DXE_CORE` module that may implement functions that satisfy the BaseMemoryAllocation library class. In this instance, the driver module would also specify the `LIBRARY_CLASS` in the `[Defines]` section. Other DXE drivers that would require a library instance for the `BaseMemoryAllocation` class could specify the `DXE_CORE` INF file as the recommended instance for satisfying the required library class instance.

Appendix G lists the available `MODULE_TYPE` values supported by EDK II INF files.

The EDK II `[Defines]` section is common to all architectures and does not permit using architectural modifiers in the section tag name.

The following table shows EDK II unique elements of a defines section that may be required for generating the `AutoGen.c` and `AutoGen.h` files. Library modules must never specify driver elements.

Note: Any lines not starting with one of the tag names defined in the table below are added to the top of the INF's generated makefile exactly as typed on the line in the INF file.

Note: `COMBINED_PEIM_DRIVER` is a driver that may be dispatched by either the PEI Core or the Dxe Core. EDK II only references the first possible dispatch instance.

Table 1 EDK II [Defines] Section Elements

Tag	Required	Value	Notes
`INF_VERSION`	REQUIRED	1.27 or 0x0001001B	This identifies the INF spec. version. It is decimal value with fraction or two-nibble hexadecimal representation of the same, for example: 1.27. Tools use this value to handle parsing of previous releases of the specification if there are incompatible changes.
`BASE_NAME`	REQUIRED	A single word	This is a single word identifier that will be used for the component name.
`EDK_RELEASE_VERSION`	Not required	Hex Double Word	The minimum revision value across the module and all its dependent libraries. If a revision value is not declared in the module or any of the dependent libraries, then the tool may use the value of 0, which disables checking.
`PI_SPECIFICATION_VERSION`	Not required	Decimal or special format of hex	The minimum revision value across the module and all its dependent libraries. If a revision value is not declared in the module or any of the dependent libraries, then tools may use the value of 0, which disables checking.
			The `PI_SPECIFICATION_VERSION` must only be set in the INF file if the module depends on services or system table fields or PI core behaviors that are not present in the PI 1.0 version. For example, if a module depends on definitions in PI 1.1 that are not in PI 1.0, then `PI_SPECIFICATION_VERSION` must be 0x0001000A
`UEFI_SPECIFICATION_VERSION`	Not required	Decimal or special format of hex	The minimum revision value across the module and all its dependent libraries. If a revision value is not declared in the module or any of the dependent libraries, then tools may use the value of 0, which disables checking.
			The `UEFI_SPECIFICATION_VERSIOn` must only be set in the INF file if the module depends on UEFI Boot Services or UEFI Runtime Services or UEFI System Table fields or UEFI core behaviors that are not present in the UEFI 2.1 version. For example, if a module depends on definitions in UEFI 2.2 that are not in UEFI 2.1, then `UEFI_SPECIFICATION_VERSION` must be 0x00020014
`FILE_GUID`	REQUIRED	GUID Value	Registry (8-4-4-4-12) Format This value is required for all EDK II format INF files, required for EDK driver INF files, not required for EDK libraries

`MODULE_TYPE`	REQUIRED		This is the type of module. One of the EDK II Module Types. For Library Modules, the `MODULE_TYPE` must specify the `MODULE_TYPE` of the module that will use the driver.
`BUILD_NUMBER`	Optional	UINT16 Value	This optional element, if present (or set in the DSC file), is used during the creation of the `EFI_VERSION_SECTION` for this module; if it is not present, then the BuildNumber field of the `EFI_VERSION_SECTION` will be set to 0.
`VERSION_STRING`	REQUIRED	String	If present, this value will be encoded as USC-2 characters in a Unicode file for the VERSION section of the FFS unless a ver or ver_ui file has been specified in the `[Binaries]` section.
`MODULE_UNI_FILE`	Optional	Filename	A Unicode file containing UCS-2 character localization strings; the file path (if present) is relative to the directory containing the INF file. The use of #include statements in this file is prohibited.
`LIBRARY_CLASS`	Typically not specified for a Driver; REQUIRED for a Library Only Module	Word | List ["|" Word | List]*	One Library Class that is satisfied by this Library Instance; one or more `LIBRARY_CLASS` lines may be specified by a module. The reserved keyword, NULL, must be listed for library class instances that do NOT support a library class keyword.
`PCD_IS_DRIVER`	Not required - Driver Only	PEI_PCD_DRIVER or DXE_PCD_DRIVER	Only required for the two (PEI_PCD_DRIVER or DXE_PCD_DRIVER) PCD Driver modules.
`ENTRY_POINT`	Not required - Driver Only	CName	This is the name of the driver's entry point function.
`UNLOAD_IMAGE`	Not required - Driver Only	CName	If a driver chooses to be unloadable, then this is the name of the module's function registered in the Loaded Image Protocol. It is called if the UEFI Boot Service UnloadImage() is called for the module, which then executes the Unload function, disconnecting itself from handles in the database as well as uninstalling any protocols that were installed in the driver entry point. The CName is the name of this module's unload function.
`CONSTRUCTOR`	Not required - Library Only	CName	This only applies to components that are libraries. It is required for EDK II libraries if the module's INF contains a Constructor element. This value is used to call the specified function before calling into the library itself.
`DESTRUCTOR`	Not required - Library Only	CName	This only applies to components that are libraries. This value is used to call the specified function before calling into the library itself.

`SHADOW`	Not required - SEC, PEIM and PEI_CORE Driver modules only	TRUE | FALSE	This boolean operator is used by `SEC` , `PEI_CORE` and `PEIM` modules to indicate if the module was coded to use `REGISTER_FOR_SHADOW` . If the value is TRUE, the .reloc section of the PE32 image is not removed, otherwise, the .reloc section is stripped to conserve space in the final binary images. The default value is FALSE.
`PCI_DEVICE_ID`	Not required - Required for UEFI PCI Option ROMs	Hex Number	The PCI Device Id for this device.
`PCI_VENDOR_ID`	Not required - Required for UEFI PCI Option ROMs	Hex Number	The PCI Vendor Id for this device
`PCI_CLASS_CODE`	Not required - Required for UEFI PCI Option ROMs	Hex Number	The PCI Class Code for this device
`PCI_COMPRESS`	Not required UEFI PCI Option ROMs	TRUE | FALSE	This flag is used by tools to compress a PCI Option ROM image file, the default (if not specified) is FALSE
`UEFI_HII_RESOURCE_SECTION`	Not required Driver Only	TRUE | FALSE	This boolean operator is used to indicate that the module will require a separate HII resource section in the efi image file.
`DEFINE`	Not required	Name = Value	The value must be a directory name, and the name can be used with $(and) character sets. This allows shortening of lines typed by users.
`SPEC`	Not required	CName = Value	A User-specified #define CName Value pair that will be included in the `AutoGen.h` file.
`CUSTOM_MAKEFILE`	Not required	Family | File	A user written makefile that will be used, the INF file will not be parsed. The Family is one of MSFT or GCC followed by a field separator "|" character, then the filename of the makefile in the same directory as the INF file. To keep GCC compatibility, the user must generate two Makefiles, one for MSFT, such as makefile and another for GCC, such as GNUmakefile
`DPX_SOURCE`	Not Required Driver Only	Filename	If present, the file must contain all DEPEX statements (as defined in the UEFI PI specification), as the tools will process the file, ignoring any content in `[Depex]` sections in this file AND all inherited dependencies from libraries. This allows the module owner to force a Depex independently. Use of this option is not recommended for normal use.

2.5 [Sources] Section

The `[Sources]` section is used to specify the files that make up the component. Directories names are required for files existing in subdirectories of the component. All directory names are relative to the location of the INF file. Each file is added to the macro of `$(INC_DEPS)`, which can be used in a makefile dependency expression.

Binary files must not be listed in this section. EDK II INF files may have a `[Binaries]` section defined that must be used to define the type and name of the binary files provided by a module.

This section is optional. If it is present, and files are listed in this section, then the build tools must process the files for AutoGen as well as `Makefile` generation. If this section is not present, then the build tools may assume that the binary files listed in the `[Binaries]` section have already been processed by the first build step - no AutoGen or Makefiles need to be generated.

If both `[Sources]` and `[Binaries]` sections are specified, the build tools assume that the code provided in the sources section must be built, and that the binary files provided are also required for the final image generation process steps.

Files listed in architectural specific sections must not be listed in common architecture `[Sources]` sections. The architectural modifier is used to specify additional files that are required over and above the non-architectural specific content. During builds, files are grouped by tools using the common and architecturally specified sections.

This section will typically use one of the following section definitions:

```
[Sources]
[Sources.common]
[Sources.IA32]
[Sources.X64]
[Sources.IPF]
[Sources.EBC]
```

The formats for entries in this section are:

```
Relative/path/and/filename.ext
Filename.ext
```

The following is an example for sources sections.

```
[Sources.common]
  DxeIpl.dxs
  DxeIpl.h
  DxeLoad.c

[Sources.Ia32]
  Ia32/VirtualMemory.h
  Ia32/VirtualMemory.c
  Ia32/DxeLoadFunc.c Ia32/ImageRead.c

[Sources.X64]
  X64/DxeLoadFunc.c

[Sources.IPF]
  Ipf/DxeLoadFunc.c
  Ipf/ImageRead.c
```

All Unicode files must be listed in the source section. If a Unicode file, `A.uni`, has the statement: `#include B.uni`, and `B.uni` has a statement: `#include C.uni`, both `B.uni` and `C.uni` files must be listed in the INF `[Sources]` section in addition to the `A.uni` file.

Specifying a file in an architectural section and in the common architecture section is prohibited (a file cannot be specific to a single architecture and also be general for all architectures).

2.6 [BuildOptions] Section

Content in the `[BuildOptions]` section defines module specific tool chain flags that must be used as the default flags for a module. These flags are appended to any standard flags that are defined by the build process. In order to replace the standard flags that are defined by the build process, an alternate assignment operator must be used; "==" is used for replacement, while "=" is used to append the flag lines. Flags specified in this section can either be appended to the standard flags (defined in the `Conf/tools_def.txt`) or replace the standard flags. In addition to flags, other tool attributes may have the item either appended or replaced.

The left side content of a statement may appear in both common and architectural sections. For example, `MSFT:DEBUG_*_*_CC_FLAGS` may be listed in a common section, while `MSFT:DEBUG_*_IA32_CC_FLAGS` may be listed in the architectural section. If the operator is a single "=" character, the flags from the architectural section are appended to the flags from the common section. Using this section may limit the ability of a module to be compiled with different tool chains or with different build systems and is therefore, discouraged.

Valid content is within this section is limited to the following description.

Table 2 EDK II [BuildOptions] Section Elements

Tag	Value	Notes
`${FAMILY}:${TARGET}_${TAGNAME}_${ARCH}_${TOOLCODE}_FLAGS`	Flags for specific tool codes for this module	Used to specify module specific flags of the module that will use the driver.
`${FAMILY}:${TARGET}_${TAGNAME}_${ARCH}_${TOOLCODE}_PATH`	The fully qualified path an executable	Used to replace a specific command, such as forcing the ASL to be iasl, instead of asl.
`${FAMILY}:${TARGET}_${TAGNAME}_${ARCH}_${TOOLCODE}_DPATH`	A fully qualified path	A path that will be added to the system Environment's PATH variable prior to executing a command
`${FAMILY}:${TARGET}_${TAGNAME}_${ARCH}_${TOOLCODE}_${ATTRIBUTE}`	Attribute specific string	This permits overriding other attributes if required.

In this section, the following table describes each of the variables that are shown above.

Table 3 EDK II [BuildOptions] Variable Descriptions

Variable	Required	Wildcard	Source
`FAMILY`	NO	No	`Conf/tools_def.txt` defines the `FAMILY` values, for example: `MSFT`, `INTEL` or `GCC`. Typically, this field is used to help the build tools determine whether the line is used for Microsoft style Makefiles or the GNU style Makefiles.
			By not specifying the `FAMILY`, the tools assume the flags are applicable to all families.
`TARGET`	YES	Yes = *	`Conf/tools_def.txt` file defines two values: `DEBUG` and `RELEASE`. Developers may define additional targets.
`TAGNAME`	YES	Yes = *	`Conf/tools_def.txt` file defines several different tag names - these are defined by developers; the default tag name, `MYTOOLS`, is provided in the template for tools_def.txt and set in the `Conf/target.txt` file.
`ARCH`	YES	Yes = *	`Conf/tools_def.txt` defines at least four architectures: `IA32`, `X64`, `IPF` and `EBC`. This tag must use all capital letters for the tag. Additional Architectures, such as PPC or ARM may be added as support becomes available.

`TOOLCODE`	YES	NO	The tool code must be one of the defined tool codes in the `Conf/tools_def.txt` file. The flags defined in this section are appended to flags defined in the `tools_def.txt` file for individual tools.
			EXCEPTION: If the INF `MODULE_TYPE` , defined in the `[Defines]` section is `USER_DEFINED` , then the flags listed in this section are the only flags used for the TOOLCODE command specified in `Conf/ tools_def.txt` .
`ATTRIBUTE`	YES	NO	The attribute must be specific to the tool code and must be a valid attribute handled by the build system.

Note: Regarding the EDK and EDK II distinctions in the table: Many EDK INF files must be processed by the EDK II build system, but no EDK INF specification exists. Therefore, items of this kind are listed in Appendix A for completeness. This limits what can be in an EDK INF file as well.

Developers should use extreme caution when specifying items in this section. The EDK II build is designed to support multiple compilers and tool chains, expecting that code is written in ANSI C. If custom tool flags are required by a module, developers must make sure that all consumers of the module are aware of the specific tools and tag names required.

Note: The lines are shown with the backslash "\" character to indicate a line continuation, they are not allowed in the actual INF file.

```
[BuildOptions.common]
  MSFT:DEBUG_*_IA32_DLINK_FLAGS = /out:"$(BIN_DIR)SecMain.exe"/base:0x10000000 /pdb:"$(BIN_DIR)SecMain.pdb"/LIBPATH:"$(VCINSTA
LLDIR)Lib"/LIBPATH:"$(VCINSTALLDIR)PlatformSdkLib"/NOLOGO /SUBSYSTEM:CONSOLE /NODEFAULTLIB /IGNORE:4086/MAP /OPT:REF /DEBUG /M
ACHINE:I386 /LTCG Kernel32.lib MSVCRTD.lib Gdi32.lib User32.libWinmm.lib
  MSFT:DEBUG_*_IA32_CC_FLAGS    = /nologo /W4 /WX /Gy /c /D UNICODE/D EFI32 /Od /DSTRING_ARRAY_NAME=SecMainStrings /FI$(DEST_D
IR_DEBUG)/AutoGen.h /EHs-c- /GF /Gs8192/Zi /Gm
```

For `[BuildOptions]` sections in the INF file, the entries with a common left side (of the "=") will be either appended or replace previous entries based on the "==" replace or "=" append assignment character sequence. Sections with identical architecture modifiers are appended to each other.

`Common Section + Architectural Section`

Example:

```
[BuildOptions.Common]
  MSFT:*_*_*_CC_FLAGS = /nologo

[BuildOptions.Common]
  MSFT:*_*_*_CC_FLAGS = /Od

[BuildOptions.IA32]
  MSFT:*_*_IA32_CC_FLAGS = /D EFI32
```

For IA32 architecture builds of an EDK II INF file would logically be:

`MSFT:*_*_IA32_CC_FLAGS = /nologo /Od /D EFI32`

For X64 architecture builds of an EDK II INF file would logically be:

```
MSFT:*_*_IA32_CC_FLAGS = /nologo /Od
```

2.7 [Binaries] Section

The `[Binaries]` section is used to specify the binary files that are distributed as part of a Binary Module. The binary files listed are not used by the `$(MAKE)` portion of a platform build, but are used by other tools to generate an image suitable for either an Application, FD or FV. A pipe character "|" is used to separate the fields. If the file is in a sub-directory, then the relative (to the INF file) path must be included as part of the file name. The first field is the FileType, which will let a platform integrator know the provided file's format, while the last three fields are optional. (Three defined targets, NOOPT, DEBUG and RELEASE are provided as part of the EDK II build environment.) The wildcard character, "*", is permitted in the fields.

Additional information, such as what flags were used during the build, can also be added in the comments preceding an entry or in an in-line comment that follows the entry.

Files listed in this section do not require generation of AutoGen or Makefiles during the pre-processing build steps.

It is prohibited to list a file in the "common" architectural section and also in a specific architectural section. Binary files can be common to all architectures or specific to individual architectures, not both. The architectural section modifier is used as a restriction to mask binaries from target architectures that are not applicable. During a build, the tools will group binaries in listed in the common sections with the binaries listed for the architecture needed by the build.

This section uses one of the following section definitions:

```
[Binaries]
[Binaries.common]
[Binaries.IA32]
[Binaries.X64]
[Binaries.IPF]
[Binaries.EBC]
```

The formats for entries in this section are:

```
FileType|Relative/path/and/filename.ext|DEBUG|GCC|UNIXGCC|TRUE
FileType|Filename.ext|*|GCC
FileType|Relative/path/and/filename.ext|RELEASE
FileType|Filename.ext|RELEASE
FileType|Filename.ext
```

The `FileType` falls into one of the following PI-defined types:

GUID

This binary is an `EFI_SECTION_GUID_DEFINED` encapsulation section. The EDK II build system does not support binary files of this type.

ACPI

The binary is ACPI binary code generated from an ACPI compiler. There is not PI defined type for this file, it uses an `EFI_SECTION_RAW` leaf section.

ASL

The binary is an ACPI Table generated from an ACPI compiler. There is no PI defined type for this file, it uses an `EFI_SECTION_RAW` leaf section.

DISPOSABLE

Unlike other file types listed in this section, the file will not be placed in a leaf section of type `EFI_SECTION_DISPOSABLE`, but rather it is a binary file that will be ignored by the build tools. (Useful for distributing PDB files with binary modules.)

UEFI_APP

The binary file is a PE32 UEFI Application which will be placed into an FFS file of type `EFI_FV_FILETYPE_APPLICATION`.

PE32

This binary is an `EFI_SECTION_PE32` leaf section.

PIC

This binary is an `EFI_SECTION_PIC` leaf section.

PEI_DEPEX

This binary is an `EFI_SECTION_PEI_DEPEX` leaf section.

DXE_DEPEX

This binary is an `EFI_SECTION_DXE_DEPEX` leaf section.

SMM_DEPEX

This binary is an `EFI_SECTION_SMM_DEPEX` leaf section.

SUBTYPE_GUID

This binary is an `EFI_SECTION_FREEFORM_SUBTYPE_GUID` leaf section.

TE

This binary is an `EFI_SECTION_TE` leaf section.

UNI_VER

This is a Unicode file that needs to be used to create an `EFI_SECTION_VERSION` leaf section.

VER

This binary is an `EFI_SECTION_VERSION` leaf section.

UNI_UI

This is a Unicode file that needs to be used to create an `EFI_SECTION_USER_INTERFACE` leaf section.

UI

This binary is an `EFI_SECTION_USER_INTERFACE` leaf section.

BIN

This binary is an `EFI_SECTION_RAW` leaf section.

RAW

This binary is an `EFI_FV_FILETYPE_RAW` leaf section.

COMPAT16

This binary is an `EFI_SECTION_COMPATIBILTY16 leaf` section.

FV

This binary is an `EFI_SECTION_FIRMWARE_VOLUME_IMAGE` leaf section.

LIB

This binary is a pre-built library instance that provides the library class defined in the `LIBRARY_CLASS` statement in the `[Defines]` section.

Note: The section names listed above refer to leaf section type values rather than the name of the data structure.

The following are examples of different types of `[Binaries]` sections.

```
[Binaries.common]
  UNI_UI|DxeIpl.ui
  UNI_VER|DxeLoad.ver

[Binaries.IA32]
  DXE_DEPEX|Ia32/DxeIpl.dpx             # MYTOOLS
  PE32|Ia32/DEBUG/DxeIpl.efi|DEBUG      # MYTOOLS
  PE32|Ia32/RELEASE/DxeIpl.efi|RELEASE  # MYTOOLS DISPOSABLE|Ia32/DEBUG/DxeIpl.pdb

[Binaries.X64]
  DXE_DEPEX|X64/DxeIpl.dpx              # MYTOOLS
  PE32|X64/DxeIpl.efi                   # MYTOOLS

[Binaries.IPF]
  DXE_DEPEX|IPF/DxeIpl.dpx              # MYTOOLS
  PE32|Ipf/DxeIpl.efi                   # MYTOOLS
```

2.8 [Includes] Section

Never use an `[Includes]` section for pure EDK II Modules or library instances. All include paths are generated by tools by parsing the package files specified in the `[Packages]` section. This section is not required for binary module INFs.

2.9 [Protocols] Section

The `[Protocols]` section of the EDK II INF file is a list of the global Protocol C Names that are used by the module developer. These C names are used by the parsing utility to lookup the actual GUID value of the PROTOCOL that is located in the EDK II package DEC files, and then emit a data structure to the module's `AutoGen.c` file.

Protocols listed in architectural sections must not be listed in common `[Protocols]` sections. The architectural section modifier is used as a restriction to mask items from architectures that are not applicable.

This section uses one of the following section definitions:

```
[Protocols]
[Protocols.common]
[Protocols.IA32]
[Protocols.X64]
[Protocols.IPF]
[Protocols.EBC]
```

The formats for entries in this section is:

`gEfiProtocolCName [ | FeatureFlagExpression] ## Usage comment`

When a `FeatureFlagExpression` is present, if the expression evaluates to `TRUE`, then the Protocal entry is valid. If the expression evaluates to `FALSE`, then the EDK II build tools must ignore the entry.

The following is an example of the `[Protocols]` section.

```
[Protocols]
  gEfiDecompressProtocolGuid
  gEfiLoadFileProtocolGuid
```

2.10 [Ppis] Section

The `[Ppis]` section of the EDK II INF file is a list of the global PPI C Names that are used by the module developer. These C names are used by the parsing utility to lookup the actual GUID value of the PPI that is located in the EDK II package DEC files, and then emit a data structure to the module's `AutoGen.c` file.

PPIs listed in architectural sections must not be listed in common `[Ppis]` sections. The architectural section modifier is used as a restriction to mask items from architectures that are not applicable.

This section uses one of the following section definitions:

```
[Ppis]
[Ppis.common]
[Ppis.IA32]
[Ppis.X64]
[Ppis.IPF]
[Ppis.EBC]
```

The formats for entries in this section is:

`gEfiPpiCName [ | FeatureFlagExpression ] ## Usage comment`

When a `FeatureFlagExpression` is present, if the expression evaluates to `TRUE`, then the PPI entry is valid. If the expression evaluates to `FALSE`, then the EDK II build tools must ignore the entry.

The following is an example of the `[Ppis]` section.

```
[Ppis]
  gEfiPeiMemoryDiscoveredPpiGuid
  gEfiFindFvPpiGuid
```

2.11 [Guids] Section

The `[Guids]` section of the EDK II INF file is a list of the global GUID C Names that are used by the module, and not already included. These C names are used by the parsing routine to lookup the actual GUID value that is located in the EDK II package DEC files, and then emit a data structure to the module's `AutoGen.c` file.

GUID C names listed in architectural sections must not be listed in common `[Guids]` sections. The architectural section modifier is used as a restriction to mask items from architectures that are not applicable.

This section uses one of the following section definitions:

```
[Guids]
[Guids.common]
[Guids.IA32]
[Guids.X64]
[Guids.IPF]
[Guids.EBC]
```

The formats for entries in this section is:

`gEfiGuidCName [ | Feature FlagExpression ] ## Usage comment`

When a `FeatureFlagExpression` is present, if the expression evaluates to `TRUE`, then the GUID entry is valid. If the expression evaluates to `FALSE`, then the EDK II build tools must ignore the entry.

The following is an example of the `[Guids]` section:

```
[Guids]
  gEfiDebugImageInfoTable
  gEfiHobMemoryAllocModuleGuid
```

2.12 [LibraryClasses] Section

The EDK II INF `[LibraryClasses]` section is used to list the names of the library classes that are required, or optionally required by a component. A library class instance, as specified in the DSC file, will be linked into the component. The Library Class' Recommended Instance path must be a package relative path.

Library classes listed in architectural sections must not be listed in common `[LibraryClasses]` sections. The architectural section modifier is used as a restriction to mask items from architectures that are not applicable.

This section uses one of the following section definitions:

```
[LibraryClasses]
[LibraryClasses.common]
[LibraryClasses.IA32]
[LibraryClasses.X64]
[LibraryClasses.IPF]
[LibraryClasses.EBC]
```

The format for entries in this section is:

`LibraryClassName1 [ | FeatureFlagExpression ]`

When a `FeatureFlagExpression` is present, if the expression evaluates to `TRUE`, then the build tools must ensure that a library class instance has been specified when building this module. If the expression evaluates to `FALSE`, then the EDK II build tools must ignore the entry.

```
LibraryClassName2
LibraryClassName3 ## $(WORKSPACE)/Path/To/RecommendedLibInstanceName.inf
```

The comment, using the double hash "##" marks, specifies the module developer's recommended library instance. This is information that the platform integrator can use to help select a library instance for a given library class during a build. The package developer may also provide a recommended library instance. The defined library instance (defined in a DSC file,) that satisfies a Library Class will be added to the LIBS definition in the output makefile:

`LIBS = $(LIBS) $(LIB_DIR)/{LibInstanceName}`

Note: The above is not the name of the INF file, but the name of the library file that was generated during the instance's compilation. Refer to the EDK II DSC File Specification for rules to select library class instances.

Note: For binary driver or application modules, this is a list of the library instances in comments that were used to create the binary (.efi) executable file.

The following is an example of the library class section.

```
[LibraryClasses]
  MemoryAllocationLib
  BaseMemoryLib
  PeiServicesTablePointerLib
  CustomDecompressLib
  TianoDecompressLib UefiDecompressLib
  EdkPeCoffLoaderLib
  CacheMaintenanceLib
  ReportStatusCodeLib
  PeiServicesLib
  PerformanceLib
  HobLib
  BaseLib
  PeimEntryPoint
  DebugLib
```

2.13 [Packages] Section

The `[Packages]` section lists all of the EDK II declaration files that are used by the component. Data from the INF and the DEC files is used to generate content for the `AutoGen.c` and `AutoGen.h` files.

Packages listed in architectural sections must not be listed in common `[Packages]` sections. The architectural section modifier is used as a restriction to mask items from architectures that are not applicable. The locations of the packages listed in this section will be used in generating include path statements for compiler tool chains. The packages must be listed in the order that resolves any include dependencies.

