EDK II Flash Description (FDF) File Specification

TABLE OF CONTENTS

EDK II Flash Description (FDF) File Specification

Revision 1.28

04/28/2025 10:59:26

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 based on enhancement requests and errata	March 2010
	Added support for SMM_CORE	
	Added support for CAPSULE_FLAGS_INITIATE_RESET	
	Added Block Statements to all Capsule sections	
	Added "Auto" keyword to FFS alignment	
	Rule processing for file type lists is alphabetical, i.e., files are added in alphabetical order	
	HSD 203863 Macro Definitions in DSC file are now global to both DSC and FDF files	

	PCD Values may be constructed using C-style expressions provided the result of the expression matches the datum type of the PCD	
	FeatureFlagExpression is now defined as a C-style expression using C relational, equality and logical numeric and bitwise operators and/or arithmetic	
	and bitwise operators that evaluate to a value that matches the Datum Type of the PCD. Precedence and associativity follow C standards	
1.22	Grammatical and formatting editing	May 2010
1.22 w/	Updated to match the implementation at the time of the UDK2010 SR1 release:	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	
	Added NOTE in 3.2 saying the sections must appear in the order listed above.	
	Spelling and punctuation fixes	
	Do not require the FDF_SPECIFICATION to be updated if that is the only thing that changes.	
	Removed duplicate content and added the scoping rules for Macros, clarified MACRO Summary	
	Added statement to allow specifying an FD in a Capsule as well as to allow any file in a Capsule	
	Removed references to system environment variables in the Macros section	
	Clarify that C data arrays must be byte arrays for PCD value fields; allowing C format GUID structures as well.	
	Updated conditional rules in 2.2.8 for how to use macro and PCDs; updated PCD usage in conditional statements	
	Add clarification to require a comparison operator for string tokens (no atomic string) in expressions for conditional directives	
	Updated expression descriptions	
	Allow a PCD's value to be used in other sections, not just the section it was defined in	
	Added table of valid environment variables that can be used in this file; put in table of System Environment variables and the EDK II Global macro values.	
	Updated how macros can be used in 2.2.6; updated where macros are evaluated	
	Provide rules for how macros can be used in different BuildOptions sections	
	Make sure that macros are not restricted to only directories	
	Prohibit macros in the filename specified in !include statements	
	Updated 2.3.3 and 2.4.3 to allow macro and expression for value in a set statement	
	Added `<Filename>` to Macro values.	
	Clarify that only a limited number of system environment variables, not	

	macros can be used in the !include statement	
	Clarify the rules for finding the !include files	
	Clarify how macros can be shared between sections	
	Add statement about not expanding macros that are in a double quoted string.	
	Update macro usages in conditional directive statements	
	Put in rules for combining sections and where macros can be inherited	
1.22 w/	Updates:	December 2011
Errata A	Updated OptionROM section for COMPRESS to PCI_COMPRESS, matching the INF spec.	
(Cont.)	Removed CREATE_FILE from all FDF sections that had it	
	Add FvBaseAddress attribute in [FV] section	
	Added BCD Hex choice for COMP_VER in VTF section	
	Changed #elif to #elseif in 3.2.3 to match implementation	
	Removed duplicate definition of true in 3.2.3	
	Added EBNF for `<Extension>`	
	Allow DATA section to contain a GUID value	
	Add reserved keyword, BINARY as Rule name, updated rule override information for binary modules	
	Clarify that the VTF COMP_VER value is stored in BCD using 1 byte for major and one byte for the minor number, yielding a maximum value 99.99	
	Removed support for variable block size in the [FD] section - not currently supported by build tools	
	Removed paragraph about build options in section 2.2.6	
	Updated priority list; described precedence of the SET statements overriding previous definitions; removing set scoping statements	
	Update EBNF for conditionals and removed	
	`<SectionStatements>` in EBNF	
	Remove the word "should" and replace it with other text that means "recommended, but not required", replaced remaining instances of should with either must or will	
	Updated 2.2.8 to include text to describe examples	
	Added 2.2.9 section which includes expression table and text	
	Updated 2.2.10 to state that macros are evaluated when used, not when entered	
	Added the "IN" operator as an Equality Operator - added description and restriction of it's usage using	
	`<MemberExpression>`	
	Define how EDK_GLOBAL values can be used, but not defined in the FDF file	
	Add text regarding the FvForceRebase flag	
	Removed PART_NAME from [Defines] section, as it is not supported by	

	current tools and none of the FDF files have the [Defines] section	
	Removed the GuidCName from `<NamedGuid>` as this is not supported by tools at this time	
1.22 w/	Updates:	June 2012
Errata B	Section 1.3, Updated UEFI and PI specifications to reflect Errata	
	Section 2.2.6, Provide clarification on MACRO values	
	Section 3.8, Remove undocumented tags that begin with "SEC_" that are used internally by tools	
	Sections 2.4.6.2, 3.6, 3.7 and 3.8, Provide explanation of SUBTYPE_GUID in Parameters section	
	Sections 3.6, 3.7 and 3.8, Modify the EBNF entries for section SUBTYPE_GUID to allow specifying the GUID value	
	3.6, 3.7 and 3.8, Clarify User Interface entries in parameters section	
	Section 3.2.1, Remove invalid reference to ExtendedLine	
	Section 3.2.1, Fix the DOS EOL character sequence	
	The FDF Specification Version will not change in the `[Defines]` section	
1.22 w/	Updates	August 2013
Errata C	Section 1.3, Updated UEFI and PI specifications with Errata	
	Section 2.3,4.3, 2.4.5, 3.5, 3.6, 3.7, 3.8 and 3.10 Allow binary files to be located outside of the WORKSPACE	
	Section 2.4.6 and 2.4.6.1, added type DISPOSABLE to encapsulation file types	
	Sections 3.6, 3.7 and 3.8 Clarify that the plain text file for the User Interface is plain ASCII text	
	Section 2.1 Updated the figure to remove top-level Makefile	
	Section 2.1 Update location of the FDF file	
	Section 2.2.8,and 3.2.3 Clarify PCD usage in conditional directive statements	
	Included new section PCD RULES	
	Added reference to the EDK II Build Specification for PCD Processing Rules	
	Added `<Afile>` Definition to the Capsule section EBNF for appending the contents of a binary file to a section; appends are listed in order first in list first append,	
	second in list is appended after the first. These are strictly data files which can only be used by a driver that has a prior knowledge of the content	
1.24	Updates:	December 2014
	Changed specification version to 1.24	
	Updated FDF_SPECIFICATION to 0x00010018 and updated EBNF to specify this value as 1.24	
	Updated UEFI specification and EDK II meta data specifications in section 1.2; added the EDK II UNI Unicode File Specification and EDK II Expression Syntax Specification	

1.24 w/	Updates:	March 2015
Errata A	Update link to the EDK II Specifications, fixed the name of the Multi-String .UNI File Format Specification	
	When FvNameGuid is present in an FV section, the EDK II build tools will generate an FvName string in the FV EXT Header	
	Removed [UserExtensions] sections 2.9 and 3.11 as these sections should be used in DSC files, not the FDF file.	
1.25	Updates:	June 2015
	Updated to support UEFI 2.5 and PI 1.4	
	Added clarificaiton regarding use of .. and . in path names	
	Add support to generate an FMP Capsule in section 3.7 and added new section for FMP Payload data.	
	Since the above adds a new feature, update minor revision of FDF SPECIFICATION to 0x00010019 Only FDF files that use new feature need to use 0x0010019,	
	older FDF files that do not use the functionality do not need to modify the FDF_SPECIFICATION values	
	Added clarification on when the FDF_SPECIFICATION value must be updated to 0x00010019	
1.24 w/	Updates:	July 2015
Errata A	- Require a boolean flag, FvNameString to an FV section in order for the tools to generate the the FvNameString entry in an extended FV header.	
1.26	Updates:	January 2016
	Specification revision to 1.26	
	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).	
	Added new system environment variables,	
	PACKAGES_PATH and EDK_TOOLS_BIN, used by the build system.	
	Allow INF statements in FD regions.	
	Clarified [UserExtensions] content in chapter 2 (to match implementation)	
1.28	Convert to GitBooks	June 2017
	#426 IMAGE_TYPE_ID must be provided with value, FDF should mark it as required section	
	#373 Conditional statement examples incorrect	
	#461 FDF Spec: add a super script number for the	
	#249 FDF spec miss '' definition	
	#350 [FDF Spec] Extend the macro usage in the !include statement	
	Changed the FDF_SPECIFICATION value from 0x0001001A to 0x0001001B or 1.27	
	Extended the FV and Capsule, FILE RAW statement formats to support	