This section uses one of the following section definitions:

```
[Packages]
[Packages.common]
[Packages.IA32]
[Packages.X64]
[Packages.IPF]
[Packages.EBC]
```

The path must include the DEC file name and the name of the directory that contains the DEC file.

```
MdeModulePkg/MdeModulePkg.dec  # MdeModulePkg
MdePkg/MdePkg.dec              # MdePkg
```

The following is an example of a packages section:

```
[Packages]
  MdeModulePkg/MdeModulePkg.dec
  MdePkg/MdePkg.dec
```

If there are files listed under the `[Sources]` section, then the `MdePkg/MdePkg.dec` file must be specified in this section. The MdePkg contains information that is required by the EDK II build system in order to compile or assemble source files using external compilers or assemblers. When generating the "As Built" binary INF, the tools must include all packages that declare PCDs used by this module.

Binary only INF files must include this section if a `[PatchPcd]` or `[PcdEx]` section contains PCD entries.

2.14 PCD Sections

These sections are used for specifying PCD information and are valid for EDK II modules only. The entries for these sections are looked up from the package declaration files (DEC) for generating the `AutoGen.c` and `AutoGen.h` files.

The PCD's Name (`PcdName`) is defined as PCD Token Space GUID C name and the PCD C name - separated by a period "." character. Unique PCDs are identified using the following format to identify the named PCD:

`PcdTokenSpaceGuidCName.PcdCName`

PCDs listed in architectural sections must not be listed in common architectural sections. It is not possible for a PCD to be valid for only IA32 and also valid for any architecture.

A PCD may be valid for IA32 and X64 and invalid for EBC and IPF usage, so mixing of specific architectural modifiers is permitted.

This section defines how a module has been coded to access a PCD. A PCD can only be accessed using the function defined by the UEFI specification for a single type, therefore, mixing PCD section types is not permitted.

There are five defined PCD types. Do not confuse these types with the data types of the PCDs. The five types are: `FeaturePcd` (in code, identified as `FEATURE_FLAG`), `FixedPcd` (`FIXED_AT_BUILD`), `PatchPcd` (`PATCHABLE_IN_MODULE`) and two dynamic types of PCDs, `Pcd` (`DYNAMIC`) and `PcdEx` (`DYNAMIC_EX`).

The two recommended types that are commonly used in modules are: Feature PCD and the dynamic PCD form. The PCD is used for configuration when the PCD value is produced and consumed by drivers during execution, the value may be user configurable from setup or the value is produced by the platform in a specified area. It is associated with modules that are released in source code. The dynamic form is the most flexible method, as platform integrators may chose a to use a different type (such as fixed) for a given platform without modifying the module's INF file or the code for the module. For modules that will be distributed as binaries, the `PatchPcd` and `PcdEx` are the only supported types.

The `FeaturePcd` is used to enable some code paths; the EDK II build system will generate a `const` statement for these PCDs.

Similar in function, the dynamic `PcdEx` type can be used with modules that are released as binary. However, the access methods for this style prevents using these PCDs as any other PCD type (source code must change in order for a `PcdEx` to be used as a `FixedPcd`).

The `FixedPcd` and `PatchPcd` are static and only the `PatchPcd` can have the value changed in a binary prior to including the module in a firmware image.

The content of this section is the PCD Token Space Guid C Name, followed by a period "." character and then the C name of the PCD. The default value is optional. (See chapter 3, Module Information (INF) Format Specification, later in this document for definition of the content.) Every PCD (`PcdName`) is identified by two parts, the PCD's Token Space Guid C Name and the PCD's C Name. These two items are separated by a period "." character. This section uses one of the following section definitions:

```
[(PcdType)]
[(PcdType).common]
[(PcdType).IA32]
[(PcdType).X64]
[(PcdType).IPF]
[(PcdType).EBC]
```

The required entries for this section are the PCD Token Space Guid C Name's for the PCD that will be looked up by tools from the DEC files, and the PCD's C name - that must be specified in the DEC files to limit accidental duplicate PCD C Name collisions. A default value that the module developer suggests to use for the PCD is optional.

`TokenSpaceGuidCName.PcdCName`

Values of PCDs defined in this file override the default values specified in the EDK II package declaration (DEC) file. The platform integrator can specify values in the DSC and FDF files or on the build command line to override any settings in this file. If a default value is not specified, the build system uses 1) values from the command line, 2) values from the FDF file, 3) values from the DSC file or 4) values from the DEC file.

Expressions, or Feature Flag Expressions, may be used on PCD entry lines.

If there are files listed in a `[Binaries]` section and this is a `PatchPcd` section, and the third field of an entry is a Hex number, `0x00000012`, then the value is an offset into a binary image. The format for this type of entry is:

`PcdName | Value | HexValue`

For all other instances, the format for this type of entry is:

`PcdName | [Value] [ | FeatureFlagExpression]`

When a `FeatureFlagExpression` is present, if the expression evaluates to `TRUE`, then the PCD entry is valid. If the expression evaluates to `FALSE`, then the EDK II build tools must ignore the entry.

2.14.1 FIXED_AT_BUILD

The content for the PCD entry is the PCD's Name (PCD's Token Space Guid C name, followed by a period "." character then the PCD's C name) and an optional Default value. These fields are separated by the pipe "|" character. If a module is coded for only `FIXED_AT_BUILD` PCDs, it can only be used during a build from source files. This section must not be present in an INF file that describes a binary only module. This section uses one of the following section definitions:

```
[FixedPcd]
[FixedPcd.common]
[FixedPcd.IA32]
[FixedPcd.X64]
[FixedPcd.IPF]
[FixedPcd.EBC]
```

The following is an example of the `PCD FIXED_AT_BUILD` type:

```
[FixedPcd.common]
  gEfiMdePkgTokenSpaceGuid.PcdFSBClock|600000000
  gEfiEdkModulePkgTokenSpaceGuid.PcdMaxSizeNonPopulateCapsule
  gEfiEdkModulePkgTokenSpaceGuid.PcdMaxSizePopulateCapsule
```

2.14.2 PATCHABLE_IN_MODULE

The PCD entry content is the PCD's Name (PCD's Token Space Guid C name, followed by a period "." and the PCD's C name) and an optional Default value. This section may be present in INF files that describe a binary only module. This type of PCD is one of the recommended formats for modules that will be distributed in binary format. These fields are separated by the pipe "|" character. This section uses one of the following section definitions:

```
[PatchPcd]
[PatchPcd.IA32]
```

```
[PatchPcd.X64]
[PatchPcd.IPF]
[PatchPcd.EBC]
[PatchPcd.common]
```

The following is an example of the `PCD FIXED_AT_BUILD` type:

```
[PatchPcd.common]
  gEfiGenericPlatformTokenSpaceGuid.PcdFlashNvStorageVariableSize
  gEfiGenericPlatformTokenSpaceGuid.PcdFlashNvStorageVariableBase
```

2.14.3 FEATURE_FLAG

The content for the PCD entry is the PCD's Name (PCD's Token Space Guid C name, followed by a period "." and the PCD's C name) and an optional Default value of either `TRUE` or `FALSE`, `1` or `0`. These fields are separated by the period "|" character. This section must not be present in INF files that describe a binary only module. This section uses one of the following section definitions:

```
[FeaturePcd]
[FeaturePcd.common]
[FeaturePcd.IA32]
[FeaturePcd.X64]
[FeaturePcd.IPF]
[FeaturePcd.EBC]
```

The following is an example of the PCD `FEATURE_FLAG` type:

```
[FeaturePcd.common]
  gEfiEdkModulePkgTokenSpaceGuid.PcdDxeIplSupportCustomDecompress
  gEfiEdkModulePkgTokenSpaceGuid.PcdDxeIplSupportTianoDecompress
  gEfiEdkModulePkgTokenSpaceGuid.PcdDxeIplSupportEfiDecompress
  gEfiEdkModulePkgTokenSpaceGuid.PcdDxeIplBuildShareCodeHobs
```

2.14.4 DYNAMIC

The content for the PCD entry is the PCD's Name (PCD's Token Space Guid C name, followed by a period "." and the PCD's C name) and an optional Default value. These entries are separated by the pipe "|" character. While this section is the recommended method for coding PCD access methods, it must not be present in INF files that describe a binary only module. This section uses one of the following section definitions:

```
[Pcd]
[Pcd.common]
[Pcd.IA32]
[Pcd.X64]
[Pcd.IPF]
[Pcd.EBC]
```

The following is an example of the PCD `DYNAMIC` type:

```
[Pcd.common]
  gEfiGenericPlatformTokenSpaceGuid.PcdFlashNvStorageVariableSize
  gEfiGenericPlatformTokenSpaceGuid.PcdFlashNvStorageVariableBase
```

2.14.5 DYNAMIC_EX

The content for the PCD entry is the PCD's Name (PCD's Token Space Guid C name, followed by a period "." and the PCD's C name) and an optional Default value. These entries are separated by the pipe "|" character. This section may be present in INF files that describe a binary only module. This type of PCD is one of the recommended formats for modules that will be distributed in binary format. This section uses one of the following section definitions:

```
[PcdEx]
[PcdEx.common]
[PcdEx.IA32]
[PcdEx.X64]
[PcdEx.IPF]
[PcdEx.EBC]
```

The following is an example of the PCD `DYNAMIC_EX` type:

```
[PcdEx.common]
  gEfiGenericPlatformTokenSpaceGuid.PcdFlashNvStorageFtwWorkingSize
  gEfiGenericPlatformTokenSpaceGuid.PcdFlashNvStorageFtwWorkingBase
  gEfiGenericPlatformTokenSpaceGuid.PcdFlashNvStorageFtwSpareSize
  gEfiGenericPlatformTokenSpaceGuid.PcdFlashNvStorageFtwSpareBase
```

Note: For binary (.efi) modules, only `PATCHABLE_IN_MODULE` or `DYNAMIC_EX` PCDs may be specified. `FixedPcd`, `DYNAMIC` and `FeaturePcd` sections are not permitted for binary distribution of modules.

2.15 [Depex] Section

The EDK II `[Depex]` section is a replacement for the `DPX_SOURCE` file using in EDK (the file is specified in the nmake section of an EDK INF file.)

This section is used for specifying a `Depex` expression, not a binary file. In the "As Built" INF files, this section contains a comment that lists the full dependency expression, including `Depex` statements AND'd from library instances linked against a module.

Binary .depex files are listed in `[Binaries]` sections of the INF files.

Having a common `[Depex]` section and architectural `[Depex]` sections is prohibited. Having multiple module type modifiers for common and architectural sections is permitted. For example, `[Depex.common.DXE_DRIVER, Depex.common.DXE_RUNTIME_DRIVER]` is valid.

This section can be used with an inheritance from libraries, by supporting logical AND'ing of the different Depex expressions together. Since more than one type of dependency expression may be required for modules DXE/SMM modules, as well as components of type `COMBINED_PEIM_DRIVER` (not supported by the EDK II build system), section modifier tags have been defined. For module types that prohibit the use of a `[Depex]` section, all `[Depex]` sections from library instances must be ignored. These are only required if more than one dependency expression is required for a module.

The format of the depex section tag is:

`Depex[.<Arch>[.<ModuleType>]]`

Additionally, the rules for specifying DEPEX sections are as follows.

- If the Module is a Library, then a `[Depex]` section is optional.
- If the Module is a Library with a MODULE_TYPE of BASE, the generic (i.e., [Depex]) and generic with only architectural modifier entries (i.e., [Depex.IA32]) are not permitted. It is permitted to have a Depex section if one ModuleType modifier is specified (i.e., [Depex.common.PEIM).
- If the ModuleType is `USER_DEFINED`, then a `[Depex]` section is optional. If a PEI, SMM or DXE DEPEX section is required, the user must specify a ModuleType of `PEIM` to generate a `PEI_DEPEX` section, a ModuleType of `DXE_DRIVER` to generate a `DXE_DEPEX` section, or a ModuleType of `DXE_SMM_DRIVER` to generate an `SMM_DEPEX` section.
- If the ModuleType is `SEC`, `UEFI_APPLICATION`, `UEFI_DRIVER`, `PEI_CORE`, `SMM_CORE` or `DXE_CORE`, no `[Depex]` sections are permitted and all library class `[Depex]` sections are ignored.
- Module types `PEIM`, `DXE_DRIVER`, `DXE_RUNTIME_DRIVER`, `DXE_SAL_DRIVER` and `DXE_SMM_DRIVER` require a `[Depex]` section unless the dependencies are specified by a `PEI_DEPEX`, `DXE_DEPEX` or `SMM_DEPEX` in the `[Binaries]` section.

The Depex section headers start with one of the following:

```
[Depex]
[Depex.IA32]
[Depex.X64]
[Depex.IPF]
[Depex.EBC]
[Depex.common]
```

When generating the "As Built" binary INF during a build, the complete dependency expression, including dependencies from library instances, will be listed in comments.

The following are examples of Depex section:

```
[Depex]
  TRUE

[Depex.IA32.DXE_DRIVER, Depex.IA32.DXE_RUNTIME_DRIVER]
  gEfiPcdProtocolGuid
```

2.16 [UserExtensions] Section

The EDK II `[UserExtensions]` sections allow for extending the INF with custom processing. The format for a user extension is:

`[UserExtensions.$(UserID).$(Identifier)]`

Having data elements under the section header is not required.

The EDK II build tools do not use this section. When generating the "As Built" binary INF during a build, this section is copied from the original source INF file if the UserId is "TianoCore". Other `[UserExtensions]` sections will not be copied. The reference tools ignore all content within a `[UserExtensions]` section.

The following is an example of a `[UserExtensions]` section:

```
[UserExtensions.NoSuchCorp."MyProcess_1.2"]
  NoSuch.bat
```

2.16.1 [UserExtensions.TianoCore."ExtraFiles"] Section

The EDK II `[UserExtensions.TianoCore."ExtraFiles"]` section allow for distributing extraneous files that are associated with a module. Files listed in this section are not processed by EDK II build tools. These files must exist in the directory or sub- directories of the directory containing the INF file.

Note: The Intel(R) UEFI Packaging Tool will parse this section and for all files listed in this file, add the file to the module distribution using the UEFI Distribution Package Distribution.

The section header must be:

`[UserExtensions.TianoCore."ExtraFiles"]`

Having data elements under the section header is not required.

The following is an example of a `[UserExtensions.TianoCore."ExtraFiles"]` section:

```
[UserExtensions.TianoCore."ExtraFiles"]
  Readme.txt
```

3 EDK II INF FILE FORMAT

This section of the document describes the EDK II INF sections using an Extended Backus-Naur Form.

Note: The elements of the EDK INF file (see Appendix A) and the EDK II INF files differ.

3.1 General Rules

The general rules for all EDK II INI style documents follow.

Note: Path and Filename elements within the INF are case-sensitive in order to support building on UNIX style operating systems. Additionally, names that are C variables or used as a macro are case sensitive. Other elements such as section tags or hex digits, in the INF file are not case-sensitive. The use of "..", "../" and "./" in paths and filenames is strictly prohibited.

Note: This document uses a backslash "\" character to indicate that a line that cannot be displayed in this document on a single line. Within the INF specification, each entry must appear on a single line.

- Multiple INF files may exist in a directory, however either the `FILE_GUID` or the `VERSION_STRING` must be unique to the INF file. It is recommended that the `BASE_NAME` also be unique (and match the INF filename, as in `BaseLib.inf` will use a `BASE_NAME` of BaseLib).
- Text in section tags (text between square brackets) is not case sensitive.
- A section terminates with either another section definition or the end of the file.
- To append comment information to any item, the comment must start with a hash "#" character.
- All comments terminate with the end of line character.
- Any comment not associated with a defined comment format is considered a global comment.
- Global comments must be separated from formatted comments by a blank line.
- Field separators for lines that contain more than one field is the pipe "|" character. This character was selected to reduce the possibility of having the field separator character appear in a string, such as a filename or text string.

Note: The only notable exception is the PcdName which is a combination of the PcdTokenSpaceGuidCName and the PcdCName that are separated by the period "." character. This notation for a PCD name was used to uniquely identify the PCD.

- A line terminates with either an end of line character or a comment.
- Except for binary "As Built" INF files generated by the tools, when processing numeric values, either integer or hex, leading zeros specified in the entry may be ignored. For example, 0x0000000000000000000001 can be a valid value for a `UINT8` data type, as the actual value is 1 The generated binary "As Built" INF file must use zero byte filled in order to specify a the length of a `VOID` * PCD value.
- Sections with duplicate tags, such as two section tags: `[BuildOptions]` , will be combined by tools, with the second section's content appended to the section content that was first in the file.
- Sections with architectural modifiers are appended by tools to section content with either the "common" or no architectural modifiers if it exists. The combined result is then considered a complete section.

3.1.1 Backslash

The backslash "\" character in this document is only for lines that cannot be displayed within the margins of this document. The backslash character must not be used to extend a line over multiple lines in the INF file.

3.1.2 Whitespace characters

Whitespace (space and tab) characters are permitted between token and field separator elements for all entries.

Whitespace characters are not permitted between the PcdTokenSpaceGuidCName and the dot, nor are they permitted between the dot and the PcdCName.

3.1.3 Paths for File Names

Note that for specifying the path for a file name, if the path value starts with a dollar "$" sign character a local MACRO variable is being specified. White space characters are not permitted in path names.

Note: The use of "..", "./" and "../" in a path element is prohibited.

For all EDK II INF files, the directory path must use the forward slash character for separating directories. (For example, `MdePkg/Include/` should be specified).

Unless otherwise noted, all file names and paths must be relative to the directory where the INF file is located.

3.2 Component INF Definition

The INF definitions describe the content of a module, either source or binary, as well as external dependencies on packages that contain declarations of GUIDs, Protocols, PPIs and Library Classes. The platform integrator will can select library instances that will be used for a given library class, providing greater flexibility to the module developer. It is not possible to code a module to a specific implementation of a library instance. It is only possible to code a module to use a library class. The `[Defines]` section must appear before any other section except the header. (The header, when specified, is always the first section of an INF file.) The remaining sections may be specified in any order within the INF file.

Summary

The EDK II Module Information (INF) file has the following format (using the EBNF).

```
<EDK_II_INF> ::= <Header>?
                 <Defines>
                 <BuildOptions>*
                 <LibraryClasses>*
                 <Packages>*
                 <Pcds>*
                 <Sources>*
                 <Protocols>*
                 <Ppis>*
                 <Guids>*
                 <Binaries>*
                 if (LIBRARY_CLASS is declared in Defines Section):
                   <Depex>*
                 elif (MODULE_TYPE == "USER_DEFINED"
                       || MODULE_TYPE == "UEFI_DRIVER"):
                          <Depex>*
                 elif (MODULE_TYPE == "PEIM"
                       || MODULE_TYPE == "DXE_DRIVER"
                       || MODULE_TYPE == "DXE_RUNTIME_DRIVER"
                       || MODULE_TYPE == "DXE_SAL_DRIVER"
                       || MODULE_TYPE == "DXE_SMM_DRIVER"):
                          <Depex>+
                 elif (MODULE_TYPE == "PEI_CORE"
                      || MODULE_TYPE == "DXE_CORE"
                      || MODULE_TYPE == "SMM_CORE"
                      || MODULE_TYPE == "UEFI_APPLCIATION"):
                         Do not specify a depex section.
                 <UserExtensions>*
```

3.2.1 Common Definitions

Summary

The following are common definitions used by multiple section types.

Prototype

```
<Word>              ::= (a-zA-Z0-9_)(a-zA-Z0-9_-,)* Alphanumeric characters
                        with optional period ".", dash
                        "-" and/or underscore "_" characters. A period
                        character may not be followed by another period
                        character.
                        No whitespace characters are permitted.
<SimpleWord>        ::= (a-zA-Z0-9)(a-zA-Z0-9_-)* A word that cannot contain a
                        period character.
```

```
<ToolWord>           ::= (A-Z)(a-zA-Z0-9)* A word that must start with a
                         capital letter and is allowed to contain additional
                         alphanumeric characters.
                         Whitespace characters are not permitted.
<FileSep>            ::= "/"
<Extension>          ::= (a-zA-Z0-9_-)+ One or more alphanumeric characters.
<File>               ::= <Word> ["." <Extension>]
<PATH>               ::= [<MACROVAL> <FileSep>] <RelativePath>
<RelativePath>       ::= <Word> [<FileSep> <DirName>]*
<DirName>            ::= {<Word>} {<MACROVAL>}
<FullFilename>       ::= <PATH> <FileSep> <File>
<Filename>           ::= [<PATH> <FileSep>] <File>
<Chars>              ::= (a-zA-Z0-9_)
<Digit>              ::= (0-9)
<NonDigit>           ::= (a-zA-Z_)
<Identifier>         ::= <NonDigit> <Chars>*
<CName>              ::= <Identifier> # A valid C variable name.
<AsciiChars>         ::= (0x21 - 0x7E)
<CChars>             ::= [{0x21} {(0x23 - 0x26)} {(0x28 - 0x5B)}
                         {(0x5D - 0x7E)} {<EscapeSequence>}]*
<DblQuote>           ::= 0x22
<SglQuote>           ::= 0x27
<EscapeSequence>     ::= "\" {"n"} {"t"} {"f"} {"r"} {"b"} {"0"} {"\"}
                         {<DblQuote>} {<SglQuote>}
<TabSpace>           ::= {<Tab>} {<Space>}
<TS>                 ::= <TabSpace>*
<MTS>                ::= <TabSpace>+
<Tab>                ::= 0x09
<Space>              ::= 0x20
<CR>                 ::= 0x0D
<LF>                 ::= 0x0A
<CRLF>               ::= <CR> <LF>
<WhiteSpace>         ::= {<TS>} {<CR>} {<LF>} {<CRLF>}
<WS>                 ::= <WhiteSpace>*
<Eq>                 ::= <TS> "=" <TS>
<FieldSeparator>     ::= "|"
<FS>                 ::= <TS> <FieldSeparator> <TS>
<Wildcard>           ::= "*"
<CommaSpace>         ::= "," <Space>*
<Cs>                 ::= "," <Space>*
<AsciiString>        ::= [ <TS>* <AsciiChars>* ]*
<EmptyString>        ::= <DblQuote><DblQuote>
<CFlags>             ::= <AsciiString>
<PrintChars>         ::= {<TS>} {<CChars>}
<QuotedString>       ::= <DblQuote> <PrintChars>* <DblQuote>
<SglQuotedString>    ::= <SglQuote> <PrintChars>* <SglQuote>
<CString>            ::= {<QuotedString>} {<SglQuotedString>}
<NormalizedString>   ::= <DblQuote> [{<Word>} {<Space>}]+ <DblQuote>
<GlobalComment>      ::= <WS> "#" [<AsciiString>] <EOL>+
<Comment>            ::= "#" <AsciiString> <EOL>+
<UnicodeString>      ::= "L" {<QuotedString>} {<SglQuotedString>}
<HexDigit>           ::= (a-fA-F0-9)
<HexByte>            ::= {"0x"} {"0X"} <HexDigit> <HexDigit>
<HexNumber>          ::= {"0x"} {"0X"} <HexDigit>*
<HexVersion>         ::= "0x" <Major> <Minor>
<Major>              ::= <HexDigit>? <HexDigit>? <HexDigit>?
                         <HexDigit>
<Minor>              ::= <HexDigit> <HexDigit> <HexDigit> <HexDigit>
<DecimalVersion>     ::= {"0"} {(0-9) (0-9)*} ["." (0-9)+]
<VersionVal>         ::= {<HexVersion>} {(0-9)+ "." (0-9)+}
<GUID>               ::= {<RegistryFormatGUID>} {<CFormatGUID>}
<RegistryFormatGUID> ::= <RHex8> "-" <RHex4> "-" <RHex4> "-" <RHex4> "-"
                         <RHex12>
<RHex4>              ::= <HexDigit> <HexDigit> <HexDigit> <HexDigit>
<RHex8>              ::= <RHex4> <RHex4>
<RHex12>             ::= <RHex4> <RHex4> <RHex4>
<RawH2>              ::= <HexDigit>? <HexDigit>
<RawH4>              ::= <HexDigit>? <HexDigit>? <HexDigit>? <HexDigit>
<OptRawH4>           ::= <HexDigit>? <HexDigit>? <HexDigit>? <HexDigit>?
<Hex2>               ::= {"0x"} {"0X"} <RawH2>
<Hex4>               ::= {"0x"} {"0X"} <RawH4>
<Hex8>               ::= {"0x"} {"0X"} <OptRawH4> <RawH4>
```

```
<Hex12>              ::= {"0x"} {"0X"} <OptRawH4> <OptRawH4> <RawH4>
<Hex16>              ::= {"0x"} {"0X"} <OptRawH4> <OptRawH4>
                         <OptRawH4> <RawH4>
<CFormatGUID>        ::= "{" <Hex8> <CommaSpace> <Hex4> <CommaSpace>
                         <Hex4> <CommaSpace> "{"
                         <Hex2> <CommaSpace> <Hex2> <CommaSpace>
                         <Hex2> <CommaSpace> <Hex2> <CommaSpace>
                         <Hex2> <CommaSpace> <Hex2> <CommaSpace>
                         <Hex2> <CommaSpace> <Hex2> "}" "}"
<CArray>              ::= "{" {<Nlist>} {<CArray>} "}"
<NList>               ::= <HexByte> [<CommaSpace> <HexByte>]*
<RawData>             ::= <TS> <Number> [<Cs> <Number> [<EOL> <TS>]]*
<Integer>             ::= {(0-9)} {(1-9)(0-9)*}
<Number>              ::= {<Integer>} {<HexNumber>}
<HexNz>               ::= (\x1 - \xFFFFFFFFFFFFFFFF)
<NumNz>               ::= (1-18446744073709551615)
<GZ>                  ::= {<NumNz>} {<HexNz>}
<TRUE>                ::= {"TRUE"} {"true"} {"True"} {"0x1"} {"0x01"} {"1"}
<FALSE>               ::= {"FALSE"} {"false"} {"False"} {"0x0"}
                          {"0x00"} {"0"}
<BoolVal>             ::= {<TRUE>} {<FALSE>}
<BoolType>            ::= {<BoolVal>} {"{"<BoolVal>"}"}
<MACRO>               ::= (A-Z)(A-Z0-9_)*
<MACROVAL>            ::= "$(" <MACRO> ")"
<PcdName>             ::= <TokenSpaceGuidCName> "." <PcdCName>
<PcdCName>            ::= <CName>
<TokenSpaceGuidCName> ::= <CName>
<UINT8>               ::= {"0x"} {"0X"} (\x0 - \xFF)
<UINT16>              ::= {"0x"} {"0X"} (\x0 - \xFFFF)
<UINT32>              ::= {"0x"} {"0X"} (\x0 - \xFFFFFFFF)
<UINT64>              ::= {"0x"} {"0X"} (\x0 - \xFFFFFFFFFFFFFFFF)
<UINT8z>              ::= {"0x"} {"0X"} <HexDigit> <HexDigit>
<UINT16z>             ::= {"0x"} {"0X"} <HexDigit> <HexDigit> <HexDigit>
                          <HexDigit>
<UINT32z>             ::= {"0x"} {"0X"} <HexDigit> <HexDigit>
                          <HexDigit> <HexDigit> <HexDigit> <HexDigit>
                          <HexDigit> <HexDigit>
<UINT64z>             ::= {"0x"} {"0X"} <HexDigit> <HexDigit>
                          <HexDigit> <HexDigit> <HexDigit> <HexDigit>
                          <HexDigit> <HexDigit> <HexDigit> <HexDigit>
                          <HexDigit> <HexDigit> <HexDigit> <HexDigit>
                          <HexDigit> <HexDigit>
<ShortNum>            ::= (0-255)
<IntNum>              ::= (0-65535)
<LongNum>             ::= (0-4294967295)
<LongLongNum>         ::= (0-18446744073709551615)
<ValUint8>            ::= {<ShortNum>} {<UINT8>} {<BoolVal>}
                          {<CString>} {<UnicodeString>}
<ValUint16>           ::= {<IntNum>} {<UINT16>} {<BoolVal>}
                          {<CString>} {<UnicodeString>}
<ValUint32>           ::= {<LongNum>} {<UINT32>} {<BoolVal>}
                          {<CString>} {<UnicodeString>}
<ValUint64>           ::= {<LongLongNum>} {<UINT64>} {<BoolVal>}
                          {<CString>} {<UnicodeString>}
<NumValUint8>         ::= {<ValUint8>} {"{"<ValUint8>"}"}
<NumValUint16>        ::= {<ValUint16>}
                          {"{"<ValUint8> [<CommaSpace> <ValUint8>]*"}"}
<NumValUint32>        ::= {<ValUint32>}
                          {"{"<ValUint8> [<CommaSpace> <ValUint8>]*"}"}
<NumValUint64>        ::= {<ValUint64>}
                          {"{"<ValUint8> [<CommaSpace> <ValUint8>]*"}"}
<StringVal>           ::= {<UnicodeString>} {<CString>} {<Array>}
<Array>               ::= "{" {<Array>} {[<Lable>] <ArrayVal>
                           [<CommaSpace> [<Lable>] <ArrayVal>]* } "}"
<ArrayVal>            ::= {<Num8Array>} {<GuidStr>} {<DevicePath>}
<NonNumType>          ::= {<BoolVal>} {<UnicodeString>} {<CString>}
                          {<Offset>} {<UintMac>}
<Num8Array>           ::= {<NonNumType>} {<ShortNum>} {<UINT8>}
<Num16Array>          ::= {<NonNumType>} {<IntNum>} {<UINT16>}
<Num32Array>          ::= {<NonNumType>} {<LongNum>} {<UINT32>}
<Num64Array>          ::= {<NonNumType>} {<LongLongNum>} {<UINT64>}
<GuidStr>             ::= "GUID(" <GuidVal> ")"
```

```
<GuidVal>            ::= {<DblQuote> <RegistryFormatGUID> <DblQuote>}
                         {<CFormatGUID>} {<CName>}
<DevicePath>         ::= "DEVICE_PATH(" <DevicePathStr> ")"
<DevicePathStr>      ::= A double quoted string that follow the device path
                         as string format defined in UEFI Specification 2.6
                         Section 9.6
<UintMac>            ::= {<Uint8Mac>} {<Uint16Mac>} {<Uint32Mac>} {<Uint64Mac>}
<Uint8Mac>           ::= "UINT8(" <Num8Array> ")"
<Uint16Mac>          ::= "UINT16(" <Num16Array> ")"
<Uint32Mac>          ::= "UINT32(" <Num32Array> ")"
<Uint64Mac>          ::= "UINT64(" <Num64Array> ")"
<Lable>              ::= "LABEL(" <CName> ")"
<Offset>             ::= "OFFSET_OF(" <CName> ")"
<ModuleType>         ::= {"BASE"} {"SEC"} {"PEI_CORE"} {"PEIM"}
                         {"DXE_CORE"} {"DXE_DRIVER"} {"SMM_CORE"}
                         {"DXE_RUNTIME_DRIVER"} {"DXE_SAL_DRIVER"}
                         {"DXE_SMM_DRIVER"} {"UEFI_DRIVER"}
                         {"UEFI_APPLICATION"} {"USER_DEFINED"}
<ModuleTypeList>     ::= <ModuleType> [" " <ModuleType>]*
<IdentifierName>     ::= <TS> {<MACROVAL>} {<PcdName>} <TS>
<Boolean>            ::= {<BoolType>} {<Expression>}
<EOL>                ::= <TS> 0x0D 0x0A
<OA>                 ::= (a-zA-Z)(a-zA-Z0-9)*
<arch>               ::= {"IA32"} {"X64"} {"IPF"} {"EBC"} {<OA>}
<Edk2ModuleType>     ::= {"BASE"} {"SEC"} {"PEI_CORE"} {"PEIM"}
                         {"DXE_CORE"} {"DXE_DRIVER"}
                         {"DXE_SAL_DRIVER"}
                         {"DXE_RUNTIME_DRIVER"}
                         {"SMM_CORE"} {"DXE_SMM_DRIVER"}
                         {"UEFI_DRIVER"} {"UEFI_APPLICATION"}
```

Note: When using CString, UnicodeString or byte array format as UINT8/UINT16/UINT32/UINT64 values, please make sure they fit in the target type's size, otherwise tool would report failure.

Note: LABEL() macro in byte arrays to tag the byte offset of a location in a byte array. OFFSET_OF() macro in byte arrays that returns the byte offset of a LABEL() declared in a byte array.

Note: When using the characters "|" or "||" in an expression, the expression must be encapsulated in open "(" and close ")" parenthesis.

Note: Comments may appear anywhere within a INF file, provided they follow the rules that a comment may not be enclosed within Section headers, and that in line comments must appear at the end of a statement.

Note: At this time, expressions are not supported in INF files.

Parameters

Expression

Expression syntax is defined the EDK II Expression Syntax Specification.

Unicode String

When the `<UnicodeString>` element (these characters are string literals as defined by the C99 specification: L"string"/L'string', not actual Unicode characters) is included in a value, the build tools may be required to expand the ASCII string between the quotation marks into a valid UCS-2 character encoded string. The build tools parser must treat all content between the field separators (excluding white space characters around the field separators) as ASCII literal content when generating the `AutoGen.c` and `AutoGen.h` files.

Comments

Strings that appear in comments may be ignored by the build tools. An ASCII string matching the format of the ASCII string defined by `<UnicodeString>` (L"Foo" for example,) that appears in a comment must never be expanded by any tool.

CFlags

CFlags refers to a string of valid arguments appended to the command line of any third party or provided tool. It is not limited to just a compiler executable tool. MACRO values that appear in quoted strings in CFlags content must not be expanded by parsing tools.

OA

Other Architecture - One or more user defined target architectures, such as ARM or PPC. The architectures listed here must have a corresponding entry in the EDK II meta-data file, `Conf/tools_def.txt`. Only IA32, X64, IPF and EBC are routinely validated.

ExtendedLine

The use of the Extended Line format is prohibited.

FileSep

FileSep refers to either the backslash "\" or forward slash "/" characters that are used to separate directory names. All EDK II INF files must use the "/" forward slash character when specifying the directory portion of a filename. Microsoft operating systems, that normally use a backslash character for separating directory names, will interpret the forward slash character correctly. Use of "..", "." and "../" in the directory path is not permitted. Use of an absolute path is not permitted.

CArray

All C data arrays used in PCD value fields must be byte arrays. The C format GUID style is a special case that is permitted in some fields that use the `<CArray>` nomenclature.

End of Line Characters

The DOS End Of Line: "0x0D 0x0A" character must be used for all EDK II metadata files. All Nix based tools can properly process the DOS EOL characters. Microsoft based tools cannot process the Nix style EOL characters.

3.2.2 MACRO Statements

Use of MACRO statements is optional.

Summary

Macro statements are characterize by a `DEFINE` line. Macro statements in INF files are only permitted to describe a path (shortcut name,) or used to provide a shorter text string in C Flags in the `[BuildOptions]` section. If the Macro statement is within the `[Defines]` section, then the Macro is common to the entire file, with local definitions taking precedence (if the same MACRO name is used in subsequent sections, then the MACRO value is local to only that section.)

Macro statements in comments must be ignored by parsing tools.

Macro statements that are referenced before they are defined will have a value of zero. A macro defined in a section that is common to all architectures is also value for sections that have architectural modifiers.

It is recommended that if the tools encounter a macroval, as in `$(MACRO)`, that is not defined, the build tools must break.

Prototype

```
<MacroDefinition> ::= <TS> "DEFINE" <MTS> <MACRO> <Eq> [<Value>] <EOL>
<Value>           ::= {<PATH>} {<CFlags>} {<Filename>}
```

Parameters

Path Definitions

Whitespace characters are not permitted in path statements. While some operating systems permit using space characters or special characters within a path element, the EDK II build system will not support whitespace characters and will only permit alphanumeric characters, and the dot, dash, underline, forward and back slash characters in file names. Use of "..", "." and "../" in the directory path is not permitted. Use of an absolute path is not permitted. It is also permitted, although not specified in the EBNF for `<PATH>` to end have the path end with the file separator character, as in MdePkg/.

Examples:

```
DEFINE TEST          = test
DEFINE TEST1         = test/
DEFINE TEST2         = MyFile.c
DEFINE TEST3         = $(TEST1)$(TEST2)
DEFINE TEST4         = $(TEST)/$(TEST2)
DEFINE GEN_SKU       = MyPlatformPkg/GenPei
DEFINE SKU1          = MyPlatformPkg/Sku1/Pei
DEFINE OPENSSL_FLAGS = -DOPENSSL_SYSNAME_UWIN -DOPENSSL_SYS_UEFI
```

Table 4 Macro Usages

MACRO DEFINITION	MACRO USAGE
`DEFINE MY_MACRO = test1`	`$(MY_MACRO)/test2/test3.inf`
`DEFINE MY_MACRO = test1/`	`$(MY_MACRO)test2/test3.inf`
`DEFINE MY_MACRO = test3.inf`	`test1/test2/$(MY_MACRO)`
`DEFINE MY_MACRO = test3`	`test1/test2/$(MY_MACRO).inf`
`DEFINE MY_MACRO = test1/test2/test3.inf`	`$(MY_MACRO)`

3.2.3 Conditional Statements

The conditional statements are not permitted anywhere within the INF file.

3.2.4 !include Statement

The `!include` statement is not permitted in an EDK II INF file.

3.2.5 Special Comment Blocks

This section defines special format comment blocks that contain information about this module. These comment blocks are not required. They may appear at the end of any section within the INF file, however it is preferred that they appear at the end of the file. The format of these comment blocks is the recommended format that will guarantee that the information is correctly inserted into UEFI Distribution Package description files. If this comment block appears in a "Source" INF file, the EDK II build tools must copy this comment block into the generated "As Built" binary INF file.

These comment blocks are only required for modules that use C calls to perform actions using UEFI defined functions listed in below.

There are three predefined types of comments.

The Event types which describe timer delays used by the Boot Services SetTimer() call.

- `EVENT_TYPE_PERIODIC_TIMER` - Event is to be signaled periodically.
- `EVENT_TYPE_RELATIVE_TIMER` - Event is to be signaled in x 100ns units.
- `UNDEFINED` - This will appear when a UEFI Distribution Package tool was unable to parse the comment (spelling error) when creating a distribution package, and the tool installed the distribution package using this value.

The BootMode types which describe the BootMode values in the Boot Services SetBootMode() and GetBootMode() calls.

- `FULL` - Equivalent to `BOOT_WITH_FULL_CONFIGURATION`
- `MINIMAL` - Equivalent to `BOOT_WITH_MINIMAL_CONFIGURATION`
- `NO_CHANGE` - Equivalent to `BOOT_ASSUMING_NO_CONFIGURATION_CHANGES`
- `DIAGNOSTICS` - Equivalent to `BOOT_WITH_FULL_CONFIGURATION`
- `DEFAULT` - Equivalent to `BOOT_WITH_FULL_CONFIGURATION`
- `S2_RESUME` - Equivalent to `BOOT_ON_S2_RESUME`
- `S3_RESUME` - Equivalent to `BOOT_ON_S3_RESUME`
- `S4_RESUME` - Equivalent to `BOOT_ON_S4_RESUME`
- `S5_RESUME` - Equivalent to `BOOT_ON_S5_RESUME`
- `FLASH_UPDATE` - Equivalent to `BOOT_ON_FLASH_UPDATE`
- `RECOVERY_FULL` - Equivalent to `BOOT_IN_RECOVERY_MODE`
- `BOOT_MFG_MODE` - Equivalent to `BOOT_WITH_MFG_MODE_SETTINGS`
- `UNDEFINED` - This will appear when a UEFI Distribution Package tool was unable to parse the comment (spelling error) when creating a distribution package and the tool installed the distribution package using this value.

The following items are defined as special boots that may use the bit field values: 100001b - 111111b per the PI PEI specification, table 6 Since this comment block is informational, no attempt is made to map these items to specific bit patterns.

- `RECOVERY_MINIMAL`
- `RECOVERY_NO_CHANGE`
- `RECOVERY_DIAGNOSTICS`
- `RECOVERY_DEFAULT`
- `RECOVERY_S2_RESUME`
- `RECOVERY_S3_RESUME`
- `RECOVERY_S4_RESUME`
- `RECOVERY_S5_RESUME`
- `RECOVERY_FLASH_UPDATE`

The Hand-Off Block types refer to the various HOBs as defined by the PI specification, HOB Code Definitions. Modules that use GetHobList() and CreateHob() should list this content.

- `PHIT` - the Phase Handoff Information Table (PHIT) Hob
- `MEMORY_ALLOCATION` - Describes all memory ranges

- `LOAD_PEIM` - This refers to EFI_HOB_TYPE_LOAD_PEIM_UNUSED
- `RESOURCE_DESCRIPTOR` - describes resource properties
- `FIRMWARE_VOLUME` , `FIRMWARE_VOLUME2` - location and type of firmware volumes
- `MEMORY_POOL` - describes memory pool allocations
- `GUID_TYPE` - for HOB types not define by the PI specification
- `UEFI_CAPSULE` - describes UEFI capsule memory pages
- `CPU` - describes processor information
- `UNUSED` - the HOB's content can be ignored
- `UNDEFINED` - This will appear when a UEFI Distribution Package tool was unable to parse the comment (spelling error) when creating a distribution package and the tool installed the distribution package using this value.

Prototype

```
<FixedCommentSection> ::= <ValidArchitectures>?
                          <EventSection>?
                          <BootModeSection>? <HobSection>*
<ValidArchitectures>  ::= "#" <EOL>
                          "#" <TS> "VALID_ARCHITECTURES" <Eq> <ArchL>
                          ["#" <EOL>]
<ArchL>               ::= <Arch> [<Space> <Arch>]* [<EbcCmt>] <EOL>
<EbcCmt>              ::= <Space> "(EBC is for build only)"
<EventSection>        ::= "#" <TS> "[Event]" <EOL> <EventBlock>*
<CommonDescription>   ::= "#" <TS> "##"<EOL>
                          ["#" <TS> "#" <TS> <Description> <EOL>]+ "#" <TS>
                          "#"<EOL>
<EventBlock>          ::= <CommonDescription>*
                          "#" <TS> <EventType> <UsageField> <EOL>
<UsageField>          ::= <TS> "##" <TS> <Usage>
<EventType>           ::= {"EVENT_TYPE_PERIODIC_TIMER"}
                          {"EVENT_TYPE_RELATIVE_TIMER"} {"UNDEFINED"}
<Usage>               ::= {"CONSUMES"} {"SOMETIMES_CONSUMES"}
                          {"PRODUCES"} {"SOMETIMES_PRODUCES"}
                          {"UNDEFINED"}
<Description>         ::= <AsciiString>
<BootModeSection>     ::= "#" <TS> "[BootMode]" <EOL> <BootModeBlock>*
<BootModeBlock>       ::= [<CommonDescription>]
                          "#" <TS> <BootModeType> <UsageField> <EOL>
<BootModeType>        ::= {"FULL"} {"BOOT_WITH_FULL_CONFIGURATION"}
                          {"MINIMAL"}
                          {"BOOT_WITH_MINIMAL_CONFIGURATION"}
                          {"NO_CHANGE"}
                          {"BOOT_ASSUMING_NO_CONFIGURATION"}
                          {"DIAGNOSTICS"}
                          {"BOOT_WITH_FULL_CONFIGURATION_PLUS_DIAGNOTICS"}
                          {"DEFAULT"} {"BOOT_WITH_DEFAULT_SETTINGS"}
                          {"S2_RESUME"} {"BOOT_ON_S2_RESUME"} {"S3_RESUME"}
                          {"BOOT_ON_S3_RESUME"}
                          {"S4_RESUME"} {"BOOT_ON_S4_RESUME"}
                          {"S5_RESUME"} {"BOOT_ON_S5_RESUME"}
                          {"FLASH_UPDATE"} {"BOOT_ON_FLASH_UPDATE"}
                          {"BOOT_MFG_MODE"} {"BOOT_WITH_MFG_MODE_SETTINGS"}
                          {"RECOVERY_FULL"} {"BOOT_IN_RECOVERY_MODE"}
                          {"RECOVERY_MINIMAL"}
                          {"RECOVERY_NO_CHANGE"}
                          {"RECOVERY_DIAGNOSTICS"}
                          {"RECOVERY_DEFAULT"} {"RECOVERY_S2_RESUME"}
                          {"RECOVERY_S3_RESUME"}
                          {"RECOVERY_S4_RESUME"}
                          {"RECOVERY_S5_RESUME"}
                          {"RECOVERY_FLASH_UPDATE"} {"UNDEFINED"}
<HobSection>          ::= "#" <TS> "[Hob" [<attribs>] "]" <EOL>
                          <HobBlock>*
<attribs>             ::= <attr> ["," <TS> "Hob" <attr>]*
<attr>                ::= "." <arch>
<HobBlock>            ::= [<CommonDescription>]
```

```
                               "#" <TS> <HobType> <UsageField> <EOL>
<HobType>                 ::= {"PHIT"} {"MEMORY_ALLOCATION"} {"LOAD_PEIM"}
                               {"RESOURCE_DESCRIPTOR"} {"FIRMWARE_VOLUME"}
                               {"FIRMWARE_VOLUME2"} {"MEMORY_POOL"}
                               {"GUID_TYPE"} {"UEFI_CAPSULE"} {"CPU"}
                               {"UNUSED"} {"UNDEFINED"}
```

Parameters

Event Usage

- `CONSUMES` - The module registers a notification function and requires that it be executed for the module to fully function.
- `SOMETIMES_CONSUMES` - A module registers a notification function and calls the function when it is signaled.
- `PRODUCES` - A module will always signal the event.
- `SOMETIMES_PRODUCES` - A module will sometimes signal the event.

Boot Mode Usage

- `CONSUMES` - The module always supports the given boot mode.
- `SOMETIMES_CONSUMES` - The module may support a given boot mode on some execution paths.
- `PRODUCES` - The module will change the boot mode.
- `SOMETIMES_PRODUCES` - The module will change the boot mode on some execution paths.

Hob Usage

- `CONSUMES` - The HOB must be present in the system.
- `SOMETIMES_CONSUMES` - If present, the HOB will be used.
- `PRODUCES` - A module will always produce the HOB.
- `SOMETIMES_PRODUCES` - The HOB may be produced by the module under some execution paths.

Usage

- Keywords are: `UNDEFINED` , `CONSUMES` , `SOMETIMES_CONSUMES` , `PRODUCES` and `SOMETIMES_PRODUCES`

3.3 Header Section

This is an optional section, while this header section is not needed by the EDK II build system, it will be used by the build tools for generating "As Built" INF files from sources.

This section is also used by the Intel(R) UEFI Packaging Tool that is included with the EDK II build system binaries.

Summary

This section contains Copyright and License notices for the INF file in comments that start the file. This section is optional using a format of:

```
## @file
# Abstract
#
# Description
#
# Copyright
#
# License
#
##
```

This information can be derived from an XML Distribution Package file or is created by a module developer creating a new module information (INF) file.

This is an optional section.

Prototype

```
<Header>             ::= <SourceHeader>
                         [<BinaryHeader>]
<SourceHeader>       ::= <Comment>*
                         "##" [<Space>] <Space> "@file"
                         [<TS> <Filename>] <EOL>
                         [<Abstract>]
                         [<Description>]
                         <Copyright>*
                         "#" <EOL>
                         <License>*
                         "##" <EOL>+
<Abstract>           ::= "#" <MTS> <AsciiString> <EOL> ["#" <EOL>]
<Description>        ::= ["#" <MTS> <AsciiString> <EOL>]+ ["#" <EOL>]
<Copyright>          ::= "#" <MTS> <CopyName> <Date> "," <CompInfo>
<CopyName>           ::= ["Portions" <MTS>] "Copyright (c)" <MTS>
<Date>               ::= <Year> [<TS> {<DateList>} {<DateRange>}]
<Year>               ::= "2" (0-9)(0-9)(0-9)
<DateList>           ::= <CommaSpace> <Year> [<CommaSpace> <Year>]*
<DateRange>          ::= "-" <TS> <Year>
<CompInfo>           ::= (0x20 - 0x7e)* <MTS> "All rights reserved."
                         [<TS> "<BR>"] <EOL>
<License>            ::= ["#" <MTS> <AsciiString> <EOL>]+
                         ["#" <EOL>]
<BinaryHeader>       ::= <Comment>*
                         "##" [<Space>] <Space> "@BinaryHeader" <EOL>
                         <BinaryAbstract>
                         "#" <EOL>
                         <BinaryDescription>
                         "#" <EOL>
                         <Copyright>+
                         "#" <EOL>
                         <BinaryLicense>+
                        "##" <EOL>+
<Filename>          ::= <Word> "." <Extension>
<BinaryAbstract>    ::= "#" <MTS> <AsciiString> <EOL>
<BinaryDescription> ::= ["#" <MTS> <AsciiString> <EOL>]+
<BinaryLicense>     ::= ["#" <MTS> <AsciiString> <EOL>]+
                        ["#" <EOL>]
```

Parameters

Abstract

A brief one line description of what the module does.

- The INF file will always have an English version of the Abstract. Other localized versions of the abstract will be stored in the Unicode file specified in the `[Defines]` section's `MODULE_UNI_FILE`.

BinaryAbstract

A brief one line description of what the module does that may be different from a source abstract.

- If this line is present, the tools will use this line when generating the binary "As Built" INF.
- If the Doxygen tag is not present, the tools will use the primary Abstract from the Source INF file when generating a binary "As Built" INF.
- In the binary "As Built" INF, the Doxygen tag must not be present.
- The INF file will always have an English version of the Abstract. Other localized versions of the abstract will be stored in the Unicode file specified in the `[Defines]` section's `MODULE_UNI_FILE`.

Note: This file permits the Intel(R) UEFI Packaging Tool to process localized module content described in the UEFI Platform Initialization Distribution Package Specification.

Description

A detailed description of what the module does.

- The INF file will always have an English version of the Description.. Other localized versions of the description will be stored in the Unicode file specified in the `[Defines]` section's `MODULE_UNI_FILE`.

BinaryDescription

A detailed description of what the module does that may be different from a source abstract.

- If this line is present, the tools will use this line when generating the binary "As Built" INF.
- If the Doxygen tag is not present, the tools will use the primary Description from the Source INF file when generating a binary "As Built" INF.
- In the binary "As Built" INF, the Doxygen tag must not be present.
- The INF file will always have an English version of the Description.. Other localized versions of the description will be stored in the Unicode file specified in the `[Defines]` section's `MODULE_UNI_FILE`.

Note: This file permits the Intel(R) UEFI Packaging Tool to process localized module content described in the UEFI Platform Initialization Distribution Package Specification.

Copyright

The copyright date should be modified if there is a functional change to the source code. Since binaries are constructed from source, the binary file uses the same copyright date as the source INF. Copyright data will not be localized.

License

One or more licenses that the module with source code is released under. License content will not be localized.

BinaryLicense

One or more licenses that the binary module is released under that may be different from the licenses used for distributing the module with source code. License content will not be localized.

- If this tag is present, the tools will use this content when generating the binary "As Built" INF.
- If the Doxygen tag is not present in the source INF, the tools will use the License content from the Source INF file when generating a binary "As Built" INF.
- In the binary "As Built" INF, the Doxygen tag must not be present.

Example

```
## @file
# EFI/Framework Base Memory Library
#
# Implementation of a base memory library that provides minimum
# functionality for accessing memory.
#
# Copyright (c) 2006 - 2008, Intel Corporation. All rights reserved.
#
# This program and the accompanying materials are licensed and made
# available under the terms and conditions of the BSD License which
# accompanies this distribution. The full text of the license may be
# found at:
# http://opensource.org/licenses/bsd-license.php
#
# THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
# WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS
# OR IMPLIED.
#
##
```

3.4 [Defines] Section

This is a required section.

Summary

This describes the required `[Defines]` section used in EDK II INF files. This file is created during installation of a UEFI distribution package or by the developer and is an input to the new build tool parsing utilities. Elements may appear in any order within this section.

The version for this specification is "0x0001001B" and new versions of this specification must increment the minor (001B) portion of the specification code for backward compatible changes, and increment the major number for non-backward compatible specification changes. This value may also be specified as a decimal value, 1.27.

The `[Defines]` section assigns values to the symbols that describe the component. Some items are emitted to the output makefile, others are used to create filenames during the build. Some symbols are emitted to the generated C files.

The `FILE_GUID` is required for all EDK II modules. This GUID is used to build the FW volume file list used by build tools to generate the final firmware volume, as well as processed in some SMM, PEI or DXE `DEPEX` statements.

All new EDK II INF files must include one of the following statements: `INF_VERSION = 0x0001001B` or `INF_VERSION = 1.27` in this section, where the number varies according to the release of this specification. It is a HexVersion type, where the 0x0001 is the major number, and the 001B is the minor number. This version of the specification provides full backward compatibility to all previous versions. This means that tools that process this version of the specification can also process earlier versions of EDK II INF files.

This version of the specification removes content in this section that was associated with EDK libraries and components. The section now lists only the defined EDK II symbols and format.

Note: Possible values for `MODULE_TYPE`, and their descriptions, are listed in the table, "EDK II Module Types." For each module, the `BASE_NAME` and `MODULE_TYPE` are required. The `BASE_NAME` definition is case sensitive as it will be used to create a directory name during a build.

Unlike EDK, only the `[Defines]` section tag is valid for EDK II INF files - architectural modifiers for the `[Defines]` section tag are not permitted. The section is processed in order by the parsing utilities. Assignments of variables in other sections override previous assignments.

The `SHADOW` keyword is only valid for `SEC`, `PEI_CORE` and `PEIM` module types. It is an error to declare the `SHADOW` keyword in other module types. The default value of `SHADOW` is "FALSE" when the `SHADOW` keyword is not specified.