	multiple binary files.	
	Changed section 3.8 [FmpPayload] to add definitions for MONOTONIC_COUNT and CERTIFICATE_GUID, plus some notes about how these are used.	
	#142 Update EDK II FDF Specification to allow sections in any order	
	#478 FDF spec: extend the to support and	
	#353 Build spec: Allow nested includes in DSC and FDF files	
	#520 FDF spec: Update Precedence of PCD Values	
	#585 FDF Spec: Update the FDF_SPECIFICATION version to 0x0001001B or 1.27	

1 INTRODUCTION

This document describes the EDK II Flash Description (FDF) file format. This format was designed to support new build requirements of building EDK and EDK II modules within the EDK II build infrastructure.

The EDK II Build Infrastructure supports generation of current Unified EFI, Inc. (UEFI 2.5 and PI 1.4) compliant binary images.

The FDF file is used to describe the content and layout of binary images. Binary images described in this file may be any combination of boot images, capsule images or PCI Options ROMs.

Note: EDK II FDF file formats have no similarity to EDK FDF file formats. New utilities and functionality have been provided to process these files.

1.1 Overview

EDK II tools use INI-style text-based files to describe components, platforms and firmware volumes. The EDK II Build Infrastructure supports generation of binary images compliant with Unified EFI Forum (UEFI) specifications UEFI and Platform Initialization (PI).

The EDK II build processes, defined in the EDK II Build Specification, use separate steps to create EFI images. EDK Build Tools are included as part of the EDK II compatibility package. In order to use EDK II Modules or the EDK II Build Tools, EDK II DSC and FDF files must be used.

The EDK II FDF file is used in conjunction with an EDK II DSC file to generate bootable images, option ROM images, and update capsules for bootable images that comply with the UEFI specifications listed above.

The FDF file describes the content and layout of binary images that are either a boot image or PCI Option ROMs.

This document describes the format of EDK II FDF files that are required for building binary images for an EDK II platform. The goals are:

- **Compatibility** - No compatibility with EDK FDF files exists in format or tools.
- **Simplified platform build and configuration** - The FDF files simplify the process of adding EDK components and EDK II modules to a firmware volume on any given platform.

The EDK build tools are provided as part of the EdkCompatibilityPkg which is included in EDK II.

Table 1 shows the FDF compatibility between platform, module and component builds.

Table 1 EDK Build Infrastructure Support Matrix

	EDK FDF	EDK II FDF	EDK DSC	EDK II DSC
EDK Build Tools	YES	NO	YES	NO
EDK II Build Tools	NO	YES	NO	YES

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 meta-data package declaration file. This file declares all public elements of a package containing similar 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 meta-data platform description file. This file describes what gets built and makes statements that affect how it is built.

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 1.10or 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 glue 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.

Library Instance

An implementation of one or more library classes.

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`, `SMM_CORE`, `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 one of the following types of modules:

- 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 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 INF 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

This document is intended for persons performing UEFI or PI compliant platform development and support for different platforms.

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 Extensible Firmware 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 braces, 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 FDF DESIGN DISCUSSION

This section of the document provides an overview to processing Flash Description File (FDF) used to create firmware images, Option ROM images or bootable images for removable media. For the purposes of this design discussion, the FDF file will be referred to as FlashMap.fdf. By convention, the name "FlashMap.fdf" has been used for the flash description file. This is only a convention and developers can maintain different flash description files using different file names, for example a given size or model of flash part. The file can have any name, however it is recommended that developers use the FDF extension for all flash description files.

The flash description file is normally in the same directory as the platform description (DSC) file.

The remainder of this document uses "FDF" instead of "Flash Description File."

The EDK II Build generates UEFI and PI specification compliant binary images. The tools provided in the EDK and the EdkCompatibilityPkg module support earlier versions of the specifications.

This revision of the specification adds support for multiple binary files in an FV FILE RAW statement. FDF files that use this feature must use the new `FDF_SPECIFICATION = 0x0001001B` in the `[Defines]` section. Older FDF files do not need to update the `FDF_SPECIFICATION` value.

The EDK II build system has been updated to allow the setting of multiple paths that will be searched when attempting to resolve the location of EDK II packages. This new feature allows for more flexibility when designing a tree layout or combining sources from different sources. The new functionality is enabled through the addition of a new environment variable (PACKAGES_PATH).

The PACKAGES_PATH variable is an ordered list of additional search paths using the default path separator of the host OS between each entry (";" on Windows, ":" on Linux and OS/X). The path specified by the WORKSPACE variable always has the highest search priority over any PACKAGE_PATH entries. The first path (left to right) in the PACKAGES_PATH list has the highest priority and the last path has the lowest priority.

Build tools will stop searching when the first location is resolved.

For the remainder of this document, unless otherwise specified (using "system environment variable, WORKSPACE"), references to the `WORKSPACE` and `$(WORKSPACE)` refer to the ordered list of directories specified by the combination of `WORKSPACE + PACKAGES_PATH`. The build system will automatically join the directories and search these paths to locate content, with the first match terminating the search. For example given the following set of environment variables, and the MdeModulePkg is located in both the edk2 and edk2Copy directories, the build system would use the C:\work\edk2\MdeModulePkg when attempting to locate the MdeModulePkg.dec file.

```
set WORKSPACE=c:\work
set PACKAGES_PATH=c:\work\edk2;c:\work\edk2Copy
```

Build tools will stop searching when the first location is resolved.

Refer to the TianoCore.org web-site for more information on the EDK II build system.

Note: Path and Filename elements within the FDF are case-sensitive in order to support building on UNIX style operating systems. Names that are used in C code are case sensitive as well as MACRO names used as short-cuts within the FDF file. Use of "/../" in a path and "./" or "../" at the start of a path is prohibited.

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 "\" 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.