EDK II modules that provide different library class implementations must use multiple `LIBRARY_CLASS` statements. Each `LIBRARY_CLASS` statement must provide the name of the library class supported, followed by the pipe "|" field separator and then a space " " delimited list of module types the library instances supports. The following is an example of specifying multiple library classes.

```
LIBRARY_CLASS = FOO | PEI_CORE PEIM
LIBRARY_CLASS = BAR | DXE_CORE DXE_DRIVER DXE_SMM_DRIVER
```

Note: Each library class requires a header file defined in the package that declares the library class. Refer to the "EDK II Module Writer's Guide" for more information about writing drivers and libraries.

Additionally, a driver module may expose internal implementations of a library class, making the internal implementations public. As an example, a `DXE_CORE` implementation that uses internal functions that provide the functionality of the EDK II Base Memory Library. The `DXE_CORE` module that provides these functions (for example, `DxeMain.inf`) can expose them for use by other DXE drivers that depend on the BaseMemoryLib library class.

The OptionRom statements must be included for UEFI PCI Option ROMs, and can only be used with a `MODULE_TYPE` of `UEFI_DRIVER`.

The optional `MODULE_UNI_FILE` entry is used to locate an Unicode file which can be used for localization of the module's Abstract and Description from the header section. The content of the file can be generated by tools during the installation of a distribution package that conforms to the UEFI Platform Initialization Distribution Packaging Specification, or by module developers creating new content.

The `FixedComments` sections that follow a defines section are to permit tools to work with UEFI Distribution Packaging Specification requirements.

Prototype

Note: The entry, `INF_VERSION`, `BASE_NAME`, `FILE_GUID` and `MODULE_TYPE` are required for EDK II Modules. The `VERSION_STRING` entry is highly recommended.

```
<Defines>          ::= "[Defines]" <EOL>
                       <DefineStatements>+
<DefineStatements> ::= <TS> "INF_VERSION" <Eq> <SpecVersion> <EOL>
                       <TS> "BASE_NAME" <Eq> <BaseName> <EOL>
                       <TS> "FILE_GUID" <Eq> <RegistryFormatGUID> <EOL>
                       <TS> "MODULE_TYPE" <Eq> <Edk2ModuleType> <EOL>
                       [<TS> "UEFI_SPECIFICATION_VERSION" <Eq>
                       <VersionVal> <EOL>]
                       [<TS> "PI_SPECIFICATION_VERSION" <Eq>
                       <VersionVal> <EOL>]
                       [<TS> "LIBRARY_CLASS" <Eq> <LibClass> <EOL>]*
                       [<TS> "BUILD_NUMBER" <Eq> <NumValUint16> <EOL>]
                       [<TS> "VERSION_STRING" <Eq>
                       <DecimalVersion> <EOL>]
                       [<TS> "PCD_IS_DRIVER" <Eq>
                       <PcdDriverType> <EOL>]
                       [<TS> "ENTRY_POINT" <Eq> <CName> [<FFE>] <EOL>]*
                       [<TS> "UNLOAD_IMAGE" <Eq> <CName>
                       [<FFE>] <EOL>]*
                       [<TS> "CONSTRUCTOR" <Eq> <CName> [<FFE>] <EOL>]*
                       [<TS> "DESTRUCTOR" <Eq> <CName> [<FFE>] <EOL>]*
                       [<TS> "SHADOW" <Eq> <BoolType> <EOL>]
                       [<OptionRomInfo>]
                       [<TS> "CUSTOM_MAKEFILE" <Eq>
                       <CustomMake> <EOL>]*
                       [<TS> "DPX_SOURCE" <Eq> <Filename> <EOL>]
                       [<TS> "SPEC" <MTS> <Spec> <EOL>]*
                       [<TS> <UefiHiiResource> <EOL>]
                       [<TS> "MODULE_UNI_FILE" <Eq> <Filename> <EOL>]
                       [<MacroDefinition>]*
<BaseName>         ::= (a-zA-Z0-9)(a-zA-Z0-9_-.)*
<UefiHiiResource>  ::= "UEFI_HII_RESOURCE_SECTION" <Eq> <TrueFalse>
<CustomMake>       ::= [<Family> <FS>] <Filename>
<PcdDriverType>    ::= {"PEI_PCD_DRIVER"} {"DXE_PCD_DRIVER"}
```

```
<Spec>            ::= <Identifier> <Eq> <DecimalVersion>
<LibClass>        ::= {<KeywordType>} {"NULL"}
<KeywordType>     ::= <SimpleWord> [<FS> <ModuleTypes>]
<ModuleTypes>     ::= <ModuleType> [<Space> <ModuleType>]*
<FFE>             ::= <FS> <Expression>
<SpecVersion>     ::= {<HexVersion>} {(0-9))+ "." (0-9)+}
<OptionRomInfo>   ::= <TS> "PCI_VENDOR_ID" <Eq> <UINT16> <EOL>
                      <TS> "PCI_DEVICE_ID" <Eq> <UINT16> <EOL>
                      <TS> "PCI_CLASS_CODE" <Eq> <UINT8> <EOL>
                      <TS> "PCI_REVISION" <Eq> <UINT8> <EOL>
                      [<TS> "PCI_COMPRESS" <Eq> <TruFal> <EOL>]
<TruFal>          ::= {"TRUE"} {"FALSE"}
```

Parameters

Filename

Filenames listed in the `[Defines]` section must be relative to the directory the INF file is in. Use of "..", "." and "../" in the directory path is not permitted. Use of an absolute path is not permitted. The file name specified in the `MODULE_UNI_FILE` entry must be a Unicode file with an extension of .uni, .UNI or .Uni.

MODULE_TYPE

Drivers and applications are not allowed to have a `MODULE_TYPE` of `"BASE`". Only libraries are permitted to a have a `MODULE_TYPE` of `"BASE`". A INF file can be used to specify other binary files types, such as logo images or legacy16 option ROMs. The `USER_DEFINED` module type must be used in all cases where the module type is not a member of `<Edk2ModuleType>`.

INF_VERSION

For new INF files, the version value must be set to `0x0001001B`. Tools that process this version of the INF file can successfully process earlier versions of the INF file (this is a backward compatible update). There is no requirement to change the value in existing INF files if no other content changes. This may also be specified as decimal value, 1.27.

EDK_RELEASE_VERSION

This optional value may be set to the major/minor number of the EDK II release required for modules to function correctly.

UEFI_SPECIFICATION_VERSION

The `UEFI_SPECIFICATION_VERSION` must only be set in the INF file if the module depends on UEFI Boot Services or UEFI Runtime Services or UEFI System Table fields or UEFI core behaviors that are not present in the UEFI 2.1 version. The version number for the UEFI 2.3.1 specification is the hex value: 0x0002001F. The minor number of the specification version is a 2 digit number, where the 2.3.1 is actually: 2.31 The major number must be incremented on a revision that would result in a minor number greater than 99.

PI_SPECIFICATION_VERSION

The `PI_SPECIFICATION_VERSION` must only be set in the INF file if the module depends on services or system table fields or PI core behaviors that are not present in the PI 1.0 version. The version number for the 1.2 PI specification is the hex value: 0x00010014 The minor number of the specification version is a 2 digit number, where the 1.2 is actually: 1.20 The major number must be incremented on a revision that would result in a minor number greater than 99.

VERSION_STRING

This is typically a decimal number, and if not specified, an empty string will be provided. This value will be converted by the build tools from an ASCII string into a null-terminated Unicode string that contains a text representation of the version. A platform integrator may specify a different version string for an FFS version section in the FDF file if more than just this value is needed. It is recommended that the decimal number be used in such a manner as the integer portion of the value is considered the major number (changing when there is a functional, non-backward compatible change). The factional portion of the value is the considered the minor number (changing anytime there is a change in the code that would result in a binary image that was not identical to the binary image created prior to the change).

BuildNumber

The optional build number must be NumValUint16 If not present, the EDK II build tools will use the `BUILD_NUMBER` from the DSC file. If the DSC file does not include the build number, the EDK II build tools will use a value of 0 If the Build number is greater than `0`, the generated INF file must contain this entry.

Spec

The user is required to ensure that a valid C name is used for the name of the specification, and provide the decimal version of the specification. For example, `SPEC USB_SPECIFICATION_VERSION = 2.0` is a valid statement. These statements are used to generate #define statements in the auto generated C files.

UefiHiiResource

This is an optional tag used to identify UEFI compliant drivers that must have a `UEFI_HII_RESOURCE_SECTION` generated as part of the efi image file. If not specified, the default is false.

DPX_SOURCE

The path and filename must be relative to the INF file and located within the module's directory tree. The file must contain only DEPEX statements as defined in the UEFI PI Specification that are valid for the module type. C style Comments are not in the file. Contents of this file completely override any dependency expressions listed in `[Depex]` sections and all inherited dependency expressions that would normally have been inherited from libraries linked to the module. Use of this feature is not recommended for normal use.

OptionRomInfo

These statements are used by developers of stand-alone PCI Option ROM drivers.

They allow the developer to forego creation of an FDF file in the package directory. Only the INF file and a DSC file are required for pure PCI Option ROM development.

ENTRY_POINT CName

This is the name of the driver's entry point function. Refer to the UEFI Driver Writer's Guide for more information.

UNLOAD_IMAGE CName

If a driver chooses to be unloadable, then this is the name of the module's function registered in the Loaded Image Protocol. It is called if the UEFI Boot Service UnloadImage() is called for the module, which then executes the Unload function, disconnecting itself from handles in the database as well as uninstalling any protocols that were installed in the driver entry point. The CName is the name of this module's unload function. Refer to the UEFI Driver Writer's Guide for more information.

Example (EDK II Driver)

```
[Defines]
  INF_VERSION                = 1.27
  BASE_NAME                  = PlatformAcpiTable
  FILE_GUID                  = 7E374E25-8E01-4FEE-87F2-390C23C606CD
  MODULE_TYPE                = DXE_DRIVER
  VERSION_STRING             = 1.0
  EDK_RELEASE_VERSION        = 0x00020000
  UEFI_SPECIFICATION_VERSION = 0x00020014
```

Example (UEFI Driver)

```
[Defines]
  INF_VERSION    = 0x0001001B
  BASE_NAME      = Abc
  FILE_GUID      = DA87D340-15C0-4824-9BF3-D52286674BEF
  MODULE_TYPE    = UEFI_DRIVER
  VERSION_STRING = 1.0
  ENTRY_POINT    = AbcDriverEntryPoint
  UNLOAD_IMAGE   = AbcUnload
```

Example (EDK II Library)

```
[Defines]
  INF_VERSION    = 1.27
  BASE_NAME      = LzmaCustomDecompressLib
  FILE_GUID      = 22f8406f-43ee-492f-82f5-4e8a1a58e6d2
  MODULE_TYPE    = BASE
  VERSION_STRING = 1.0
  LIBRARY_CLASS  = CustomDecompressLib
```

3.5 [BuildOptions] Sections

These sections are optional for EDK II INF files.

Summary

Defines the `[BuildOptions]` section content. These sections are used to define the custom definitions for individual tools. There are two styles for options, a replacement of any previous definition (for this module only) of the flags or commands used (specified by the double "==" equal sign) to process the module code, or an append option, (specified by the single "=" equal sign) which will be appended to the previous definition. When the single "=" equal sign is used, the string to the left, is appended, typically used to override a single flag. When the double "==" sign is used, then any previous definition (reference build requires that it must be defined `tools_def.txt` file) will be cleared, and the left value replaces the entire previous definition.

The double "==" equal sign must be used to replace a command (specified by the PATH attribute,) appending is not an option.

For the DPATH attribute, can use either the replace or append, and is used to expand the system environment `PATH` variable prior to processing any commands.

Other, user defined attributes, can be specified, in this section, using either the append or replace. Usage of these overrides is implementation specific.

Macro use is permitted in the `[BuildOptions]` sections, and must follow the rule that Macros used in this section must be defined locally within the INF file; use of externally defined MACROs is prohibited. Additionally, a `$(MACRO)` that appears inside of a quoted string in R-Values (following an append "=" or replace "==") is permitted, as parsing tools are not required to expand those macro values. Macros within quoted strings do not need to be defined locally.

Macro statements, `$(MACRO)`, that are encapsulated in double quotation marks are not expanded, nor are they processed by parsing tools, as double quotation marks indicate a string that must be treated as a single entity. Macro statements in comments must also be ignored by parsing tools.

Macros are not allowed on the left side of the assignment statement (left of the equal sign).

The EDK II build system will provide an option to create an "As Built" INF file that can be used for binary distributions. This section will be completed, listing all of the option flags for every application that was used to create the binary. Since these "As Built" flags are within comment sections, the actual flag string can be extended to a new comment line without using the line extension character.

Tools that create "As Built" information must expand any macro values used by the tools during the module build. The standard Macro Definitions are not permitted within this section for an "As Built" INF file.

Build options listed in architectural sections will be appended to build options listed in the common architectural section.

Comments in this section must appear on a separate line, they may not be appended after statements.

The "common" architecture modifier in a section tag must not be combined with other architecture type; doing so will result in a build break.

Prototype

```
<BuildOptions>  ::= "[BuildOptions" [<com_attrs>] "]" <EOL> <BuildStmts>
<com_attrs>     ::= {".common"} {<attrs>}
```

```
<attrs>         ::= <Archs> ["," <TS> "BuildOptions" <attrs>]*
<Archs>         ::= "." <arch>
<BuildStmts>    ::= {<BuildOptStmts>*} {<AsBuiltStmts>}
<AsBuiltStmts>  ::= <TS> "##" <TS> "@AsBuilt" <EOL> <AsBuiltFfe>*
<AsBuiltFfe>    ::= <TS> "##" <FamId> <ToolFlags> <Eq> <CFlags> <EOL> [<TS> "#"
                    <CFlagsContd> <EOL>]*
<CFlagsContd>   ::= <CFlags>
<BuildOptStmts> ::= {<FlagExpr>} {<PathExpr>} {<CmdExpr>} {<Other>}
<FamId>         ::= <Family> ":"
<FlagExpr>      ::= <TS> [<FamId>] <ToolFlags> <Equal> <CFlags> <EOL>
<PathExpr>      ::= <TS> [<FamId>] <ToolPath> <Equal> <PATH> <EOL>
<CmdExpr>       ::= <TS> [<FamId>] <ToolCmd> <ReplaceEq> <PathCmd> <EOL>
<Other>         ::= {<OtherTool>} {<MacroDefinition>}
<OtherTool>     ::= <TS> [<FamId>] <ToolOther> <Equal> <String> <EOL>
<Family>        ::= {"MSFT"} {"GCC"} {"INTEL"} {<Usr>} {<Wildcard>}
<Usr>           ::= <ToolWord>
<Equal>         ::= {<AppendEq>} {<ReplaceEq>}
<AppendEq>      ::= <Eq>
<ReplaceEq>     ::= <TS> "==" <TS>
<ToolSpec>      ::= <Target> "_" <TagName> "_" <tarch> "_" <CmdCode>
<ToolFlags>     ::= <ToolSpec> "_FLAGS"
<ToolPath>      ::= <ToolSpec> "_DPATH"
<ToolCmd>       ::= <ToolSpec> "_PATH"
<ToolOther>     ::= <ToolSpec> "_" <Attribute>
<Target>        ::= {<Wildcard>} {Target}
<TagName>       ::= {<Wildcard>} {TagName}
<CmdCode>       ::= CommandCode
<CommandName>   ::= CommandExecutable
<Attribute>     ::= Attribute
<tarch>         ::= {"IA32"} {"X64"} {"IPF"} {"EBC"} {<OA>}
                    {<Wildcard>}
<CFlags>        ::= (0x20 - 0x7e)+
<PathCmd>       ::= <TOOLPATH> <FileSep> <CommandName>
<TOOLPATH>      ::= {<PATH>} {<ABS_PATH> <PATH>}
<ABS_PATH>      ::= {(a-zA-Z) ":\"} {"\"} {"/"}
```

Parameters

All of the keywords that make up the left side of the expression must be alphanumeric only - no special characters are permitted. For more information about the following parameters, refer to the Build Specification for as description of the `tools_def.txt` file. In order for the entries in the INF file to be valid, there must be a matching definition in the `tools_def.txt` file. The tool chain tag name must also match the one used for the build.

Family

Must match a `FAMILY` name defined in the EDK II `tools_def.txt` file. If not present, then the entry is valid for all tool chain families.

TOOLPATH

Paths listed in the `[BuildOptions]` section are relative to the system environment variable, `EDK_TOOLS_PATH`, or they may be absolute. Use of "..", "." and "../" in the directory path is not permitted. If the absolute format is used, the module cannot be distributed using a UEFI Distribution Package.

Target

A keyword that uniquely identifies the build target; the first field, where fields are separated by the underscore character. Three values, "NOOPT", `"DEBUG`" and `"RELEASE`" have been pre-defined. This keyword is used to bind command flags to individual commands.

Users may want to add other definitions, such as, PERF, SIZE or SPEED, and define their own set of FLAGS to use with these tags. The wildcard character, "``*", is permitted after it has been defined one time for a tool chain.

TagName

A keyword that uniquely identifies a tool chain group; the second field. Wildcard characters are permitted if and only if a command is common to all tools that will be used by a developer. As an example, if the development team only uses IA32 Windows workstations, the ACPI compiler can be specified as: `DEBUG_*_*_ASL_PATH` and `RELEASE_*_*_ASL_PATH`.

Arch

A keyword that uniquely identifies the tool chain target architecture; the third field. This flag is used to support the cross-compiler features, such as when a development platform is IA32 and the target platform is X64 Using this field, a single `TagName` can be setup to support building multiple target platform architectures with different tool chains. As an example, if a developer is using Visual Studio .NET 2003 for generating IA32 platform and uses the WINDDK version 3790.1830 for X64 or IPF platform images, a single tag (see the MYTOOLS PATH settings in the generated `Conf/tools_def.txt` or provided `BaseTools/Conf/tools_def.template` file.) The wildcard character, "``*", is permitted if and only if the same tool is used for all target architectures.

CommandExecutable

The full executable name, such as `cl.exe` or `gcc`, with the preceding path specifying the exact location of the command. If the executable can be located under a directory specified in the system environment `PATH` variable, only the filename is required. Otherwise, a relative path to a path listed in the system environment PATH variable or an absolute path must be given. If an absolute path is used, the build system will fail the build if the executable cannot be found.

CommandCode

A keyword that uniquely identifies a specific command; the fourth field. Several `CommandCode` keywords have been predefined. See table below for the pre-defined keywords and functional mappings. The wildcard character, "", is permitted only for the *`FAMILY`*, *`DLL`* and *`DPATH`* attributes (see *Attributes below.)

Table 5 Predefined Command Codes

CommandCode	Function
`APP`	C compiler for applications.
`ASL`	ACPI Compiler for generating ACPI tables.
`ASLCC`	ACPI Table C compiler
`ASLDLINK`	ACPI Table C Dynamic linker
`ASLPP`	ASL C pre-processor
`ASM`	A Macro Assembler for assembly code in some libraries.
`ASMLINK`	The Linker to use for assembly code generated by the ASM tool.
`CC`	C compiler for PE32/PE32+/Coff images.
`DLINK`	The C dynamic linker.
`MAKE`	Required for tool chains. This identifies the utility used to process the Makefiles generated by the first phase of the build.
`PCH`	The compiler for generating pre-compiled headers.
`PP`	The C pre-processor command.
`SLINK`	The C static linker.
`TIANO`	This special keyword identifies a compression tool used to generate compression sections as well as the library needed to uncompress an image in the firmware volume.
`VFR`	The VFR file compiler which creates IFR code.

VFRPP	The C pre-processor used to process VFR files.

Attribute

A keyword to uniquely identify a property of the command; the fifth and last field. Several pre-defined attributes have been defined: `DLL`, `FAMILY`, `FLAGS`, `GUID`, `OUTPUT` and `PATH`. Use quotation marks if and only if the quotation marks must be included in the flag string. The following example shows the format for the required quoted string, `"C:\Program Files\Intel\EBC\Lib\EbcLib.lib"`. Normally, the quotation characters are not required as everything following the equal sign to the end of the line is used for the flag.

Flags

Must be a valid string for the tool specified. The string will be appended to the end of the tool's flags (from the `tools_def.txt`.) Both Microsoft and GCC evaluate options from left to right on the command line. This allows disabling some flags that may have been specified in the `tools_def.txt` by providing an alternate flag, i.e., if the tools_def: CC FLAGS defines /O2 and an /O1 options is specified for this module, the module will compile with /O1 (size) not with /O2 (speed.)

Space characters are allowed. Macros are also permitted to be used in Flag strings.

Example

```
[BuildOptions.common]
  DEFINE MACRO                   = /nologo
  *_WINDDK3790x1830_*_CC_FLAGS  = /Qwd1418,810
  *_MYTOOLS_*_CC_FLAGS          = /Qwd1418,810
  *_VS2003_*_CC_FLAGS           = /wd4244
  *_WINDDK3790x1830_*_CC_FLAGS  = /wd4244
  *_MYTOOLS_*_CC_FLAGS          = /wd4244
  RELEASE_MYTOOLS_IPF_ASM_FLAGS = = -N us -X explicit -M ilp64 - N so -W3 MSFT:*_*_*_*_FLAGS = /od $(MACRO)
```

3.6 [LibraryClasses] Sections

These are optional sections.

Summary

Defines the EDK II `[LibraryClasses]` section content. The Library Class entries are single lines with one or two fields, separated by the pipe "|" character.

The EDK II build system will provide an option to generate an "As Built" INF that can be used to distribution binary modules. Since a binary distribution does not build, the library instances that were linked into the binary are listed in comments, rather than as library class keywords and recommended instances. Tools that create "As Built" information must expand any macro values used by the tools during the module build. Listing a library class keyword outside of the "As Built" information is prohibited.

Each library class keyword must only be listed once in a library classes section. Library class keywords listed in architectural sections are not permitted to be listed in the common architectural section.

The `"common"` architecture modifier in a section tag must not be combined with other architecture type; doing so will result in a build break.

The use of any form of the word `"NULL`" (as in "Null" or "null") as a keyword in the entries is prohibited.

Prototype

```
<LibraryClasses>     ::= "[LibraryClasses" [<com_attrs>] "]" <EOL>
                         [<Statements>]
<com_attrs>          ::= {".common"} {<attrs>}
<attrs>              ::= <Archs> ["," <TS> "LibraryClasses" <attrs>]*
<Archs>              ::= "." <arch>
<Statements>         ::= {<SourceContent>*} {<AsBuiltInfo>}
<SourceContent>      ::= <TS> {<SourceStmts>} {<MacroDefinition>}
<SourceStmts>        ::= [<RecInstanceCmt> <Filename> <EOL>] <TS> <Keyword>
                         [<Field2>] <EOL>
<RecInstanceCmt>     ::= "##" <MTS> "@RecommendedInstance" <MTS>
<NoN>                ::= (A-MO-Z)
<NoU>                ::= (a-tv-zA-TV-Z0-9)
<NoL>                ::= (a-km-zA-TV-Z0-9)
<Keyword>            ::= {(A-Z) (a-zA-Z0-9){0,2}}
                         {(A-Z) (a-zA-Z0-9){4,}}
                         {<NoN> (a-zA-Z0-9){1,}}
                         {(A-Z) <NoU> (a-zA-Z0-9){0,}}
                         {(A-Z) (a-zA-Z0-9) <NoL> (a-zA-Z0-9){0,}}
                         {(A-Z) (a-zA-Z0-9){2} <NoL>(a-zA-Z0-9){0,}}
<Field2>             ::= <FS> <FeatureFlagExpress>
<FeatureFlagExpress> ::= <Boolean>
<AsBuiltInfo>        ::= <TS> "##" <MTS> "@LIB_INSTANCES" <WS>
                         <LibInstance>*
<LibInstance>        ::= <TS> "#" <TS> <InfFile> <EOL>
```

Parameters

Field1

This is a keyword that uniquely identifies a library class required to successfully execute the driver.

Filename

Filenames listed in the Recommended Instance comment in the `[LibraryClasses]` section must be a package relative path. Use of "..", "." and "../" in the directory path is not permitted. If the Recommended Instance INF file is a member a Package (the Package contains a DEC file) the Package (DEC file) of must also be present in the `[Packages]` section. Use of an absolute path is prohibited.

FeatureFlagExpress

When present, the feature flag expression determines whether the entry line is valid. If the feature flag expression evaluates to FALSE, this entry will be ignored by the EDK II build tools.

Example

```
[LibraryClasses.common]
  DEFINE MDE = MdePkg/Library
  ## @RecommendedInstance $(MDE)/BaseDebugLibNull/BaseDebugLibNull.inf
  DebugLib
  UefiDriverModelLib
  PcdComponentNameDisable
  ## @RecommendedInstance $(MDE)/UefiDriverModelLib/UefiDriverModelLib.inf
  PcdDriverDiagnosticsDisable
  UefiDriverEntryPoint
  UefiLib
  ## @RecommendedInstance $(MDE)/BaseLib/BaseLib.inf
  BaseLib
```

3.7 [Packages] Sections

These are optional sections. If there are files listed under a `[Sources]` section, then the INF file is required to list the MdePkg/MdePkg.dec file as the first file in a `[Packages]` section. This section is also required if the module uses PCDs for both source and the binary "As Built" INF modules.

Summary

Defines the `[Packages]` section tag that is used in EDK II module INF files. Each entry in this section contains a directory name, forward slash character and the name of the DEC file contained in the directory name.

Packages must be listed in the order that may be required for specifying include path statements for a compiler. For example, the MdePkg/MdePkg.dec file must be listed before the `MdeModulePkg/MdeModulePkg.dec` file. If there are PCDs listed in the generated "As Built" INF, the packages that declare any PCDs must be listed in this section.

Each package filename must be listed only once per section. Package filenames listed in architectural sections are not permitted to be listed in the common architectural section.

The `"common"` architecture modifier in a section tag must not be combined with other architecture type; doing so will result in a build break.

Packages listed under the `"common"` architecture section must not be listed in sections that have other architecture modifiers.

Prototype

```
<Packages>          ::= "[Packages" [<com_attrs>] "]" <EOL> <Statements>*
<com_attrs>         ::= {".common"} {<attrs>}
<attrs>             ::= <Archs> ["," <TS> "Packages" <Archs>]
<Archs>             ::= "." <arch>
<Statements>        ::= {<MacroDefinition>} {<PkgStatements>}
<PkgStatements>     ::= <TS> <Filename> [<Field2>] <EOL>
<Field2>            ::= <FS> <FeatureFlagExpress>
<FeatureFlagExpress> ::= <Boolean>
```

Parameters

Filename

Paths listed in the `[Packages]` section contain a directory name, forward slash character and the name of the DEC file contained in the directory name. Use of "..", "." and "../" in the directory path is not permitted. Use of an absolute path is prohibited.

FeatureFlagExpress

When present, the feature flag expression determines whether the entry line is valid. If the feature flag expression evaluates to FALSE, this entry will be ignored by the EDK II build tools.