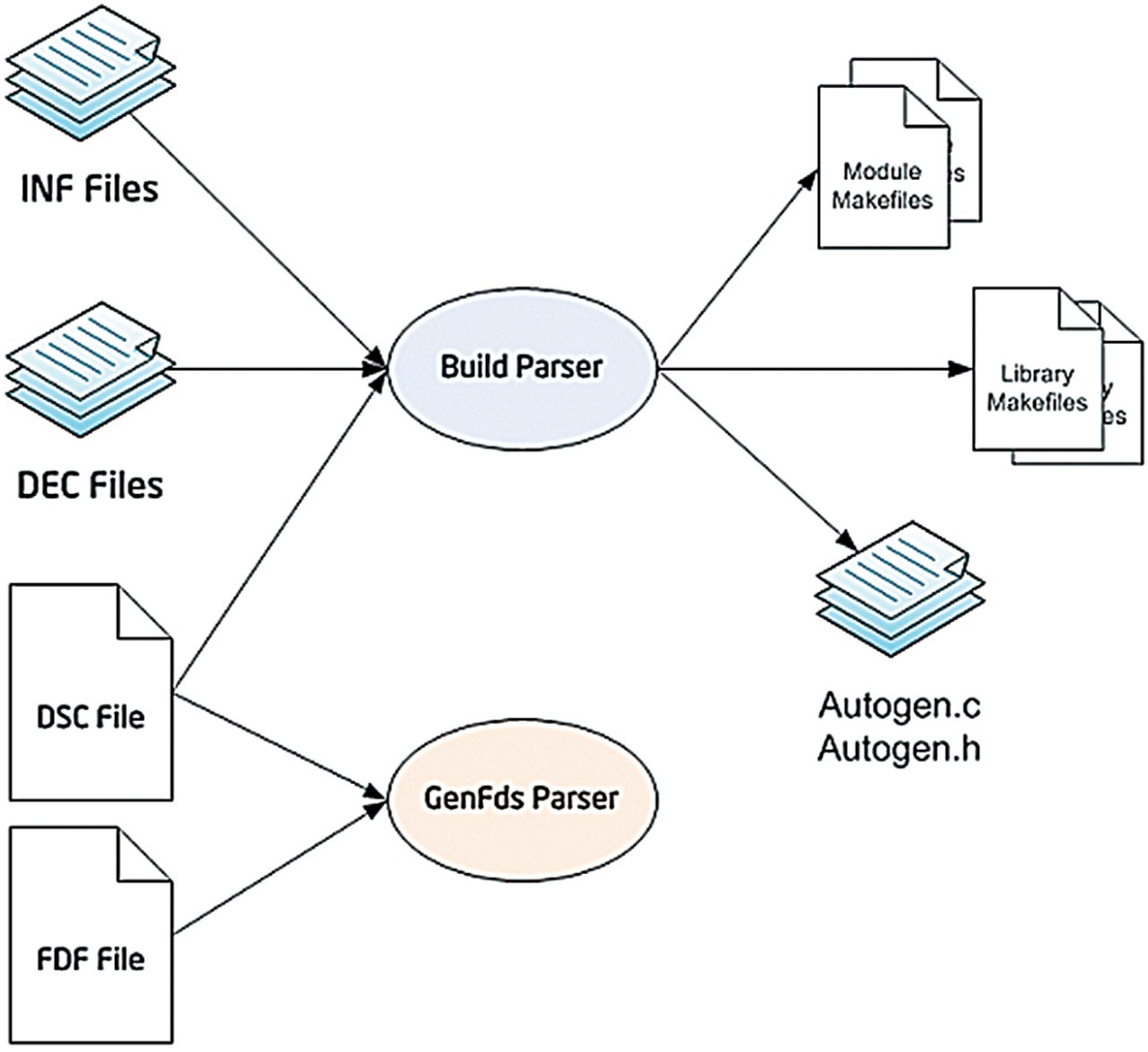
Note: The total path and file name length is limited by the operating system and third party tools. It is recommended that for EDK II builds that the project directories under a subst drive in Windows (s:/build as an example) or be located in either the /opt directory or in the user's /home/username directory for Linux and OS/X. This will minimize the path lengths of filenames for the command-line tools.

2.1 Processing Overview

The EDK II FDF file describes information about flash parts as well as rules for combining binaries (Firmware Image) built from a DSC file. Additionally, if a DSC file specifies a `FLASH_DEFINITION` file, then the EDK II tools will locate the FDF file (looking in the same directory as the DSC file, then the parsing utilities will scan the FDF file to gather PCD information that can be used by AutoGen utilities for building components or modules. The output of the first phase of an EDK II build (as defined in the EDK II Build Specification) generates valid PE32/PE32+/Coff image files. The second phase of the build process consumes the images generated during the first phase, using statements and rules defined in the FDF file to process the PE32/PE32+/Coff images files into one or more EFI sections. The EFI sections may get combined with other optional sections (version, depex, user interface) sections, into EFI Firmware File system (FFS) Sections. FFS images are put into Firmware Volumes (FVs,) and finally, the FV sections are combined into one or more Flash Device binary image (FD).