Example

```
[Packages]
  MdePkg/MdePkg.dec
  MdeModulePkg/MdeModulePkg.dec

[Packages.IA32]
```

```
  DEFINE CPUS = IA32FamilyCpuPkg
  $(CPUS)/DualCore/DualCore.dec
```

3.8 PCD Sections

The PCD sections are optional. If the source module code contains any Patchable in Module or DynamicEx PCDs, then this section must be generated in the "As Built" INF file listing each PCD with its `<CommentBlock>` content if available. Refer to the EDK II Build Specification, section 8.4.1 for PCD processing rules.

Summary

Defines the `[(PcdType)]` section content for EDK II module INF files. If a default value is specified on the entry line, it must match the datum type specified in the DEC file that declares this PCD. The Datum Type values are defined in the PI Specification. PCD expressions are single lines with two or three fields; fields are separated using the pipe "|" character. Empty Fields are permitted.

The EDK II Build system will generate an "As Built" INF file that can be delivered with a binary distribution. Only `[PatchPcd]` and [`PcdEx]` section types are valid for in "As Built" INF file. Tools that create "As Built" information must expand any macro values used by the tools during the module build.

The format of the `<CommentBlock>` is the recommended format that will guarantee that the information is correctly inserted into UEFI Distribution Package description files by the Intel(R) UEFI Packaging Tool included in the EDK II base tools project.

Each PCD name must only be listed once in a section. PCD names listed in architectural sections must not be listed in the common architectural section. PCDs should be either architectural in nature or common to all architectures. The module developer should note that all recommended values may be overridden by values specified by a platform integrator in a platform description (DSC) file.

The "common" architecture modifier in a section tag must not be combined with other architecture type; doing so will result in a build break.

It is not permissible to list a PCD name in different PCD type sections. Listing a PCD indicates what library function is used to access a PCD; only one type of access is permitted for a specified PCD.

A generated "As Built" INF file must not contain any `FeatureFlagExpression` content.

Prototype

```
<Pcds>                 ::= {<AsBuiltPcdSec>} {<SrcPcdSec>}
                           {<FeaturePcd>}
<SrcPcdSec>            ::= {<Patch>} {<Fixed>} {<Dyn>} {<DynEx>}
                           <PcdEntriesStmts>*
<PcdEntriesStmts>      ::= <PcdEntries>
<Patch>                ::= "[PatchPcd" [<com_patchAttrs> "]" <EOL>
<com_patchAttrs>       ::= {".common"} {<patchAttrs>}
<patchAttrs>           ::= <attrs> ["," <TS> "PatchPcd" <attrs>]*
<Fixed>                ::= "[FixedPcd" [<com_fixedAttrs> "]" <EOL>
<com_fixedAttrs>       ::= {".common"} {<fixedAttrs>}
<fixedAttrs>           ::= <attrs> ["," <TS> "FixedPcd" <attrs>]*
<Dyn>                  ::= "[Pcd" [<com_pcdAttrs> "]" <EOL>
<com_pcdAttrs>         ::= {".common"} {<pcdAttrs>}
<pcdAttrs>             ::= <attrs> ["," <TS> "Pcd" <attrs>]*
<DynEx>                ::= "[PcdEx" [<com_pcdexAttrs> "]" <EOL>
<com_pcdexAttrs>       ::= {".common"} {<pcdexAttrs>}
<pcdexAttrs>           ::= <attrs> ["," <TS> "PcdEx" <attrs>]*
<FeaturePcd>           ::= "[FeaturePcd" [<com_FFArchAttrs>] "]" <EOL>
                           <FeatureEntriesStmts>*
<FeatureEntriesStmts> ::= <FeatureEntries>
<com_FFArchAttrs>      ::= {".common"} {<FFArchAttrs>}
<FFArchAttrs>          ::= <attrs> ["," <TS> "FeaturePcd" <attrs>]*
<FeatureEntries>       ::= [<NUsageBlock>]
```

```
                           <TS> <PcdName> [<FfField1>] <TailCmt>
<TailCmt>              ::= {<1UsageBlock>} {<EOL>}
<FfField1>             ::= <FS> [<Boolean>] [<FfField2>]
<FfField2>             ::= <FS> [<FFE>]
<AsBuiltPcdSec>        ::= {<BuiltPatchPcd>} {<BuiltPcdEx>}
<BuiltPatchPcd>        ::= "[PatchPcd" [<com_PPArchAttrs>] "]" <EOL>
                           <ValueOffsetPcd>*
<com_PPArchAttrs>      ::= {".common"} {<PPArchAttrs>}
<PPArchAttrs>          ::= <attrs> ["," <TS> "PatchPcd" <attrs>]*
<BuiltPcdEx>           ::= "[PcdEx" [<com_PEArchAttrs>] "]" <EOL>
                           <AbPcdEx>*
<com_PEArchAttrs>      ::= {".common"} {<PEArchAttrs>}
<attrs>                ::= "." <arch>
<PEArchAttrs>          ::= <attrs> ["," <TS> "PcdEx" <attrs>]*
<AbPcdEx>              ::= [<NUsageBlockAb>]
                           <TS> <PcdName> [<TailCmt>] <EOL>
<NUsageBlockAb>        ::= {<HiiComment>} {<NUsageBlock>}
<HiiComment>           ::= <TS> ["##" <Usage> [<MTS> <HiiInfo>] <TS>
                           <CmtOrEol>
<HiiInfo>              ::= <TS> "##" <TS> "L" <QuotedString> <FS> <CName>
                           <Offset>
<ValuePcd>             ::= <TS> <PcdName> <FS> <ValUse>
<ValUse>               ::= <AsBuiltValue> <TailCmt>
<ValueOffsetPcd>       ::= [<NUsageBlock>]
                           <TS> <PcdName> <FS> <ValOffUse>
<ValOffUse>            ::= <AsBuiltValue> <Offset> <TailCmt>
<Offset>               ::= <FS> {<LongNum>} {<UINT32>}
<AsBuiltByteArray>     ::= "{" <NList> "}"
<AsBuiltValue>         ::= if (pcddatumtype == "BOOLEAN"):
                             {"0x00"} {"0x01"}
                           elif (pcddatumtype == "UINT8"):
                             <UINT8z>
                           elif (pcddatumtype == "UINT16"):
                             <UINT16z>
                           elif (pcddatumtype == "UINT32"):
                             <UINT32z>
                           elif (pcddatumtype == "UINT64"):
                             <UINT64z>
                           else:
                             <AsBuiltByteArray>
<PcdEntries>           ::= [<NUsageBlock>]
                           <TS> <PcdName> [<PField1>] <TailCmt>
<PField1>              ::= <FS> [<Value>] [<FFE>]
<Value>                ::= if (pcddatumtype == "BOOLEAN"):
                             <Boolean>
                           elif (pcddatumtype == "UINT8"):
                             {<NumValUint8>} {<Expression>}
                           elif (pcddatumtype == "UINT16"):
                             {<NumValUint16>} {<Expression>}
                           elif (pcddatumtype == "UINT32"):
                             {<NumValUint32>} {<Expression>}
                           elif (pcddatumtype == "UINT64"):
                             {<NumValUint64>} {<Expression>}
                           else:
                             {<StringVal>} {<Expression>}
<FFE>                  ::= <FS> <FeatureFlagExpress>
<1UsageBlock>          ::= <CommentBlock>
<NUsageBlock>          ::= <CommentBlock>+
<FeatureFlagExpress>   ::= <Boolean>
<CommentBlock>         ::= <TS> ["##" <TS> <Usage>] <TS> <CmtOrEol>
<CmtOrEol>             ::= {<Comment>} {<EOL>}
<Usage>                ::= {"CONSUMES"} {"SOMETIMES_CONSUMES"}
                           {"PRODUCES"} {"SOMETIMES_PRODUCES"}
                           {"UNDEFINED"}
```

Parameters

FeatureFlagExpress

The feature flag expression determines whether the entry line is valid. If the expression evaluates to `FALSE`, then the entry line is ignored by the EDK II build system.

1UsageBlock and **NUsageBlock**

The 1UsageBlock location, after the entry, is preferred if there is only one Usage for the PCD entry (this may also be referred to as a tail comment). If a PCD has multiple usages, then all CommentBlock statements must precede the entry.

Values

If a value is specified in an element and no value is set in the platform file, the platform will use the value specified here, rather than the default value specified in the DEC file that declares the PCD. The value must always match the Datum type, as specified in the DEC file. When specifying a value for PCD here, expression or macros are not permitted; only actual values are permitted.

UNDEFINED

Typically, this entry will be used when tools creating/installing UEFI Distribution Packages encounter a missing or misspelled usage.

CONSUMES

This module always gets the PCD entry. This is the only usage allowed for Feature PCDs.

PRODUCES

The module always sets the PCD entry.

SOMETIMES_CONSUMES

The module gets the PCD entry under certain conditions or execution paths.

SOMETIMES_PRODUCES

The module sets the PCD entry under certain conditions or execution paths.

AsBuiltByteArray

A byte array containing exactly the number of bytes (as specified as the maximum number of bytes in the DSC file) used for the patchable in module PCD when the binary was created. Any additional bytes for a value of less than the maximum number of bytes will be zero filled. For example, if the actual value of the array was only 4 bytes, but 10 bytes were allocated during the build, the tools will zero fill remaining bytes (in the example, 6 additional bytes of 0x00 will be added).

Examples

```
[FixedPcd]
  gEfiMdePkgTokenSpaceGuid.PcdFSBClock|600000000
  gEfiMdePkgTokenSpaceGuid.PcdMaximumUnicodeStringLength

[FeaturePcd]
  gEfiMdePkgTokenSpaceGuid.PcdComponentNameDisable|FALSE
  gEfiMdePkgTokenSpaceGuid.PcdDriverDiagnosticsDisable

[Pcd.IA32]
  gEfiNt32PkgTokenSpaceGuid.PcdWinNtMemorySizeForSecMain

[PatchPcd.IA32]
  ## @AsBuilt
  gEfiMdePkgTokenSpaceGuid.PcdDebugPrintErrorLevel | 0x80000040 |0x00004118
```

3.9 [Sources] Sections

These sections are optional.

Summary

Defines the `[Sources]` section content.

All file names specified in this section must be in the directory containing the INF file or in sub-directories of the directory containing the INF file.

The '`common`' architecture modifier in a section tag must not be combined with other architecture type; doing so will result in a build break.

All paths are relative to the directory containing the INF file. If the filename is listed as myfile.c, the file must be located in the same directory as the INF file. Absolute paths in the filename are prohibited.

There can be multiple sources sections, depending on the target processor. Example sources sections are listed below. The parsing utility creates a directory path for each file (`$(DEST_DIR)...\MyFile.c`), and looks up the makefile template for the `COMPONENT_TYPE` (EDK) or `MODULE_TYPE` (EDK II) to emit.

It is not permissible to mix EDK and EDK II style files within a module.

The macro, `TABLE_NAME` may be used in existing EDK INF files that point to ACPI tables, this value will be ignored by EDK II build tools.

All HII Unicode format files must be listed in this section as well as any other "source" type file, such as local module header files, Vfr files, etc.

Each source file must be listed only once per section. Files listed in architectural sections are not permitted to be listed in the common architectural section.

This section is not valid for a generated "As Built" binary INF file.

```
<Sources>            ::= "[Sources" [<com_attribs>]* "]" <EOL>
                         [<TS> "TABLE_NAME" <Eq> <SimpleWord> <EOL>]
                         <SourceFileStmts>*
<com_attribs>        ::= {".common"} {<attribs>}
<attribs>            ::= <attrs> ["," <TS> "Sources" <attrs>]*
<attrs>              ::= "." <arch>
<SourceFileStmts>    ::= {<MacroDefinition>} {<SourceFileEntry>}
<SourceFileEntry>    ::= <TS> <Filename> [<Options>] <EOL>
<Options>            ::= <FS> [<Family>] [<opt1>]
<opt1>               ::= <FS> [<TagName>] [<opt2>]
<opt2>               ::= <FS> [<ToolCode>] [<opt3>]
<opt3>               ::= <FS> [<FeatureFlagExpress>]
<Family>             ::= {"MSFT"} {"GCC"} {"INTEL"} {<Wildcard>}
<TagName>            ::= {<ToolWord>} {"*"}
<ToolCode>           ::= _CommandCode_
<FeatureFlagExpress> ::= <Boolean>
```

Parameters

Filename

Paths listed in the filename elements of the `[Sources]` section must be relative to the directory the INF file resides in. Use of "..", "." and "../" in the directory path is not permitted.

FeatureFlagExpress

When present, the feature flag expression determines whether the entry line is valid. If the feature flag expression evaluates to FALSE, this entry will be ignored by the EDK II build tools.

TagName

A keyword that uniquely identifies a tool chain group; the second field. Wildcard characters are permitted if and only if a command is common to all tools that will be used by a developer. As an example, if the development team only uses IA32 Windows workstations, the ACPI compiler can be specified as:

`DEBUG_*_*_ASL_PATH` and `RELEASE_*_*_ASL_PATH`

CommandCode

A keyword that uniquely identifies a specific command; the fourth field. Several CommandCode keywords have been predefined, however users may add additional keywords, with appropriate modifications to buildrule.txt. See table below for the pre-defined keywords and functional mappings. The wildcard character, "", is permitted only for the *`FAMILY`*, *`DLL`* and *`DPATH`* attributes (see *_Attributes** below.)

Family

`Family` is keyword that uniquely identifies a tool chain family. The `Family` must be either a wildcard character (meaning any Family) or it must match a defined value for a Family label in the `tools_def.txt` file for at least one tool chain `TagName` specified in `tools_def.txt` (or the `TagName` field that follows this field in the entry).

Example

```
[Sources.common]
  Diskio.c
  Diskio.h
  ComponentName.c

[Sources.IA32}
  Ia32DiskIo.h
```

3.10 [UserExtensions] Sections

These are optional sections.

Summary

Defines the optional EDK II INF file `[UserExtensions]` section tag. The build tools must have an a priori knowledge of how to process any items in this section.

Each UserExtensions section must have a unique set of `UserId` , `IdString` and Arch values.

The "common" architecture modifier in a section tag must not be combined with other architecture type; doing so will result in a build break.

This means that the same `UserId` can be used in more than one section, provided the `IdString` or Arch values are different. The same `IdString` values can be used if the `UserId` or Arch values are different. The same `UserId` and the same `IdString` can be used if the Arch values are different.

Any `[UserExtensions]` sections that are present in the source INF with a UserId of "TianoCore" will be copied into the "As Built" INF file. `[UserExtensions]` sections with other UserId values will not be copied to the "As Built" INF file.

Files listed in a `[UserExtensions.TianoCore."ExtraFiles"]` section must be included in a UEFI Distribution Package.

```
<UserExtensions> ::= "[UserExtensions" <com_attribs> "]" <EOL>
                     <statements>*
<com_attribs>    ::= {<com_arch>} {<attribs>}
<com_arch>       ::= <IdContent> [".common"]
<attribs>        ::= <IdContent> ["," <TS> "UserExtensions"
                     <IdContent>]*
<IdContent>      ::= <UserId> <IdString> [<attrs>]
<attrs>          ::= "." <arch>
<UserId>         ::= "." {(a-zA-Z)(a-zA-Z0-9_.)*} {"TianoCore"}
<IdString>       ::= "." {<NormalizedString>} {<SimpleWord>}
                     {<ReservedWord>}
<ReservedWord>   ::= {"PRE_PROCESS"} {"POST_PROCESS"}
                     {"ExtraFiles"}
<statements>     ::= Content is build tool chain specific.
```

Parameters

UserId

Words that contain period "." must be encapsulated in double quotation marks.

IdString

Normalized strings that contain period "." or space characters must be encapsulated in double quotation marks. The `IdString` must start with a letter.

Example

```
[UserExtensions.Edk2AcpiTable."1.0"]
   Any content may go here
```

3.10.1 [UserExtensions.TianoCore."ExtraFiles"] Section

This is an optional section.

Defines the optional EDK II INF file `[UserExtensions.TianoCore."ExtraFiles"]` section tag. The EDK II build tools must not process any files listed in this section.

Summary

This section is used by the Intel(R) UEFI Packaging Tool, that is distributed as part of the EDK II BaseTools, to locate files listed under this section header and add them to the UEFI distribution package. When installing a UEFI distribution package, these files will be installed in the module's directory tree.

Prototype

```
<UserExtensions> ::= "[UserExtensions" <TcEf> "]" <EOL> <FileNames>*
<TcEf>           ::= ".TianoCore." <DblQuote> "ExtraFiles" <DblQuote>
<FileNames>      ::= <TS> [<RelativePath>] <File> <EOL>
```

Parameters

FileNames

Paths listed in the filename elements of the this section must be relative to the directory the INF file resides in. Use of "..", "." and "../" in the directory path is not permitted.

Example

```
[UserExtensions.TianoCore."ExtraFiles"]
  Readme.txt
```

3.11 [Protocols] Sections

These are optional sections. If the source module code contains any protocols, then this section must be generated in the "As Built" INF file listing each protocol with its `<CommentBlock>` content.

Summary

Defines the optional EDK II INF file `[Protocols]` section tag. This is a list of the global PROTOCOL C Names that are referenced in the EDK II Module's C code.

Each protocol must be listed only once per section. Protocols listed in architectural sections are not permitted to be listed in the common architectural section.

The "common" architecture modifier in a section tag must not be combined with other architecture type; doing so will result in a build break.

The format of the `<CommentBlock>` is the recommended format that will guarantee that the information is correctly inserted into UEFI Distribution Package description files by the Intel(R) UEFI Packaging Tool included in the EDK II base tools project. The usages in the comment block describe how the Protocol is used in the C code.

A binary INF file must not contain any `FeatureFlagExpression` content.

Prototype

```
<Protocols>          ::= "[Protocols" [<com_attribs>] "]" <EOL>
                         <ProtoStatments>*
<com_attribs>        ::= {".common"} {<attribs>}
<attribs>            ::= <attrs> ["," <TS> "Protocols" <attrs>]*
<attrs>              ::= "." <arch>
<ProtoStatements>    ::= [<NUsageBlock>]
                         <TS> <ProtocolSpec> {<1UsageBlock>} {<EOL>}
<ProtocolSpec>       ::= <CName> [<FS> <FeatureFlagExpress>]
                         <1
UsageBlock>          ::= <CommentBlock>
<NUsageBlock>        ::= <CommentBlock>+
<FeatureFlagExpress> ::= <Boolean>
<CommentBlock>       ::= <TS> <UsageField> <TS> [<Comment>] <EOL>
<UsageField>         ::= ["##" <TS> <Usage> <TS>] [<Notify>]
<Notify>             ::= "##" <TS> "NOTIFY" <TS>
<Usage>              ::= {"PRODUCES"} {"SOMETIMES_PRODUCES"}
                         {"CONSUMES"} {"SOMETIMES_CONSUMES"}
                         {"TO_START"} {"BY_START"} {"UNDEFINED"}
```

Parameters

FeatureFlagExpress

When present, the feature flag expression determines whether the entry line is valid. If the feature flag expression evaluates to FALSE, this entry will be ignored by the EDK II build tools.

1UsageBlock and NUsageBlock

The `1UsageBlock` location, after the entry, is preferred if there is only one usage for the Protocol entry. If a Protocol has multiple usages, then all `CommentBlock` statements must precede the entry.

UNDEFINED

Typically, this entry will be used when tools creating/installing UEFI Distribution Packages encounter a missing or misspelled usage. `UNDEFINED` is also valid when the Protocol is not used as a Protocol and the GUID value of the Protocol is used for something else.

CONSUMES

This module does not install the protocol, but needs to locate a protocol. Not valid if the `Notify` attribute is true.

PRODUCES

This module will install this protocol. Not valid if the `Notify` attribute is true.

SOMETIMES_CONSUMES

This module does not install the protocol, but may need to locate a protocol under certain conditions, (such as if it is present.) If the `Notify` attribute is set, then the module will use the protocol, named by GUID, via a registry protocol notify mechanism.

SOMETIMES_PRODUCES

This module will install this protocol under certain conditions. Not valid if the `Notify` attribute is true.

TO_START

The protocol is consumed by a Driver Binding protocol Start function. Thus the protocol is used as part of the UEFI driver model. Not valid if the `Notify` attribute is true.

BY_START

The protocol is produced by a Driver Binding protocol Start function. Thus the protocol is used as part of the UEFI driver model. Not valid if the `Notify` attribute is true.

NOTIFY

This specifies whether this is a Protocol or ProtocolNotify. If set, then the module will use this protocol, named by GUID, via a registry protocol notify mechanism.

Example

```
[protocols]
  gEfiDecompressProtocolGuid
  gEfiLoadFileProtocolGuid
```

3.12 [Ppis] Sections

These are optional sections. If the source module code contains any PPIs, then this section must be generated in the "As Built" INF file listing each PPI with its `<CommentBlock>` content.

Summary

Defines the EDK II INF file `[PPIs]` section content. This is a list of the global PPI C Names that are referenced in the EDK II Module's C code.

Each PPI must be listed only once per section. PPIs listed in architectural sections are not permitted to be listed in the common architectural section.

The "common" architecture modifier in a section tag must not be combined with other architecture type; doing so will result in a build break.

The format of the `<CommentBlock>` is the recommended format that will guarantee that the information is correctly inserted into UEFI Distribution Package description files by the Intel(R) UEFI Packaging Tool included in the EDK II base tools project. The usages in the comment block describe how the PPI is used in the C code.

A binary INF file must not contain any `FeatureFlagExpression` content.

```
<Ppis>                ::= "[Ppis" [<com_attribs>] "]" <EOL> <PpiStatements>*
<com_attribs>         ::= {".common"} {<attribs>}
<attribs>             ::= <attrs> ["," <TS> "Ppis" <attrs>]*
<attrs>               ::= "." <arch>
<PpiStatements>       ::= [<NUsageBlock>]
                          <TS> <PpiSpec> [<1UsageBlock>]
<PpiSpec>             ::= <CName> [<FS> <FeatureFlagExpress>]
                          <1
UsageBlock>           ::= <CommentBlock>
<NUsageBlock>         ::= <CommentBlock>+
<FeatureFlagExpress> ::= <Boolean>
<CommentBlock>        ::= <TS> <UsageField> <TS> [<Comment>] <EOL>
<UsageField>          ::= ["##" <TS> <Usage> <TS>] [<Notify>]
<Notify>              ::= "##" <TS> "NOTIFY" <TS>
<Usage>               ::= {"PRODUCES"} {"SOMETIMES_PRODUCES"}
                          {"CONSUMES"} {"SOMETIMES_CONSUMES"}
                          {"UNDEFINED"}
```

Parameters

FeatureFlagExpress

When present, the feature flag expression determines whether the entry line is valid. If the feature flag expression evaluates to FALSE, this entry will be ignored by the EDK II build tools.

1UsageBlock and **NUsageBlock**

The `1UsageBlock` location, after the entry, is preferred if there is only one usage for the PPI entry. If a PPI has multiple usages, then all `CommentBlock` statements must precede the entry.

UNDEFINED

Typically, this entry will be used when tools creating/installing UEFI Distribution Packages encounter a missing or misspelled usage. `UNDEFINED` is also valid when the PPI is not used as a PPI and the GUID value of the PPI is used for something else.

CONSUMES

This module does not install the PPI, but needs to locate a PPI. Not valid if the `Notify` true.

PRODUCES

This module will load this PPI. Not valid if the `Notify` attribute is true.

SOMETIMES_CONSUMES

This module does not install the PPI, but may need to locate a PPI under certain conditions or execution paths. If the `Notify` attribute is set, then the module will use the PPI, named by GUID, via a registry PPI notify mechanism.

SOMETIME_PRODUCES

This module will load this PPI under certain conditions or execution paths. Not valid if the `Notify` attribute is true.

NOTIFY

This specifies whether this is a Ppi or PpiNotify. If set to, the module requires or consumes a PPI, named by GUID, via a register PPI notify mechanism.

Example

```
[ppis]
  gEfiPeiMemoryDiscoveredPpiGuid
  gEfiFindFvPpiGuid
```

3.13 [Guids] Sections

These are optional sections. If the source module code contains any GUIDs (other than PPI or PROTOCOL GUIDs), then this section must be generated in the "As Built" INF file listing each GUID with its `<CommentBlock>` content.

Summary

Defines the EDK II INF file `[Guids]` section content. This is a list of the global GUID C Names that are referenced in the module's code, but not already referenced in the INF. Unique GUID C names may be published in the DEC file (or come from a Distribution Package surface area description.) The C Names used in this section are formatted using the external name.

The "common" architecture modifier in a section tag must not be combined with other architecture type; doing so will result in a build break.

Each GUID must be listed only once per section. GUIDs listed in architectural sections are not permitted to be listed in the common architectural section.

The format of the `<CommentBlock>` is the recommended format that will guarantee that the information is correctly inserted into UEFI Distribution Package description files by the Intel(R) UEFI Packaging Tool included in the EDK II base tools project. The usages in the comment block describe how the GUID is used in the C code.

A "binary" INF file must not contain any `FeatureFlagExpression` content.

```
<Guids>              ::= "[Guids" [<com_attribs>] "]" <EOL> <GuidsStatements>*
<com_attribs>        ::= {".common"} {<attribs>}
<attribs>            ::= <attrs> [ "," <TS> "Guids" <attrs>]*
<attrs>              ::= "." <arch>
<GuidsStatements>    ::= [<NUsageBlock>}
                         <TS> <GuidSpec> [<1UsageBlock>]
<GuidSpec>           ::= <CName> [<FS> <FeatureFlagExpress>]
                         <1
UsageBlock>          ::= <CommentBlock>
<NUsageBlock>        ::= <CommentBlock>+
<FeatureFlagExpress> ::= <Boolean>
<CommentBlock>       ::= <TS> [<UseOrFieldOpt>] <CmtOrEol>
<UseOrFieldOpt>      ::= [<UsageField>] [<GuidTypeField>]
<UsageField>         ::= "##" <TS> <Usage> <TS>
<GuidTypeField>      ::= "##" <TS> <GuidType> <TS>
<CmtOrEol>           ::= {<Comment>} {<EOL>}
<Usage>              ::= {"CONSUMES"} {"SOMETIMES_CONSUMES"}
                         {"PRODUCES"} {"SOMETIMES_PRODUCES"}
                         {"UNDEFINED"}
<GuidType>           ::= {"Event"} {"File"} {"FV"} {"GUID"} {"HII"}
                         {"HOB"} {"SystemTable"} {"TokenSpaceGuid"}
                         {<VariableType>} {"UNDEFINED"}
<VariableType>       ::= {"Variable"}
                         {"Variable:" <TS> <VariableName>}
<VariableName>       ::= <UnicodeString>
```

Parameters

FeatureFlagExpress

When present, the feature flag expression determines whether the entry line is valid. If the feature flag expression evaluates to FALSE, this entry will be ignored by the EDK II build tools.

1UsageBlock and **NUsageBlock**

The `1UsageBlock` location, after the entry, is preferred if there is only one If a GUID has multiple usages, then all `CommentBlock` statements must precede the entry.

Usage

One or more `Usage` comment lines may be specified for a given GUID. If more than one `Usage`, then the comment section following the `Usage` must be provided, explaining the different usages.

UNDEFINED - Usage

Typically, this entry will be used when tools creating/installing UEFI Distribution Packages encounter a missing or misspelled usage. UNDEFINED may also be used for GUIDs that identify different types of information.

CONSUMES

`CONSUMES` means that a module may use this GUID that does not fit into the defined PROTOCOL or PPI types. The module will use the named GUID. For GUID type of:

- Event - `CONSUMES` means that the module has an event waiting to be signaled (i.e., the module registers a notification function and calls the function when it is signaled.
- System Table - this means that the module will use a GUIDed entry in the system table.
- Variable - this means that the module may use the variable entry.
- HII - the formset may be registered into HII by this module.
- TokenSpaceGuid - this means that an Token Space GUID will be required for PCD entries used in this module.
- Hob - this means that a HOB may need to be present in the system.
- File/FV - this means that a file must be present in an FV, such as a module that loads a processor microcode patch file.
- GUID - this means that a module may use this GUID that does not fit into the defined GUID types.

PRODUCES

`PRODUCES` means that a module will produce a GUID that does not fit into the defined PROTOCOL or PPI types. This module always produces a named GUID. For GUID type of:

- Event - this means that module will signal all events in an event group.
- System Table - this means that the module will produce a GUIDed entry in the system table.
- Variable - this means that the module will write the variable.
- HII - `PRODUCES` is not valid for this GUID type.
- TokenSpaceGuid - `PRODUCES` is not valid for this GUID type.
- Hob - this means that the HOB will be produced by the module.
- File/FV - this means that a module creates a file that is present in an FV, such as a file that contains a microcode patch.
- GUID - this means that a module will produce a GUID that does not fit into the defined PROTOCOL, PPI or GUID types.

Example

```
[Guids]
  gEfiDebugImageInfoTable
  gEfiHobMemoryAllocModuleGuid

  gEfiAbcVariableGuid           ## PRODUCES ## Variable:L"XYZ" # Sets the variable

  # Event registered to EFI_HII_SET_KEYBOARD_LAYOUT_EVENT_GUID group,
  # which will be triggered by EFI_HII_DATABASE_PROTOCOL.SetKeyboardLayout().
  ## SOMETIME_CONSUMES ## Event
  gEfiHiiKeyBoardLayoutGuid
```

3.14 [Depex] Sections

These are optional sections

Summary

Defines the optional EDK II INF file `[Depex]` section content. The `[Depex]` section is a replacement for the dependency file specified by the driver writer. The `DPX_SOURCE` in the `[Defines]` section an EDK INF file will over-ride the dependency specified here. This section can be used for inheritance from libraries, by supporting logical AND'ing of the different Depex expressions together.

The Rules would be as follows:

- EDK II INF - `[Depex]` section and inheritance from libraries is supported via AND'ing the different Depex expressions together
- EDK II INF - The `[Defines]` section's keyword, `DPX_SOURCE`, would override Depex section and let module owner force a Depex independent of the `[Depex]` inheritance. Not recommended, but gives complete control to the driver writer.
- Each `[Depex]` section tag listed in an INF file must be unique. If there are multiple `[Depex]` sections that have the same section tag, i.e., `[Depex.IA32.DXE_DRIVER]` and another `[Depex.IA32.DXE_DRIVER]` section in the same INF, the build must break.

If a `DPX_SOURCE` is specified in the `[Defines]` section, the `[Depex]` section is ignored, and the file specified in the `DPX_SOURCE` is used instead.

When processing the file, the INF file name specified in the `<GuidStmt>` and `<DepInstruct>` statement is replaced by the `FILE_GUID` value from the INF file, translated to a POSIX C structure as shown below:

```
INTERFACENAME = { /* 0F05DE03-8A1B-408C-8F84-B547F593E463 */
    0x0F05DE03,
    0x8A1B,
    0x408C,
    {0x8F, 0x84, 0xB5, 0x47, 0xF5, 0x93, 0xE4, 0x63}
  };
```

The term, "SOR" is ignored as part of the dependency processing. The DXE driver is to remain on the Schedule on Request (`SOR`) queue until the DXE Service `Schedule()` is called for this DXE. The dependency expression evaluator treats this operation like a No Operation (`NOP`).

There are three types of dependency sections (PEI, SMM and DXE) permitted by specifications. The SMM dependency section uses the same grammar as the DXE dependency section. The optional tags (identified as `<DepexType>` in the EBNF, below) must be used at the start of a depex listing. The depex expression for a given type is terminated by the start of a new optional section tag, the start of a new section or the end of file.

Drivers with `MODULE_TYPE` set to `SEC` , `PEI_CORE` , `DXE_CORE` , `SMM_CORE` , `UEFI_DRIVER` and `UEFI_APPLICATION` cannot have `[Depex]` sections. Libraries and modules that are `USER_DEFINED` may have a `[Depex]` section. All remaining drivers, `PEIM` , `DXE_DRIVER` , `DXE_SAL_DRIVER` , `DXE_RUNTIME_DRIVER` and `DXE_SMM_DRIVER` module types must have a `[Depex]` section.

Libraries of type `SEC` , `PEI_CORE` , `DXE_CORE` , `SMM_CORE and` `UEFI_APPLICATION` are not allowed to have a `[Depex]` . The `MODULE_TYPE` entry in the `[Defines]` section for a library only defines the module type that the build system must assume for building the library. It does not define the types of modules that a library may be linked with. Instead, the `LIBRARY_CLASS` entry in the `[Defines]` section optionally lists the supported module types that the library may be linked against.

Libraries of type `BASE` are not permitted to have generic (i.e., `[Depex]`) and generic with only architectural modifier (i.e., `[Depex.IA32]`) entries. Library of type `BASE` are permitted to have a Depex section if one ModuleType modifier is specified (i.e., `[Depex.common.PEIM`).

When using the ModuleType as a section modifier (for example: [Depex.IA32.PEIM]), for drivers, the ModuleType must match the value of `MODULE_TYPE` entry in the `[Defines]` section. For Libraries, the ModuleType used in the section modifier must be a member of the Module Types listed after the `LIBRARY_CLASS` keyword in the `[Defines]` section. If no module types are listed after the `LIBRARY_CLASS` keyword in the `[Defines]` section, then the library is compatible with all module types, so all module types may be used as a section modifier.

The "common" architecture modifier in a section tag must not be combined with other architecture type; doing so will result in a build break.

If the `MODULE_TYPE` is `UserDefined`, the build tools must exit gracefully and provide the user with an error message stating that the `[Depex]` section header does not provide enough information to determine the type of the Depex section.

If the module is not a library (no `LIBRARY_CLASS` in the `[Defines]` section) and the `MODULE_TYPE` is `SEC`, `SMM_CORE`, `DXE_CORE`, `PEI_CORE`, `UEFI_DRIVER` or `UEFI_APPLICATION` a Depex section is not permitted. If one is found, the build tools must exit gracefully and provide the user with an error message stating the `[Depex]` section is not valid for the `MODULE_TYPE`

If the module is a library (with a `LIBRARY_CLASS` statement in the `[Defines]` section) and there is no module type defined in Depex section's modifier and there is a MTL defined (with a ModuleTypeList statement following by the `LIBRARY_CLASS` statement in the `[Defines]` section) and each module type in MTL is `PEIM,`

`DXE_DRIVER`, `DXE_SAL_DRIVER`, `DXE_RUNTIME_DRIVER`, `DXE_SMM_DRIVER` or `UEFI_DRIVER` the build tools must create a related Depex section for each module type in XML.

If the module is a library and the `MODULE_TYPE` is not `BASE` (with a `LIBRARY_CLASS` statement in the `[Defines]` section) and it has no MTL defined (without a `ModuleTypeList` statement following by the `LIBRARY_CLASS` statement in the `[Defines]` section) a Depex section is not permitted. If one is found, the build tools must exit gracefully and provide the user with an error message stating the `[Depex]` section is not valid for the module.

If the module is a library (with a `LIBRARY_CLASS` statement in the [Defines] section) and the `MODULE_TYPE` is `UEFI_DRIVER`, the `[Depex]` section must map to a `DxeDepex` section in the XML.

For a binary INF file, the `[Depex]` section will contain the full dependency expression, including the dependencies from the linked libraries in a comment.

Note: A UEFIDRIVER which is not included in an FD image (such as a driver that will be loaded from the shell or stored in a PCI option ROM) will not have an FFS DEPEX section generated by the tools.

Note: Capitalization in the Prototypes listed below does not match the capitalization used in the PI Specification.

Prototype

```
<Depex>              ::= "[Depex" [<com_attribs>] "]" <EOL>
                         {<DepexSection>*} {<AsBuiltDepex>}
<com_attribs>        ::= {".common"} {<attribs>}
<attribs>            ::= <attrs> ["," <TS> "Depex" <attrs>]*
<attrs>              ::= "." <arch> [<Module>]
```

```
<Module>               ::= "." <DepexModuleType>
<DepexModuleType>      ::= {"PEIM"} {"DXE_DRIVER"} {"DXE_SMM_DRIVER"}
                           {"DXE_RUNTIME_DRIVER"} {"DXE_SAL_DRIVER"}
                           {"UEFI_DRIVER"} {"USER_DEFINED"}
<DepexSection>         ::= {<PeiDepex>} {<SmmDepex>} {<DxeDepex>}
<DxeDepex>             ::= <DxeDepexStatements>+ ["END" <EOL>]
<DxeDepexStatements>   ::= {<SorStmt>} {<GuidStmt>} {<BoolStmt>}
<PeiDepex>             ::= <PeiDepexStatements>*
                           ["END" <EOL>]
<PeiDepexStatements>   ::= {<BoolStmt>} {<DepInstruct>}
<SmmDepex>             ::= <DxeDepex>
<GuidStmt>             ::= {"BEFORE"} {"AFTER"} <GuidCName> [<EOL>]
<DepInstruct>          ::= "PUSH" <CFormatGUID> [<EOL>]
<SorStmt>              ::= "SOR" <BoolStmt> [<EOL>]
<BoolStmt>             ::= {<Bool>} {<BoolExpress>}
<Bool>                 ::= {"TRUE"} {"FALSE"} {<GuidCName>} [<EOL>]
<GuidCName>            ::= <CName> # A Guid C Name
<NTs>                  ::= "NOT" <TS>
<BoolExpress>          ::= <NTs> <Bool> [<BoolExpressCont>]*
<BoolExpressCont>      ::= <MTS> {"AND"} {"OR"} <MTS> [<NTs>] <Bool>
<AsBuiltDepex>         ::= "#" {<FullDxe>} {<FullPei>} {<FullSmm>} <EOL>
<FullDxe>              ::= <DxeStatements>+
                           [<TS> "END"]
<DxeStatements>        ::= {<SorAbStmt>} {<GuidAbStmt>} {<BoolAbStmt>}
<FullPei>              ::= <PeiStatements>*
                           [<TS> "END"]
<PeiStatements>        ::= {<BoolAbStmt>} {<DepAbInstruct>}
<FullSmm>              ::= <FullDxe>
<GuidAbStmt>           ::= {"BEFORE"} {"AFTER"} <TS> <GuidCName> <TS>
<DepAbInstruct>        ::= "PUSH" <TS> <CFormatGUID> <TS>
<SorAbStmt>            ::= "SOR" <TS> <BoolAbStmt> <TS>
<BoolAbStmt>           ::= {<BoolAb>} {<BoolAbExpress>}
<BoolAb>               ::= {"TRUE"} {"FALSE"} {<GuidCName>} <TS>
<GuidCName>            ::= <CName> # A Guid C Name
<BoolAbExpress>        ::= <NTs> <Bool> [<BoolAbExpressCont>]*
<BoolAbExpressCont>    ::= <MTS> {"AND"} {"OR"} <MTS> [<NTs>] <Bool>
```

Example

```
[Depex]
  gEfiFirmwareVolumeBlockProtocolGuid
  AND gEfiAlternateFvBlockGuid
  AND gEfiFaultTolerantWriteLiteProtocolGuid

[Depex]
  SOR gEfiProtocolIDependOnGuid

[Depex.common]
  SOR gEfiProtocolIDependOnGuid

[Depex.IA32.DXE_SMM_DRIVER]
  TRUE

[Depex.IA32.DXE_DRIVER]
  TRUE AND gEfiAlternateFvBlockGuid
```

3.15 [Binaries] Section

Summary

Defines the `[Binaries]` tag is required for EDK II INF files for Binary Modules.

This is a required section for Binary Modules Only.

Each binary file must be listed only once per section. Files listed in architectural sections are not permitted to be listed in the common architectural section.

The "common" architecture modifier in a section tag must not be combined with other architecture type; doing so will result in a build break.

There can be multiple `[Binaries]` sections, depending on the target processor. Example binaries sections are listed below. Each binary file's path is relative to the location of the component's INF file. The parsing utility creates a directory path for each file (`$(DEST_DIR)/Path/OUTPUT`), and copies each file (or a processed version of a Unicode User Interface or Version section) to the `OUTPUT` directory. No makefile is produced, as binary files are only used by the third phase of a build, creating FV, FD or similar binary files.

All file names specified in this section must be in the directory containing the INF file or in sub-directories of the directory containing the INF file.

When a binary INF file is generated by tools during a source build, if a symbol file, such as a PDB or SYM file is generated, the tools must add the file to this section using a binary file type of "DISPOSABLE".

One and only one `EFI_SECTION_VERSION` is allowed in a FFS image, therefore one and only one `VER` (`VER` or `UNI_VER`) can be included in any one `[Binaries]` section.

One and only one `EFI_SECTION_USER_INTERFACE` is allowed in a FFS image, therefore one and only one `UI` (`UI` or `UNI_UI`) can be included in any one `[Binaries]` section.

One and only one `EFI_SECTION_FREEFORM_SUBTYPE_GUID` is allowed in a FFS image, therefore one and only one `SUBTYPE_GUID` with a unique GUID Value can be included in any one `[Binaries]` section.

Prototype

```
<Binaries>          ::= "[Binaries" [<com_attribs>] "]" <EOL>
                        [<UiExpression>]
                        [<VerExpression>]
                        <BinariesStatements>*
<BinariesStatements> ::= {<MacroDefinition>} {<BinaryFiles>} {<SubTypeGuid>}
<com_attribs>       ::= {".common"} {<attribs>}
<attribs>           ::= <attrs> ["," <TS> "Binaries" <attrs>]*
<attrs>             ::= "." <arch>
<UiExpression>      ::= <TS> <UiFile> [<UiOptions>] <EOL>
<UiOptions>         ::= <FS> [<Target>] [<FS> <FeatureFlagExpress>]
<UiFile>            ::= <UiType> <FS> <Filename>
<UiType>            ::= {"UNI_UI"} {"UI"}
<VerExpression>     ::= <TS> <VerFile> [<VerOptions>] <EOL>
<VerOptions>        ::= <FS> [<Target>] [<FS> <FeatureFlagExpress>]
<VerFile>           ::= <VerType> <FS> <Filename>
<VerType>           ::= {"UNI_VER"} {"VER"}
<SubTypeGuid>       ::= <TS> "SUBTYPE_GUID" <GuidOpts> <EOL>
<GuidOpts>          ::= <FS> <GuidValue> <FileOpts> <EOL>
<GuidValue>         ::= {<CName>} {<RegistryFormatGUID>}
<BinaryFiles>       ::= <TS> <FileType> <FileOpts> <EOL>
<FileOpts>          ::= <FS> <Filename> [<Options>] <EOL>
<Options>           ::= <FS> [<Target>] [<Opt1>]
<Opt1>              ::= <FS> [<Family>] [<Opt2>]
<Opt2>              ::= <FS> [<TagName>] [<Opt3>]
```

```
<Opt3>               ::= <FS> <FeatureFlagExpress>
<Target>             ::= {<ToolWord>} {<Wildcard>}
<Family>             ::= {"MSFT"} {"GCC"} {"INTEL"} {<Usr>}
                         {<Wildcard>}
<Usr>                ::= <ToolWord>
<TagName>            ::= {ToolWord} {<Wildcard>}
<FeatureFlagExpress> ::= <Boolean>
<FileType>           ::= <Edk2FileType>
<Edk2FileType>       ::= {"ACPI"} {"ASL"} {"PE32"} {"PIC"} {"FV"}
                         {"PEI_DEPEX"} {"DXE_DEPEX"} {"SMM_DEPEX"}
                         {"TE"} {"BIN"} {"RAW"} {"COMPAT16"}
                         {"DISPOSABLE"} {"LIB"}
```

Parameters

FeatureFlagExpress

When present, the feature flag expression determines whether the entry line is valid. If the feature flag expression evaluates to FALSE, this entry will be ignored by the EDK II build tools.

Note: For more information about the following parameters, refer to the Build Specification for a description of the tools_def.txt file. In order for the entries in the INF file to be valid, there must be a matching definition in the tools_def.txt file. The tool chain tag name must also match the one used for the build.

Target

A keyword that uniquely identifies the build target. This keyword is used to bind command flags to individual commands. Refer to the Build Specification for the exact format of the `tools_def.txt` file. The `tools_def.txt` file defines a label to specify different items such as executables, options and locations. The label is broken up into fields which are separated by an underscore character. The Target must be either a wildcard character (meaning all targets) or it must be specified in the first field of at least one of these labels. Three values, "NOOPT", "DEBUG" and "RELEASE", have been pre-defined.

Users may want to add other definitions, such as, PERF, SIZE or SPEED, and define their own set of FLAGS to use with these tags.

TagName

TagNames must also appear in at least one Label specified in the `tools_def.txt` file. The `TagName` must be either a wildcard character (meaning any TagName) or it must match a defined value for a `TagName` label in the `tools_def.txt` file for the tool chain tag name specified.

Family

`Family` is keyword that uniquely identifies a tool chain family. The `Family` must be either a wildcard character (meaning any `Family`) or it must match a defined value for a `Family` label in the `tools_def.txt` file for at least one tool chain `TagName` specified in `tools_def.txt` (or the `TagName` field that follows this field in the entry).

FileType: "SUBTYPE_GUID"

The file type, "SUBTYPE_GUID" is shorthand for the `EFI_FREEFORM_SUBTYPE_GUID_SECTION` section.

FileType: "DISPOSABLE"

The file type, "DISPOSABLE" does not represent content for the `EFI_SECTION_DISPOSABLE`. These files will not be processed by EDK II build tools, but rather, may specify other types of files that may be used such as PDB or SYMS files generated for symbolic debugging.

Example

```
[Binaries.common]
  UNI_UI|DxeIpl.ui
  UNI_VER|DxeLoad.ver

[Binaries.Ia32]
  DXE_DEPEX|Release/DxeIpl.dpx            # MYTOOLS
  PE32|Debug/Ia32/DxeIpl.efi|DEBUG        # MYTOOLS
  PE32|Release/Ia32/DxeIpl.efi|RELEASE    # MYTOOLS
  DISPOSABLE|Debug/Ia32/DxeIpl.pdb|DEBUG

[Binaries.X64]
  DXE_DEPEX|Debug/X64/DxeIpl.dpx          # MYTOOLS
  PE32|Debug/X64/DxeIpl.efi|DEBUG         # MYTOOLS
  DISPOSABLE|Debug/X64/DxeIpl.pdb|DEBUG

[Binaries.IPF]
  DXE_DEPEX|Debug/IPF/DxeIpl.dpx          # MYTOOLS
  PE32|Debug/Ipf/DxeIpl.efi|DEBUG         # MYTOOLS
  DISPOSABLE|Debug/Ipf/DxeIpl.pdb|DEBUG
```

APPENDIX A EDK INF FILE SPECIFICATION

This appendix covers the format of the original EDK INF files. However, the format of comments in the EDK INF may vary from this specification, as the original EDK parsing tool, ProcessDSC, only looked for a specific set of tokens. Due the extensive use of MACRO statements in the EDK components and libraries INF files, EDK INF files cannot be processed by tools to create a distribution that complies with the UEFI Platform Initialization Distribution Package Specification.

A.1 Design Discussion

Directive statements are permitted within the EDK INF files.

A.1.1 [defines] Section

The `[defines]` section of an EDK INF file is used to define variable assignments that can be used in later build steps. The EDK parsing utilities process local symbol assignments made in this section. Note that the sections are processed in the order listed here, and later assignments of these local symbols do not override previous assignments.

This section will typically use one of the following section definitions:

```
[define]
[defines]
[defines.IA32]
[defines.X64]
[defines.IPF]
[defines.EBC]
```

Note: The `[define]` section tag is only valid for EDK INF files. EDK II INF files must use the 'defines' keyword.

The format for entries in this section is:

`Name = Value`

The following is an example of this section.

```
[defines]
  BASE_NAME      = DiskIo
  FILE_GUID      = CA261A26-7718-4b9b-8A07-5178B1AE3A02
  COMPONENT_TYPE = BS_DRIVER
```

The following table lists the possible content of this section.

Table 6 EDK [defines] Section Elements

Tag	Required	Value	Notes
`BASE_NAME`	Yes	A single word	This is a single word identifier that will be used for the component name.
`COMPONENT_TYPE`	Yes	One of the EDK I Component Types	See Table EDK I Component (module) Types for possible values
`FILE_GUID`	No--Optional for Libraries, Required for all other component types	Guid Value	Registry (8-4-4-4-12) Format GUID

`EDK_RELEASE_VERSION`	Optional	Hex Value	A Hex version number, 0x00020000
`EFI_SPECIFICATION_VERSION`	No--Optional	HexValue	A Hex version number, 0x00020000
`MAKEFILE_NAME`	No--Optional	Filename.ext	The name of the Makefile to be generated for this component
`CUSTOM_MAKEFILE`	No--Optional	Filename.ext	This specifies the name of a custom makefile that should be used, instead of a generated makefile. **NOTE**: EDK INF components specifying a custom EDK style makefile cannot be used in an EDK II build.
`BUILD_NUMBER`	No--Optional	Set this four digit value in the generated Makefile	Normally not used in INF files.
`C_FLAGS`	No--Optional	Microsoft C Flags to use with for a cl commands for this module	Normally not used in INF files. Typically, an EDK INF file will provide a separate nmake section to specify different build parameters.
`FFS_EXT`	No--Optional	File Extension	The FFS extension to use for this component, refer to the table _EDK I Component (module) Types for the default FFS extension. This value is used to create a component PKG file.
`FV_EXT`	No--Optional	File Extension	The FV extension to use for this component, refer to the table _EDK I Component (module) Types for the default FV extension.
`SOURCE_FV`	No--Optional	Word	If present, the variable is set at the beginning of the generated makefile
`VERSION`	No--Optional	Four digit integer	If present, this value will be used for the VERSION section of the FFS.
`VERSION_STRING`	No--Optional	String	If present, this value will be used to generate the UNICODE file for the VERSION section of the FFS.

The following table lists the available `COMPONENT_TYPE` values supported by EDK INF files.

Table 7 EDK Component (module) Output File Extensions

COMPONENT_TYPE	EDK II Extension	EDK File, FFS or FV Extension	Description
`LIBRARY`	.lib	.lib	Library component linked as part of the build with other components.
`FILE`	From file name or .FFS	From file name or .FFS	Raw file copied directly to FV
`Apriori`	.bin	.SEC	This EDK INF component is not supported in the EDK II build - it is created from content in other EDK II build meta-data files.
`EFI Binary Executable`	.efi	.pe32	PE32/PE32+/Coff binary executable. The extension of the file has changed for EDK II builds which generate processed (GenFw)

`AcpiTable`	.acpi	.SEC	An ACPI Table.
`Legacy16`	.bin	.SEC	The MODULE_TYPE for a Legacy16 when migrating to EDK II should be specified as USER_DEFIND. The .rom or .bin file should be included under a [binaries] section. In EDK, the COMPONENT_TYPE of Legacy16 was mostly used to specify PCI Option ROMs.
`BINARY`	.bin	.FFS	
`CONFIG`	.bin	.SEC	
`LOGO`	.bin	.SEC	The MODULE_TYPE for a LOGO when migrating to EDK II should be specified as USER_DEFINED. The .bmp file should be include under a [binaries] section. In EDK, the COMPONENT_TYPE of LOGO was used to specify a .bmp file.
`RAWFILE`	.raw	.RAW	
`FVIMAGEFILE`	.fv	.FVI	
`SECURITY_CORE`	.efi	.SEC	Modules of this type are designed to start execution at the reset vector of a CPU. They are responsible for preparing the platform for the PEI Phase. Since there are no standard services defined for SEC, modules of this type follow the same rules as modules of type Base and typically include some amount of CPU specific assembly code to establish temporary memory for a stack. Modules of this type may optionally produce services that are passed to the PEI Phase in HOBs and those services must be compliant with the PEI CIS.
`PEI_CORE`	.efi	.PEI	This module type is used by PEI Core implementations that are complaint with the PEI CIS.
`COMBINED_PEIM_DRIVER`	.efi	.PEI	
`PIC_PEIM`	.efi	.PEI	
`RELOCATABLE_PEIM`	.efi	.PEI	When migrating to EDK II, this type of module should use the register for shadow PPI, and set the [defines] entry: `SHADOW = TRUE`
`PE32_PEIM`	.efi	.PEI	This module type is used by PEIMs that are compliant with the PEI CIS
`BS_DRIVER`	.efi	.DXE	This module type is either the DXE Core or DXE Drivers that are complaint with the DXE CIS. These modules only execute in the boot services environment and are destroyed when ExitBootServices() is called.
`RT_DRIVER`	.efi	.DXE	This module type is used by DXE Drivers that are complaint with the DXE CIS. These modules execute in both boot services and runtime services environments. This means the services that these modules produce are available after ExitBootServices() is called. If SetVirtualAddressMap() is called, then modules of this type are relocated according to virtual address map provided by the operating system.
			This module type is used by DXE Drivers that can be called in physical mode before

`SAL_RT_DRIVER`	.efi	.DXE	SetVirtualAddressMap() is called and either physical mode or virtual mode after SetVirtualAddressMap() is called. This module type is only available to IPF CPUs. This means the services that these modules produce are available after ExitBootServices().
`BS_DRIVER`	.efi	.SMM	This module type is used by DXE Drivers that are loaded into SMRAM. As a result, this module type is only available for IA-32 and x64 CPUs. These modules only execute in physical mode, and are never destroyed. This means the services that these modules produce are available after ExitBootServices().
`APPLICATION`	.efi	.APP	This module type is used by UEFI Applications that are compliant with the EFI 1.10 Specification or the UEFI 2.0 Specification. UEFI Applications are always unloaded when they exit.
`EFI USER INTERFACE`	.ui	.ui	
`EFI VERSION`	.ver	.ver	
`EFI DEPENDENCY`	.dpx	.dpx	

A.1.2 [sources] Section

The `[sources]` section is used to specify the files that make up the component. Directories names are required for files existing in subdirectories of the component. All directory names are relative to the location of the INF file. Macros are allowed in the source file path. For EDK builds, each file is added to the macro of `$(INC_DEPS)`, which can be used in a makefile dependency expression.

This section will typically use one of the following section definitions:

```
[sources]
[sources.common]
[sources.IA32]
[sources.X64]
[sources.IPF]
[sources.EBC]
```

The following example demonstrates entries in this section.

```
[sources.common]
  DxeIpl.dxs
  DxeIpl.h
  DxeLoad.c

[sources.Ia32]
  Ia32/VirtualMemory.h
  Ia32/VirtualMemory.c
  Ia32/DxeLoadFunc.c
  Ia32/ImageRead.c

[sources.X64]
  X64/DxeLoadFunc.c

[sources.IPF]
  Ipf/DxeLoadFunc.c
  Ipf/ImageRead.c
```

Binary file types - EDK does not have the flexibility of EDK II, but does provide a method for specifying binary files in the `[sources]` section. The following lists the mapping of EDK specific binary file types to EFI sections.

SEC_GUID

The binary file is an `EFI_SECTION_FREEFORM_SUBTYPE_GUID` section.

SEC_PE32

This binary is an `EFI_SECTION_PE32` section.

SEC_PIC

This binary is an `EFI_SECTION_PIC` section.

SEC_PEI_DEPEX

This binary is an `EFI_SECTION_PEI_DEPEX` section.

SEC_DXE_DEPEX

This binary is an `EFI_SECTION_DXE_DEPEX` section.

SEC_TE

This binary is an `EFI_SECTION_TE` section.

SEC_VER

This binary is an `EFI_SECTION_VERSION` section.

SEC_UI

This binary is an `EFI_SECTION_USER_INTERFACE` section.

SEC_BIN

The binary is an `EFI_SECTION_RAW` section.

SEC_COMPAT16

This binary is an `EFI_SECTION_COMPATIBILTY16` section.

A.1.3 [libraries] Section

The `[libraries]` section of the EDK INF is used to list the names of the libraries that will be linked into the EDK component. The library names do not include the directory locations or the extension name of the file. For each library, `{LibName}`, found, the `{LibName}` is added to the LIBS definition in the output makefile:

`LIBS = $(LIBS) $(LIB_DIR)\{LibName}`

This section will typically use one of the following section definitions:

```
[libraries.common]
[libraries.IA32]
[libraries.X64]
[libraries.IPF]
[libraries.EBC]
```

The formats for entries in this section is:

`LibraryName`

The following is an example of a libraries section.

```
[libraries.common]
```

```
  EfiProtocolLib
  EfiDriverLib
```

A.1.4 [includes] Section

The `[includes]` section of the EDK INF file is a list of directories to be included on the compile command line. These are included in a section of the Makefile generated by the parsing utilities. For each include path specified, the following line is written to the component's makefile.

`INC = $(INC) -I $(SOURCE_DIR)\{path}`

The path must be absolute, however the use of the global variable, EDK_SOURCE is recommended to construct the path.

This section will typically use one of the following section definitions:

```
[includes.common]
[includes.IA32]
[includes.X64]
[includes.IPF]
[includes.Nt32]
[include.common]
[include.IA32]
[include.X64]
[include.IPF]
```

The formats for entries in this section is:

`$(EDK_SOURCE)/path/to/header/files`

The following is an example of the [includes] section.

```
[includes.common]
  $(EDK_SOURCE)FoundationEfi
  $(EDK_SOURCE)Foundation
  $(EDK_SOURCE)FoundationFramework
  .
  $(EDK_SOURCE)FoundationInclude
  $(EDK_SOURCE)FoundationEfiInclude
  $(EDK_SOURCE)FoundationFrameworkInclude
  $(EDK_SOURCE)FoundationIncludeIndustryStandard
  $(EDK_SOURCE)FoundationCoreDxe
  $(EDK_SOURCE)FoundationLibraryDxeInclude
```

A.1.5 [nmake] Section

The optional EDK `[nmake]` section may also include a ".ProcessorName" to restrict processing based on the processor name. The section data is simply copied directly to the component makefile, before the build commands are emitted.

This section will typically use one of the following section definitions:

```
[nmake]
[nmake.common]
[nmake.IA32]
[nmake.X64]
[nmake.IPF]
[nmake.EBC]
```

The format for entries in this section is any valid Makefile syntax. Refer to make command reference for your tool chains.

The following is an example of the EDK `[nmake]` section.

```
[nmake.common]
  IMAGE_ENTRY_POINT = DiskIoDriverEntryPoint
```

A.2 EDK File Specification

The general rules for all EDK INI style documents follow.

Note: Path and Filename elements within the INF are case-sensitive in order to support building on UNIX style operating systems.

A section terminates with either another section definition or the end of the file.

Summary

Component EDK INF description

Prototype

```
<EDK_INF> ::= [<Header>]
              <Defines>
              <Sources>
              [<Includes>]
              [<Libraries>]
              [<Nmake>]
```

A.2.1 Header Section

Summary

This section contains Copyright and License notices for the INF file in comments that start the file. This section is optional using a format of:

```
#/*++
#
# Copyright
# License
#
# Module Name:
# EdkFrameworkProtocolLib.inf
#
# Abstract:
#
# Component description file.
#
#--*/
```

This information a developer creating a new EDK component or library information (INF) file.

This is an optional section.

Prototype

```
<Header>     ::= ["#"] "/*++" <EOL>
                 [<Copyright>]
                 [<License>]
                 [<ModuleName>]
                 [<Abstract>]
                 ["#"] "--*/" <EOL>
<Abstract>   ::= ["#"] "Abstract:" <EOL>
                 [["#"] <Sentence> <EOL>]*
                 ["#"] <EOL>
<ModuleName> ::= ["#"] "Module Name:" <EOL>
                 [["#"] <Sentence>+ <EOL>]+
                 ["#"] <EOL>
<Copyright>  ::= [["#"] "Copyright (c) <Date> "," <CompExtra> <EOL>]+ ["#"]
                 <EOL>
<License>    ::= [["#"] <LicenseSentence> <EOL>]+
                 ["#"] <EOL>
```

Example

```
#/*++
#
# Copyright (c) 2004, Intel Corporation
# All rights reserved. This program and the accompanying materials
# are licensed and made available under the terms and conditions of the
# BSD License which accompanies this distribution. The full text of the
# license may be found at
# http://opensource.org/licenses/bsd-license.php
#
# THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
# WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR
# IMPLIED.
#
# Module Name:
#
# EdkFrameworkProtocolLib.inf
#
# Abstract:
#
# Component description file.
#
#--*/
```

A.2.2 [defines] Section

Summary

This describes the required `[define]` tag, which is required in all EDK INF files. This file is created by the developer and is an input to the new build tool parsing utilities. Elements may appear in any order within this section.

This is a required section.

The define sections defines symbols that describe the component. Some items are emitted to the output makefile.

The `FILE_GUID` is required for all EDK components that are not libraries. This guid is used to build the FW volume file list used by build tools to generate the final firmware volume, as well as processed in some SMM, PEI or DXE DEPEX statements.

Note: Possible values for `COMPONENT_TYPE`, and their descriptions, are listed in the table, "Component (module) Types." For each component, the `BASE_NAME` and `COMPONENT_TYPE` are required. The `COMPONENT_TYPE` definition is case sensitive. The default FV extension can be overridden by defining the symbol `FV_EXT`.

Section `[defines.$(PROCESSOR).$(PLATFORM)]` is used with EDK components only. The section is processed in order by the parsing utilities. Assignments of variables in other sections do not override previous assignments.

Platform integrators that will be using EDK components that require a static FlashMap.h (and/or FlashMap.inc) must code them by hand and maintain the state of the static FlashMap files with the EDK II DSC and FDF files.

Prototype

```
<Defines>      ::= "[defines" [<attribs>] "]" <EOL> <expression>+
<attribs>      ::= <attrs> ["," "defines" <attrs>]*
<attrs>        ::= "." <arch> ["." <PlatformName>]
<arch>         ::= {"IA32"} {"X64"} {"IPF"} {"EBC"} {"common"}
<PlatformName> ::= {<Word>} {"$(PLATFORM)"} {"platform"}
<expression>   ::= "BASE_NAME" "=" <Word> <EOL>
                   ["COMPONENT_TYPE" "=" <EdkCompType> <EOL>]
                   ["FILE_GUID" "=" <GuidOrVar> <EOL>]
                   ["EDK_RELEASE_VERSION" "=" "0x00020000" <EOL>]
                   ["MAKEFILE_NAME" "=" <Filename> <EOL>]
                   ["CUSTOM_MAKEFILE" "=" <Filename> <EOL>]
                   ["BUILD_NUMBER" "=" <Integer>{1,4} <EOL>]
                   ["BUILD_TYPE" "=" <MakefileType> <EOL>]
                   ["FFS_EXT" "=" <Word> <EOL>]
                   ["FV_EXT" "=" <Word> <EOL>]
                   ["SOURCE_FV" "=" <Word> <EOL>]
                   ["PACKAGE" "=" "CompressPEIM" <EOL>]
                   ["VERSION_NUMBER" "=" <Integer>{1,4} <EOL>]
                   ["VERSION_STRING" "=" <String> <EOL>]
                   ["GENERIC_CAPSULE_FILE_PATH" "=" <PathOnly> <EOL>]
                   ["MICROCODE_ALIGNMENT" "=" <HexNumber> <EOL>]
                   ["MICROCODE_FILE_PATH" "=" <PathOnly> <EOL>]
                   ["PLATFORM_BDS_FILE_PATH" "=" <PathOnly> <EOL>]
                   ["RESTRICTED_BDS_FILE_PATH" "=" <PathOnly> <EOL>]
<Filename>     ::= [<PATH>] <Word> ["." <Extension>]
<PathOnly>     ::= <PATH> <Word>
<MakefileType> ::= {"MAKEFILE"} {"CUSTOM_MAKEFILE"} {<Filename>}
<PATH>         ::= [<Variable> "\"] <Path>
<Path>         ::= [{<Word> "\"} {"..\"}]+
<Variable>     ::= {"$(" <MacroName> ")"}
                   {"$(EFI_SOURCE)"} {"$(EDK_SOURCE)"}
<GuidOrVar>    ::= {<RegistryFormatGUID>}
                   {"$(EFI_APRIORI_GUID)"}
                   {"$(EFI_ACPI_TABLE_STORAGE_GUID)"}
                   {"$(EFI_DEFAULT_BMP_LOGO_GUID)"}
                   {"$(EFI_PEI_APRIORI_FILE_NAME_GUID)"}
<EdkCompType>  ::= {"APPLICATION"} {"AcpiTable"} {"APRIORI"}
                   {"BINARY"} {"BS_DRIVER"} {"CONFIG"} {"FILE"}
                   {"FVIMAGEFILE"} {"LIBRARY"} {"LOGO"} {"LEGACY16"}
                   {"MICROCODE"} {"PE32_PEIM"} {"PEI_CORE"}
                   {"RAWFILE"} {"RT_DRIVER"} {"SAL_RT_DRIVER"}
                   {"SECURITY_CORE"} {"COMBINED_PEIM_DRIVER"} {"PIC_PEIM"}
                   {"RELOCATABLE_PEIM"}
<MacroName>    ::= <Word>
```

Example (EDK Driver)

```
[Defines]
  BASE_NAME      = DiskIo
  FILE_GUID      = CA261A26-7718-4b9b-8A07-5178B1AE3A02
  COMPONENT_TYPE = BS_DRIVER
```

Example (EDK Library)

```
[Defines]
  BASE_NAME      = WinNtLib
  COMPONENT_TYPE = LIBRARY
```

A.2.3 [includes] Section

Summary

Defines the optional "includes paths" for EDK INF files only. These sections should never be used in EDK II INF files. These sections are used to define the include paths for compiling the component source files. Valid sections for EDK include the `[includes.$(PROCESSOR).$(PLATFORM)]`, `[includes.$(PROCESSOR)]`, and `[includes.common]` sections.

NOTE: EDK uses both "include" and "includes" section header types. These sections are processed if present. These paths are used to define the `$(INC)` macro and is written to the component's makefile.

This is an optional section.

The standard Macro Definitions are not permitted within this section.

For EDK modules, the path must include either the `$(EFI_SOURCE)` or `$(EDK_SOURCE)` environment variable.

This section also allows for specifying individual header files that will be added to the `$(INC)` macro using the `/FI` (Microsoft) or `-include` (GCC) switch. This is an optional section.

Prototype

```
<Includes>   ::= "[include" ["s"] [<Attrs>] ']'" <EOL>
                 <PATH>+
<Attrs>      ::= <Attributes> ["," "include" ["s"]? <Attrs>]*
<Attributes> ::= [<Archs>] [<VarName>] [<Platform>]
<Archs>      ::= "." <arch>
<arch>       ::= {"IA32"} {"X64"} {"IPF"} {"EBC"} {"common"} {<OA>}
<VarName>    ::= "." {"$(PROCESSOR)"} {"$(" <Word> ")"}
<Platform>   ::= "." {"Platform") {"nt32"} {"$(PLATFORM)"}
<PATH>       ::= {<MACRO>} {<RelDir>}
<MACRO>      ::= <MacroName> ["\" <Word>]+
<MacroName>  ::= {"$(EDK_SOURCE)"} {"$(EFI_SOURCE)"} {"$(BUILD_DIR)"}
                 {"$(SOURCE_DIR)"}
<RelDir>     ::= ["..\"]+ {".."} {<Word>}
```

Example

```
[Includes.common]
  $(EDK_SOURCE)FoundationEfi
  $(EDK_SOURCE)Foundation
  $(EDK_SOURCE)FoundationFramework
  .
  $(EDK_SOURCE)FoundationInclude
  $(EDK_SOURCE)FoundationEfiInclude
  $(EDK_SOURCE)FoundationFrameworkInclude
  $(EDK_SOURCE)FoundationIncludeIndustryStandard
  $(EDK_SOURCE)FoundationCoreDxe
  $(EDK_SOURCE)FoundationLibraryDxeInclude
```

A.2.4 [libraries] Section

Summary

Defines the optional `[libraries]` section tag for EDK INF files. The `[libraries]` section is used to define a `$(LIBS)` macro in the EDK component's makefile. All libraries listed in the `[libraries.common]`, `[libraries.$(PROCESSOR)]`, and `[libraries.$(PROCESSOR).$(PLATFORM)]` sections are added to the LIBS definition as either `$(LIB_DIR)\LibName` or `$(PROCESSOR)\LibName`. The libraries are specified without path information.

The standard Macro Definitions are not permitted in this section.

This is an optional section.

Prototype

```
<Libraries> ::= "[libraries" [<attrs>] ']" <EOL> <LibName>+
<attrs>     ::= "." <archs> ["." <platform>]
<archs>     ::= <arch> ["," <arch> ["." <platform>]]
                ["," "libraries." <attrs>]*
<arch>      ::= {"IA32"} {"X64"} {"IPF"} {"EBC"} {"common"}
                {"platform"} {"nt32"} {"$(PROCESSOR)"}
<platform>  ::= {"platform"} {"$(PLATFORM)"} {<Word>}
<LibName>   ::= <Word>
```

Example

```
[Libraries.common]
  EfiProtocolLib
  EfiDriverLib
```

A.2.5 [nmake] Section

Summary

Defines the [nmake] section tag for EDK INF files. These sections are used to make a direct addition to the component's output makefile. EFI section `[nmake.$(PROCESSOR).$(PLATFORM)]`, `[nmake.$(PROCESSOR)]`, and `[nmake.common]` are processed if present. The section data is simply copied directly to the component makefile, before the build commands are emitted. Filenames specified in this section are relative to the EDK INF file or the EDK DSC file.

This is an optional section.

This section is not permitted in EDK II modules. Note that the `C_STD_INCLUDE` line is usually used to clear any flags that might have been set by the Microsoft tool chain setup scripts, therefore no `<CFlags>` are printed, and the line in the Example is the most common usage.

The standard Macro Definitions are not permitted in this section.

Prototype

```
<Nmake>      ::= "[nmake" [<attrs>] ']" <EOL>
                 <expression>+
<attrs>      ::= <Archs> [<plat>] ["," "nmake" <attrs>]*
<Archs>      ::= "." <arch>
<plat>       ::= "." <Word>
<arch>       ::= {"IA32"} {"X64"} {"IPF"} {"EBC"} {"common"}
<expression> ::= [<IEP>] [<DS>] [<CSI>] [<CPF>] [<APF>] [<LSF>] [<EBC>] [<DEF>]
<IEP>        ::= "IMAGE_ENTRY_POINT" "=" <CName> <EOL>
<DS>         ::= "DPX_SOURCE" "=" <Filename> <EOL>
<CSI>        ::= "C_STD_INCLUDE" "=" [<CFlags>] <EOL>
<CPF>        ::= "C_PROJ_FLAGS" "=" <CFlags> <EOL>
<APF>        ::= "ASM_PROJ_FLAGS" "=" <AFlags> <EOL>
```

```
<LSF>        ::= "LIB_STD_FLAGS" "=" <LFlags> <EOL>
<EBC>        ::= ["EBC_C_STD_FLAGS" "=" <String> <EOL>]
                 ["EBC_LIB_STD_FLAGS" "=" <String> <EOL>]
<DEF>        ::= <Word> "=" {<Word>} {<String>} <EOL>
<STATEMENTS> ::= Valid NMAKE Makefile syntax.
```

Parameters

CFlags

This content must be valid compiler flags for the Microsoft C compiler, cl.exe.

AFlags

This content must be valid flags for the Microsoft Assembler, ml.exe.

LFlags

This content must be valid flags for the Microsoft Linker, lib.exe.

Example

```
[nmake.common]
  C_STD_INCLUDE =
  IMAGE_ENTRY_POINT=WatchdogTimerDriverInitialize
  DPX_SOURCE=WatchDogTimer.dxs

[nmake.common]
  C_FLAGS       = $(C_FLAGS) /D EDKII_GLUE_LIBRARY_IMPLEMENTATION
  LIB_STD_FLAGS = $(LIB_STD_FLAGS) /IGNORE:4006 /IGNORE:4221

[nmake.ia32]
  C_FLAGS = $(C_FLAGS) /D MDE_CPU_IA32

[nmake.x64]
  C_FLAGS = $(C_FLAGS) /D MDE_CPU_X64

[nmake.ipf]
  C_FLAGS = $(C_FLAGS) /D MDE_CPU_IPF

[nmake.ebc]
  EBC_C_STD_FLAGS   = $(EBC_C_STD_FLAGS) /D EDKII_GLUE_LIBRARY_IMPLEMENTATION
  EBC_LIB_STD_FLAGS = $(EBC_LIB_STD_FLAGS) /IGNORE:4006 /IGNORE:4221
  EBC_C_STD_FLAGS   = $(EBC_C_STD_FLAGS) /D MDE_CPU_EBC
```

A.2.6 [sources] Section

Summary

Defines the `[sources]` section tag is required for EDK INF files. NOTE: EDK uses both "source" and "sources" in the section header.

There can be multiple sources sections, depending on the target processor. Example sources sections are listed below. The parsing utility creates a directory path for each file (`$(DEST_DIR)...\MyFile.c`), and looks up the makefile template for the `COMPONENT_TYPE` (EDK) to emit.

It is not permissible to mix EDK and EDK II style files within a module.

The macro, TABLE_NAME may be used in existing EDK INF files that point to ACPI tables, this value wil be ignored by EDK II build tools.

Prototype

```
<sources>        ::= "[source" ["s"] [<attrs>] "]" <EOL>
                     [<MacroDefinition>]* [<EdkExpression>]+
<attrs>          ::= <Archs> ["," "sources" <attrs>]*
<Archs>          ::= if (COMPONENT_TYPE defined in defines):
                     "." <archs> [<plat>] else:
                     "." <archs>
<archs>          ::= if (COMPONENT_TYPE defined in defines): <EdkArch> [","
                     <EdkArch>]* else:
                     <arch> ["|" <arch>]*
<EdkArch>        ::= {"IA32"} {"X64"} {"IPF"} {"Common"}
<plat>           ::= "." {"$(PLATFORM)"} {"$(PROCESSOR)"} {<Word>}
<DefineStatement> ::= ["DEFINE" <Word> "=" [<PATH>] <EOL>]*
                     ["TABLE_NAME" "=" <Word> <EOL>]
<EdkExpression>  ::= <Filename> ["|" <Family>] <EOL>
<Family>         ::= {"MSFT"} {"GCC"} {"INTEL"} {"*"}
<Filename>       ::= <Path> <Word> "." <Extension>
<Path>           ::= [<Macro> {"\"} {"/"}] [<Word> {"\"} {"/"}]+
```

Examples

```
[sources.common]
  BsDataHubStatusCode.c
  BsDataHubStatusCode.h
```

APPENDIX B BUILD CHANGES AND CUSTOMIZATIONS

B.1 Customizing EDK Compilation for a Component

There are several mechanisms for customizing the build for a firmware component. These include:

- Creating a new component INF file that specifies `BUILD_TYPE=xxx`, and then creating a `[build.$(PROCESSOR).xxx]` section in the platform DSC file.
- Creating a new component INF file and a makefile for the component, and specifying `BUILD_TYPE=MAKEFILE` in the INF file. Then add a `[build.$(PROCESSOR).makefile]` section to the DSC file that describes how to "build" the component using the makefile. Typically this will be commands to copy the file to the `$(DEST_DIR)`, and then invoking nmake.
- Trivial customizations can be accomplished by adding or modifying the `[nmake]` sections in the component INF file. This may require defining `$(PLATFORM)` in the EDK DSC file, and then adding a new `[nmake.$(PROCESSOR).$(PLATFORM)]` section in the component INF file.
- Another option is to define a variable in the component INF file, passing it to the component makefile via the DSC `[makefile.common]` section, and then using `!IFDEF` statements in the `[build.$(PROCESSOR).$(COMPONENT_TYPE)]` section to perform custom steps.

B.2 Changing Files in an EDK Library

Library INF files are shared among different platforms. However, not all platforms require all the same source files. To customize the library INF files for different platforms, simply define `$(PLATFORM)`, either on the command line, or in the DSC file, and then make customizations in the `[sources.$(PROCESSOR).$(PLATFORM)]` section of the library INF file.

An alternative to this method is to simply create a new INF file for the library, and then use it in place of the existing library INF file

B.3 Customizing EDK II Compilation for a Module Common Definitions

The preferred method for customizing a build is to copy the source module directory to a new directory and modifying the INF file and module sources. This method is preferred over the EDK methods as build reproducibility is more easily accomplished.

Additional customizations for build options should be made in the platform description (DSC) file. While it is permitted to use the `[BuildOptions]` section to define custom compiler flags, this section should only be used as a last resort. The default flags defined the `tools_def.txt` file provide the best known size and speed optimizations, and the platform DSC file can override the defaults in its `[BuildOptions]` section.

APPENDIX C SYMBOLS

One of the core concepts of this utility is the notion of symbols. Use of symbols follows the makefile convention of enclosing within `$()`, for example `$(EDK_SOURCE)`. As the utility processes files during execution, it will often perform parsing of variable assignments. These variables can then be referenced in other sections of the DSC file. Variable assignments will be saved internally in either a local or global symbol table. The local symbol table is purged following processing of individual component INF files. Global symbol values persist throughout execution of the utility. Local symbol values take precedent over global symbols. The following table describes the symbols generated internally by the utility. They can be overridden either on the command line, in the DSC file, or in individual INF files. The G/L column indicates whether the symbol is typically a global or a local symbol.

Table 8 Symbol Description

Symbol Name	G/L	Description
`EDK_SOURCE`	G	Defines the root directory of the local EDK source tree, for example C:\EFI2.0 If not defined as an environmental variable when the tool is invoked, the utility will attempt to determine a reasonable value based on the current working directory.
`EFI_SOURCE`	G	Defines the root directory of the local EDK source tree, for example C:\EFI2.0 If not defined as an environmental variable when the tool is invoked, the utility will attempt to determine a reasonable value based on the current working directory.
`PROCESSOR`	G/L	Defines the target processor for which the code is to be built. This symbol will typically be used to include or exclude source files in component INF files, and to define the tool chain for building.
`BUILD_DIR`	G	Defines the build tip directory for the current platform. For example, this may be `$(EDK_SOURCE)\Platform\Anacortes_870`.
`SOURCE_DIR`	L	For a component, defines the directory of the component source files.
`DEST_DIR`	L	For a component, defines the directory (typically under BUILD_DIR) where the component object files are to be built.
`LIB_DIR`	L	Specifies the directory where EFI libraries are deposited after building. Typically `$(BUILD_DIR)\$(PROCESSOR)`
`BIN_DIR`	L	Specifies the directory where final component binaries are deposited during build. Typically `$(BUILD_DIR)\$(PROCESSOR)`
`OUT_DIR`	L	Unused, but typically `$(BUILD_DIR)\$(PROCESSOR)`
`DSC_FILENAME`	G	Name of the DSC file as specified on the command line. Can be used for dependencies in the makefiles.
`INF_FILENAME`	L	Name of the INF file for a given component. Can be used for dependencies in the makefiles.
`FV_EXT`	L	Common component type (BS driver, application, etc) have predefined file name extensions assigned (.dxe, .app, etc). If there is a deviation from the convention, or a new
		(unknown to the utility) component type is being built, then `FV_EXT` may need to be defined for the component so the utility knows the result file name extension. This information is necessary to generate dependencies in makefile.out.
`MAKEFILE_NAME`	L	Name of the output makefile for the component. Default is "makefile". This value can be overridden to support building different variations of a component in the same `DEST_DIR` directory.

`PLATFORM`	L	This symbol can be used to provide more selectivity of files in the component INF files. If assigned, then the utility will also process any files in the INF file under sections `[sources.$(PROCESSOR).$(PLATFORM)]` , `[includes.$(PROCESSOR).$(PLATFORM)]` , and `[libraries.$(PROCESSOR).$(PLATFORM)]` .
`FILE`	L	As the utility processes each source file in the component INF file, this symbol gets assigned the name of the file, less the file extension.
`PACKAGE`	L/G	If defined, then the utility will create a package file named `$(DEST_DIR)\$(BASE_NAME).pkg` , and copy, with macro expansion, the `[package.$(COMPONENT_TYPE).$(PACKAGE)]` section from the DSC file to the output file.
`PACKAGE_FILE`	L	If defined, then the utility will not generate a package file. The build can then use the value `$(PACKAGE_FILE)` to have GenFfsFile use an existing package file for creating the firmware file.

APPENDIX D SAMPLE DRIVER INF FILES

The following INF file example are from EDK II `MdeModulePkg/Universal/Disk/DiskIoDxe/DiskIoDxe.inf` and `IntelFrameworkModulePkg/Universal/StatusCode/RuntimeDxe/StatusCodeRuntimeDxe.inf` driver modules.

D.1 DiskIoDxe INF file

```
## @file
# Module that lays Disk I/O protocol on every Block I/O protocol.
#
# This module produces Disk I/O protocol to abstract the block accesses
# of the Block I/O protocol to a more general offset-length protocol
# to provide byte-oriented access to block media. It adds this protocol
# to any Block I/O interface that appears in the system that does not
# already have a Disk I/O protocol. File systems and other disk access
# code utilize the Disk I/O protocol.
#
# Copyright (c) 2006 - 2012, Intel Corporation. All rights reserved.<BR>
# This program and the accompanying materials
# are licensed and made available under the terms and conditions of the
# BSD License which accompanies this distribution. The full text of the
# license may be found at:
# http://opensource.org/licenses/bsd-license.php
#
# THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
# WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR
# IMPLIED.
#
##

[Defines]
  INF_VERSION     = 0x0001001B
  BASE_NAME       = DiskIoDxe
  MODULE_UNI_FILE = DiskIoDxe.uni
  FILE_GUID       = 6B38F7B4-AD98-40e9-9093-ACA2B5A253C4
  MODULE_TYPE     = UEFI_DRIVER
  VERSION_STRING  = 1.0
  ENTRY_POINT     = InitializeDiskIo

#
# The following information is for reference only and not required by the build tools.
#
# VALID_ARCHITECTURES = IA32 X64 IPF EBC
#
# DRIVER_BINDING = gDiskIoDriverBinding
# COMPONENT_NAME = gDiskIoComponentName
# COMPONENT_NAME2 = gDiskIoComponentName2
#

[Sources]
  ComponentName.c
  DiskIo.h
  DiskIo.c

[Packages]
  MdePkg/MdePkg.dec

[LibraryClasses]
  UefiBootServicesTableLib
  MemoryAllocationLib
  BaseMemoryLib
  BaseLib
  UefiLib
  UefiDriverEntryPoint
  DebugLib
```

```
[Protocols]
  gEfiDiskIoProtocolGuid   ## BY_START
  gEfiBlockIoProtocolGuid  ## TO_START
```

D.2 StatusCodeRuntimeDxe INF file

```
## @file
# Status Code Runtime Dxe driver produces Status Code Runtime Protocol.
#
# Copyright (c) 2006 - 2012, Intel Corporation. All rights reserved.<BR>
#
# This program and the accompanying materials
# are licensed and made available under the terms and conditions of the
# BSD License which accompanies this distribution. The full text of the
# license may be found at:
# http://opensource.org/licenses/bsd-license.php
# THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
# WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR
# IMPLIED.
#
##

[Defines]
  INF_VERSION     = 0x0001001B
  BASE_NAME       = StatusCodeRuntimeDxe
  MODULE_UNI_FILE = StatusCodeRuntimeDxe.uni
  FILE_GUID       = FEDE0A1B-BCA2-4A9F-BB2B-D9FD7DEC2E9F
  MODULE_TYPE     = DXE_RUNTIME_DRIVER
  VERSION_STRING  = 1.0
  ENTRY_POINT     = StatusCodeRuntimeDxeEntry

#
# The following information is for reference only and not required by the
# build tools.
#
# VALID_ARCHITECTURES = IA32 X64 EBC
#
# VIRTUAL_ADDRESS_MAP_CALLBACK = VirtualAddressChangeCallBack
#

[Sources]
  SerialStatusCodeWorker.c
  RtMemoryStatusCodeWorker.c
  DataHubStatusCodeWorker.c
  StatusCodeRuntimeDxe.h
  StatusCodeRuntimeDxe.c

[Packages]
  MdePkg/MdePkg.dec
  MdeModulePkg/MdeModulePkg.dec
  IntelFrameworkPkg/IntelFrameworkPkg.dec
  IntelFrameworkModulePkg/IntelFrameworkModulePkg.dec

[LibraryClasses]
  OemHookStatusCodeLib
  SerialPortLib
  UefiRuntimeLib
  MemoryAllocationLib
  UefiLib
  UefiBootServicesTableLib
  UefiDriverEntryPoint
  HobLib
  PcdLib
  PrintLib
  ReportStatusCodeLib
  DebugLib
  BaseMemoryLib
  BaseLib
```

```
[Guids]
  gEfiDataHubStatusCodeRecordGuid    ## SOMETIMES_PRODUCES  ## UNDEFINED  # DataRecord Guid
  gEfiStatusCodeDataTypeDebugGuid    ## SOMETIMES_PRODUCES  ## UNDEFINED  # Record data type
  gMemoryStatusCodeRecordGuid        ## SOMETIMES_CONSUMES  ## HOB
  gEfiEventVirtualAddressChangeGuid  ## CONSUMES            ## Event
  gEfiStatusCodeDataTypeStringGuid   ## SOMETIMES_CONSUMES  ## UNDEFINED

[Protocols]
  gEfiStatusCodeRuntimeProtocolGuid  ## PRODUCES
  gEfiDataHubProtocolGuid            ## SOMETIMES_CONSUMES  # Needed if Data Hub is supported for status code

[FeaturePcd]
  gEfiMdeModulePkgTokenSpaceGuid.PcdStatusCodeReplayIn               ## CONSUMES
  gEfiIntelFrameworkModulePkgTokenSpaceGuid.PcdStatusCodeUseOEM      ## CONSUMES
  gEfiIntelFrameworkModulePkgTokenSpaceGuid.PcdStatusCodeUseDataHub  ## CONSUMES
  gEfiMdeModulePkgTokenSpaceGuid.PcdStatusCodeUseMemory              ## CONSUMES
  gEfiMdeModulePkgTokenSpaceGuid.PcdStatusCodeUseSerial              ## CONSUMES

[Pcd]
  gEfiMdeModulePkgTokenSpaceGuid.PcdStatusCodeMemorySize|128|gEfiMdeModulePkgTokenSpaceGuid.PcdStatusCodeUseMemory  ## SOMETIM
ES_CONSUMES

[Depex]
  TRUE
```

Note: In the above example, the backslash "\" character is used to show a line continuation for readability. Use of a backslash character in the actual INF file is not permitted.

APPENDIX E SAMPLE LIBRARY INF FILES

The following INF file are examples of INF files for the EDK II MdePkg library, PeiServicesTablePointerLib and the MdeModulePkg libraries, DxeCoreMemoryAllocationLib.inf and SmmCorePerformanceLib.inf.

E.1 PeiServicesTablePointerLib.inf

```
## @file
# Instance of PEI Services Table Pointer Library using global variable for the table pointer.
#
# PEI Services Table Pointer Library implementation that retrieves a
# pointer to the PEI Services Table from a global variable. Not available
# to modules that execute from read-only memory.
#
# Copyright (c) 2007 - 2012, Intel Corporation. All rights reserved.<BR>
#
# This program and the accompanying materials are licensed and made
# available under the terms and conditions of the BSD License which
# accompanies this distribution. The full text of the license may be
# found at:
# http://opensource.org/licenses/bsd-license.php.
# THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
# WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR
# IMPLIED.
#
##

[Defines]
  INF_VERSION     = 0x0001001B
  BASE_NAME       = PeiServicesTablePointerLib
  MODULE_UNI_FILE = PeiServicesTablePointerLib.uni
  FILE_GUID       = 1c747f6b-0a58-49ae-8ea3-0327a4fa10e3
  MODULE_TYPE     = PEIM
  VERSION_STRING  = 1.0
  LIBRARY_CLASS   = PeiServicesTablePointerLib|PEIM PEI_CORE SEC
  CONSTRUCTOR     = PeiServicesTablePointerLibConstructor

#
# VALID_ARCHITECTURES = IA32 X64 IPF EBC (EBC is for build only)
#

[Sources]
  PeiServicesTablePointer.c

[Packages]
  MdePkg/MdePkg.dec

[LibraryClasses]
  DebugLib
```

E.2 DxeCoreMemoryAllocationLib.inf

```
## @file
# Memory Allocation Library instance dedicated to DXE Core.
#
# The implementation borrows the DxeCore Memory Allocation services as
# the primitive for memory allocation instead of using UEFI boot
# services in an indirect way.
# It is assumed that this library instance must be linked with DxeCore
# in this package.
#
# Copyright (c) 2008 - 2010, Intel Corporation. All rights reserved.<BR>
```

```
#
# This program and the accompanying materials are licensed and made
# available under the terms and conditions of the BSD License which
# accompanies this distribution. The full text of the license may be
# found at:
# http://opensource.org/licenses/bsd-license.php
#
# THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
# WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR
# IMPLIED.
#
##

[Defines]
  INF_VERSION    = 0x0001001B
  BASE_NAME      = DxeCoreMemoryAllocationLib
  FILE_GUID      = 632F3FAC-1CA4-4725-BAA2-BDECCF9A111C
  MODULE_TYPE    = DXE_CORE
  VERSION_STRING = 1.0
  LIBRARY_CLASS  = MemoryAllocationLib|DXE_CORE

#
# The following information is for reference only and not required by the
# build tools.
#
# VALID_ARCHITECTURES = IA32 X64 IPF EBC
#

[Sources]
  MemoryAllocationLib.c
  DxeCoreMemoryAllocationServices.h

[Packages]
  MdePkg/MdePkg.dec

[LibraryClasses]
  DebugLib
  BaseMemoryLib
```

E.3 SmmCorePerformanceLib.inf

```
## @file
# Performance library instance used by SMM Core.
#
# This library provides the performance measurement interfaces and
# initializes performance logging for the SMM phase.
# It initializes SMM phase performance logging by publishing the SMM
# Performance and PerformanceEx Protocol, which is consumed by
# SmmPerformanceLib to logging performance data in SMM phase.
# This library is mainly used by SMM Core to start performance logging
# to ensure that SMM Performance and PerformanceEx Protocol are
# installed at the very beginning of SMM phase.
#
# Copyright (c) 2011 - 2012, Intel Corporation. All rights reserved.<BR>
#
# This program and the accompanying materials are licensed and made
# available under the terms and conditions of the BSD License which
# accompanies this distribution.
# The full text of the license may be found at:
# http://opensource.org/licenses/bsd-license.php
#
# THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
# WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR
# IMPLIED.
#
##

[Defines]
  INF_VERSION              = 0x0001001B
  BASE_NAME                   = SmmCorePerformanceLib
  FILE_GUID                   = 36290D10-0F47-42c1-BBCE-E191C7928DCF
  MODULE_TYPE                 = SMM_CORE
  VERSION_STRING              = 1.0
  PI_SPECIFICATION_VERSION = 0x0001000A
  LIBRARY_CLASS               = PerformanceLib|SMM_CORE
  CONSTRUCTOR                 = SmmCorePerformanceLibConstructor

#
# The following information is for reference only and not required by the
# build tools.
#
# VALID_ARCHITECTURES = IA32 X64
#

[Sources]
  SmmCorePerformanceLib.c
  SmmCorePerformanceLibInternal.h

[Packages]
  MdePkg/MdePkg.dec
  MdeModulePkg/MdeModulePkg.dec

[LibraryClasses]
  MemoryAllocationLib
  UefiBootServicesTableLib
  PcdLib
  TimerLib
  BaseMemoryLib
  BaseLib
  DebugLib
  SynchronizationLib
  SmmServicesTableLib

[Protocols]
  gEfiSmmBase2ProtocolGuid    ## CONSUMES
  gEfiSmmAccess2ProtocolGuid  ## CONSUMES

[Guids]
  ## PRODUCES ## UNDEFINED # Install protocol
  ## CONSUMES ## UNDEFINED # SmiHandlerRegister
 gSmmPerformanceProtocolGuid
  ## PRODUCES ## UNDEFINED # Install protocol
  ## CONSUMES ## UNDEFINED # SmiHandlerRegister
  gSmmPerformanceExProtocolGuid

[Pcd]
  gEfiMdePkgTokenSpaceGuid.PcdPerformanceLibraryPropertyMask## CONSUMES
```

Note: In the above example, the backslash "\" character is used to show a line continuation for readability. Use of a backslash character in the actual INF file is not permitted.

APPENDIX F SAMPLE BINARY INF FILES

The following are example INF files for the binary modules, EnhancedFatDxe, in the FatBinPkg. The second example is a generated binary INF file for the RuntimeDxe driver in the MdeModulePkg.

F.1 FatBinPkg/EnhancedFatDxe/Fat.inf

```
## @file
#
# Binary FAT32 EFI Driver for IA32, X64, IPF and EBC arch.
#
# This UEFI driver detects the FAT file system in the disk.
# It also produces the Simple File System protocol for the consumer to
# perform file and directory operations on the disk.
#
# Copyright (c) 2007 - 2010, Intel Corporation. All rights reserved.<BR>
#
# This program and the accompanying materials are licensed and made
# available under the terms and conditions of the BSD License which
# accompanies this distribution. The full text of the license may be
# found at:
# http://opensource.org/licenses/bsd-license.php
# THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
# WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR
# IMPLIED.
#
##

[Defines]
  INF_VERSION    = 0x00010009
  BASE_NAME      = Fat
  FILE_GUID      = 961578FE-B6B7-44c3-AF35-6BC705CD2B1F
  MODULE_TYPE    = UEFI_DRIVER
  VERSION_STRING = 1.0

#
# The following information is for reference only and not required by the
# build tools.
#
# VALID_ARCHITECTURES = IA32 X64 IPF EBC
#

[Binaries.Ia32]
  PE32|Ia32/Fat.efi|*

[Binaries.X64]
  PE32|X64/Fat.efi|*

[Binaries.IPF]
  PE32|Ipf/Fat.efi|*

[Binaries.EBC]
  PE32|Ebc/Fat.efi|*

[Binaries.ARM]
  PE32|Arm/Fat.efi|*
```

F.2 MdeModulePkg/Core/RuntimeDxe.inf

```
## @file
# Module that produces EFI runtime virtual switch over services.
#
```

```
# This runtime module installs Runtime Architectural Protocol and
# registers CalculateCrc32 boot services table, SetVirtualAddressMap &
# ConvertPointer runtime services table.
#
# Copyright (c) 2006 - 2012, Intel Corporation. All rights reserved.<BR>
#
# This program and the accompanying materials are licensed and made
# available under the terms and conditions of the BSD License which
# accompanies this distribution. The full text of the license may be
# found at:
# http://opensource.org/licenses/bsd-license.php
# THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
# WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR
# IMPLIED.
#
##

[Defines]
  INF_VERSION    = 0x0001001B
  BASE_NAME      = RuntimeDxe
  FILE_GUID      = B601F8C4-43B7-4784-95B1-F4226CB40CEE
  MODULE_TYPE    = DXE_RUNTIME_DRIVER
  VERSION_STRING = 1.0

[Packages.IA32]
  MdePkg/MdePkg.dec
  MdeModulePkg/MdeModulePkg.dec

[Binaries.IA32]
  PE32|RuntimeDxe.efi
  DXE_DEPEX|RuntimeDxe.depex

[PatchPcd.IA32]
  ## CONSUMES
  gEfiMdePkgTokenSpaceGuid.PcdDebugPrintErrorLevel|0x80000047|0x1EC8

  ## CONSUMES
  gEfiMdePkgTokenSpaceGuid.PcdDebugPropertyMask|0x27|0x1ECC

  ## CONSUMES
  gEfiMdePkgTokenSpaceGuid.PcdReportStatusCodePropertyMask|0x07|0x1ECD

[Protocols.IA32]
  ## PRODUCES
  gEfiRuntimeArchProtocolGuid

  ## SOMETIMES_CONSUMES
  ## CONSUMES
  gEfiLoadedImageProtocolGuid

  ## SOMETIMES_CONSUMES
  gPcdProtocolGuid

  ## CONSUMES
  gEfiPcdProtocolGuid

  ## SOMETIMES_CONSUMES
  gEfiDevicePathProtocolGuid

  ## CONSUMES
  gEfiStatusCodeRuntimeProtocolGuid

  ## SOMETIMES_PRODUCES
  gEfiDriverBindingProtocolGuid

  ## SOMETIMES_CONSUMES
  gEfiSimpleTextOutProtocolGuid

  ## SOMETIMES_CONSUMES
  gEfiGraphicsOutputProtocolGuid

  ## SOMETIMES_CONSUMES
  gEfiHiiFontProtocolGuid

  ## SOMETIMES_CONSUMES # Consumes if gEfiGraphicsOutputProtocolGuid uninstalled
  gEfiUgaDrawProtocolGuid

  ## SOMETIMES_PRODUCES # User chooses to produce it
  gEfiComponentNameProtocolGuid

  ## SOMETIMES_PRODUCES # User chooses to produce it
  gEfiComponentName2ProtocolGuid

  ## SOMETIMES_PRODUCES # User chooses to produce it
  gEfiDriverConfigurationProtocolGuid

  ## SOMETIMES_PRODUCES # User chooses to produce it
  gEfiDriverConfiguration2ProtocolGuid

  ## SOMETIMES_PRODUCES # User chooses to produce it
  gEfiDriverDiagnosticsProtocolGuid

  ## SOMETIMES_PRODUCES # User chooses to produce it
  gEfiDriverDiagnostics2ProtocolGuid

[Ppis.IA32]

[Guids.IA32]
  ## CONSUMES # Event
  ## CONSUMES # Event
  ## PRODUCES # Event # RuntimeDriverSetVirtualAddressMap() signals this event.
  gEfiEventVirtualAddressChangeGuid

  ## SOMETIMES_CONSUMES
  ## SOMETIMES_CONSUMES ## UNDEFINED
  gEfiStatusCodeDataTypeDebugGuid

  ## CONSUMES # Event
  ## CONSUMES # Event
  gEfiEventExitBootServicesGuid

  ## SOMETIMES_CONSUMES ## UNDEFINED
  gEfiStatusCodeSpecificDataGuid

  ## SOMETIMES_CONSUMES # Event
  gEfiEventReadyToBootGuid

  ## SOMETIMES_CONSUMES # Event
  gEfiEventLegacyBootGuid

  ## SOMETIMES_CONSUMES # Variable
  gEfiGlobalVariableGuid

[PcdEx.IA32]

#
# The following information is for reference only and not required by the
# build tools.
#
# VALID_ARCHITECTURES = IA32 X64 IPF EBC
#

## @AsBuilt
## MSFT:DEBUG_VS2008x86_IA32_SYMRENAME_FLAGS = Symbol renaming not needed for
## MSFT:DEBUG_VS2008x86_IA32_ASLDLINK_FLAGS = /NODEFAULTLIB /ENTRY:ReferenceAcpiTable /SUBSYSTEM:CONSOLE
## MSFT:DEBUG_VS2008x86_IA32_VFR_FLAGS = -l -n
## MSFT:DEBUG_VS2008x86_IA32_PP_FLAGS = /nologo /E /TC /FIAutoGen.h
## MSFT:DEBUG_VS2008x86_IA32_GENFW_FLAGS =
## MSFT:DEBUG_VS2008x86_IA32_OPTROM_FLAGS = -e
## MSFT:DEBUG_VS2008x86_IA32_SLINK_FLAGS = /NOLOGO /LTCG /Zd /Zi
## MSFT:DEBUG_VS2008x86_IA32_ASL_FLAGS =
## MSFT:DEBUG_VS2008x86_IA32_CC_FLAGS = /nologo /c /WX /GS- /W4/Gs32768 /D UNICODE /O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF /G
y /Zi /Gm
## MSFT:DEBUG_VS2008x86_IA32_VFRPP_FLAGS = /nologo /E /TC/DVFRCOMPILE /FI$(MODULE_NAME)StrDefs.h /TC /Dmain=ReferenceAcpiTable
## MSFT:DEBUG_VS2008x86_IA32_APP_FLAGS = /nologo /E /TC
## MSFT:DEBUG_VS2008x86_IA32_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB/IGNORE:4001 /OPT:REF /OPT:ICF=10 /MAP /ALIGN:32/SECTION:.xdat
a,D /SECTION:.pdata,D /MACHINE:X86 /LTCG /DLL/ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER/SAFESEH:NO /BASE:0
 /DRIVER /DEBUG/PDB:$(OUTPUT_PATH)$(BASE_NAME).pdb/PDBSTRIPPED:$(OUTPUT_PATH)$(BASE_NAME)_Stripped.pdb
## MSFT:DEBUG_VS2008x86_IA32_ASLPP_FLAGS = /nologo /E /C /FIAutoGen.h
## MSFT:DEBUG_VS2008x86_IA32_OBJCOPY_FLAGS = objcopy not needed for
## MSFT:DEBUG_VS2008x86_IA32_MAKE_FLAGS = /nologo
## MSFT:DEBUG_VS2008x86_IA32_ASMLINK_FLAGS = /nologo /tiny
```

Note: In the above example, the backslash "\" character is used to show a line continuation for readability. Use of a backslash character in the actual INF file is not permitted.

APPENDIX G MODULE TYPES

Table 9 EDK II Module Types

FILE TYPE	MODULE_TYPE	EDK II Extension	Description
Library	BASE, SEC, PEI_CORE, PEIM, DXE_CORE, DXE_DRIVER, DXE_RUNTIME_DRIVER, DXE_SAL_DRIVER, DXE_SMM_DRIVER, SMM_CORE, UEFI_DRIVER, UEFI_APPLICATION, USER_DEFINED	.lib	Library component used in the build for linking against components.
File	USER_DEFINED	.bin	In EDK, this was used to specify various types of source or binary files.
Legacy16	USER_DEFINED	.bin or .rom	In EDK, this was used to specify various types of binary files.
BINARY	USER_DEFINED	.bin or .rom	In EDK, this was used to specify various types of binary files.
CONFIG	USER_DEFINED	.bin	In EDK, this was used to sepcify INI text files that were processed by different tools into binary files.
LOGO	USER_DEFINED	.bmp	The MODULE_TYPE for a LOGO, when migrating to EDK II should be specified as USER_DEFINED. The .bmp file should be included under a [binaries] section. In EDK, the COMPONENT_TYPE of LOGO was used to specify a .bmp file in the [sources] section.
RAWFILE	USER_DEFINED	.raw	In EDK, this was used to specify various types of binary files.
FVIMAGEFILE	N/A	.fv	
APRIORI	Not Supported.	.bin	Distribution of an Aprior file is not supported. These files are created during image generation stage based on content of a .FDF file. In EDK, the file was just a text file with a single line for each driver.
BASE	BASE	.efi	Modules of this type can be ported to any execution environment. This module type is intended to be use by silicon module developers to produce source code that is not tied to any specific execution environment.
			Modules of this type are designed to start execution at the reset vector of a CPU. They are responsible for preparing

SEC	SEC	.efi	the platform for the PEI Phase. Since there are no standard services defined for SEC, modules of this type follow the same rules as modules of type Base and typically include some amount of CPU specific assembly code to establish temporary memory for a stack. Modules of this type may optionally produce services that are passed to the PEI Phase in HOBs and those services must be compliant with the PEI CIS.
PEI_CORE	PEI_CORE	.efi	This module type is used by PEI Core implementations that are complaint with the PEI CIS.
PEIM	PEIM	.efi	This module type is used by PEIMs that are compliant with the PEI CIS
DXE_CORE	DXE_CORE	.efi	This module type is used by DXE Core implementations that are compliant with the DXE CIS.
DXE_DRIVER	DXE_DRIVER	.efi	This module type is used by DXE Drivers that are complaint with the DXE CIS. These modules only execute in the boot services environment and are destroyed when ExitBootServices() is called.
DXE_RUNTIME_DRIVER	DXE_RUNTIME_DRIVER	.efi	This module type is used by DXE Drivers that are complaint with the DXE CIS. These modules execute in both boot services and runtime services environments. This means the services that these modules produce are available after ExitBootServices() is called. If SetVirtualAddressMap() is called, then modules of this type are relocated according to virtual address map provided by the operating system.
DXE_SAL_DRIVER	DXE_SAL_DRIVER	.efi	This module type is used by DXE Drivers that can be called in physical mode before SetVirtualAddressMap() is called and either physical mode or virtual mode after SetVirtualAddressMap() is called. This module type is only available to IPF CPUs. This means the services that these modules produce are available after ExitBootServices().
SMM_CORE	SMM_CORE	.efi	This module is the SMM Core.
DXE_SMM_DRIVER	DXE_SMM_DRIVER	.efi	This module type is used by DXE. Drivers that are loaded into SMRAM. As a result, this module type is only available for IA-32 and x64 CPUs. These modules only execute in physical mode, and are never destroyed. This means the services that these

			modules produce are available after ExitBootServices().
UEFI_DRIVER	UEFI_DRIVER	.efi	This module type is used by UEFI Drivers that are compliant with the EFI 1.10 Specification or the UEFI 2.x Specification. These modules provide services in the boot services execution environment. UEFI Drivers that return EFI_SUCCESS are not unloaded from memory. UEFI Drivers that return an error are unloaded from memory.
UEFI_APPLICATION	UEFI_APPLICATION	.efi	This module type is used by UEFI Applications that are compliant with the EFI 1.10 Specification or the UEFI 2.x Specification. UEFI Applications are always unloaded when they exit.
USER_DEFINED	USER_DEFINED	.bin or .rom	User defined extension
EFI Dependency Section	Any - the code for these sections is included as part of any module, and no separate INF is required.	.dpx	This is the compiled dependency section for SMM, PEIM or DXE modules. A dependency section may also be generated from a dependency source (.dxs) file, if specified in the `[Sources]` section.
EFI User Interface Section	Any - the code for these sections is included as part of any module, and no separate INF is required.	.ui	This is a processed User Interface section
EFI Version Section	Any - the code for these sections is included as part of any module, and no separate INF is required.	.ver	This is a processed Version section

This following table shows the mapping of EDK II Module Types to EDK Component Types.

Table 10 Module Type to Component Type Mapping

EDK II Module Type	EDK Component Type
BASE, SEC, PEI_CORE, PEIM, DXE_CORE, DXE_DRIVER, DXE_RUNTIME_DRIVER, DXE_SAL_DRIVER, DXE_SMM_DRIVER, UEFI_DRIVER, UEFI_APPLICATION	LIBRARY
BASE	LIBRARY, SECURITY_CORE, PEI_CORE, COMBINED_PEIM_DRIVER, PIC_PEIM, RELOCATABLE_PEIM, BS_DRIVER, RT_DRIVER, SAL_RT_DRIVER, APPLICATION
SEC	SECURITY_CORE
PEI_CORE	PEI_CORE
PEIM	COMBINED_PEIM_DRIVER (See NOTE below)
PEIM	PIC_PEIM
PEIM	RELOCATABLE_PEIM

DXE_CORE	BS_DRIVER
DXE_DRIVER	BS_DRIVER with a Dependency Section
DXE_RUNTIME_DRIVER	RT_DRIVER
DXE_SAL_DRIVER	SAL_RT_DRIVER
DXE_SMM_DRIVER	BS_DRIVER
UEFI_DRIVER	BS_DRIVER without a Dependency Section.
UEFI_APPLICATION	APPLICATION

Note: The EDK II build system does not support creation of `COMBINED_PEIM_DRIVER` EFI leaf sections.