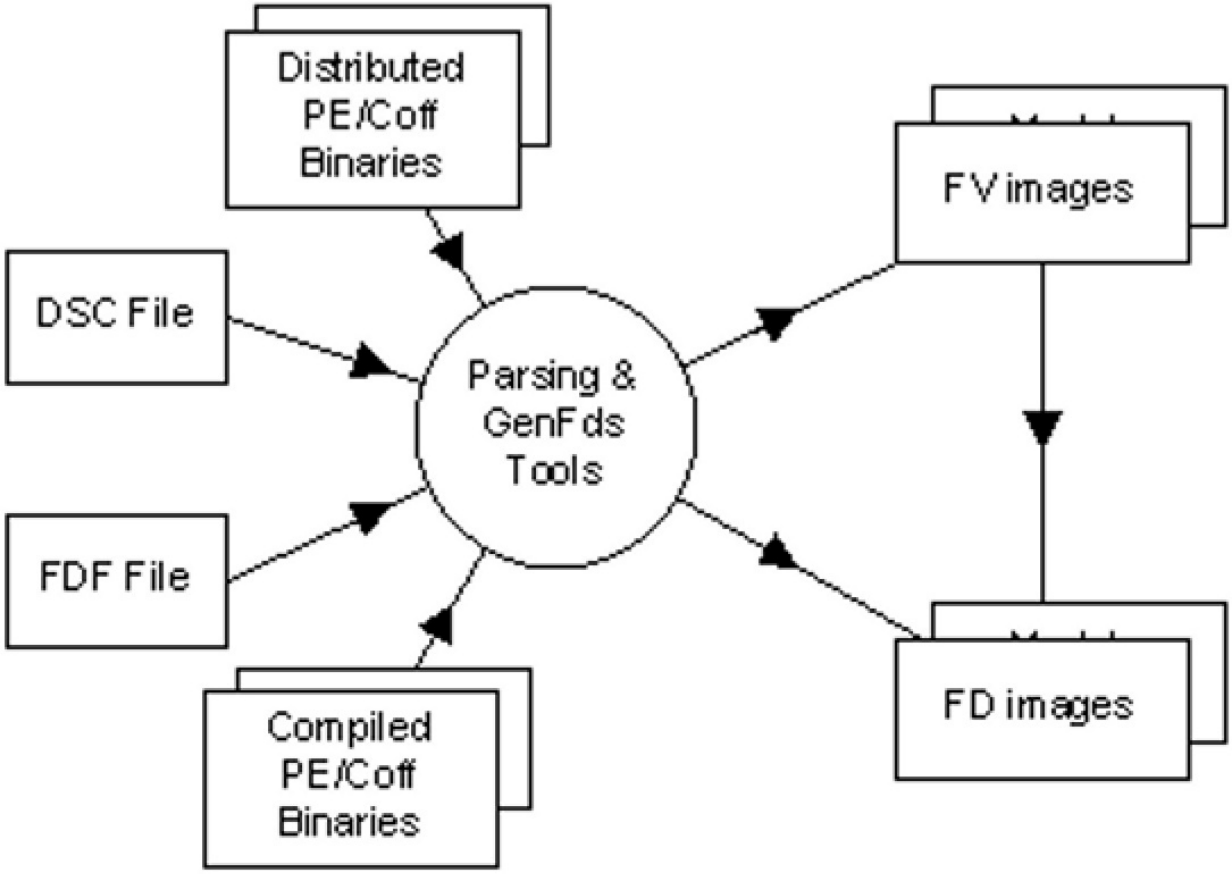
The following diagrams illustrate the process flow for generating the PE/PE32+/Coff files that will be used for Flash Image files.

Figure 1 EDK II Build Data Flow



The following diagram shows the overview of the process used to create final image files.

Figure 2 EDK II Create Image Flow



It should be noted that some `SEC`, `PEI_CORE` and/or `PEIM` modules are coded XIP (eXecute In Place) running directly from ROM, rather than from memory. For modules that always execute from ROM, the relocation (`.reloc`) section of the PE32 image can be removed (after running a fix up tool) as a space saving technique. Some PEIM modules may run from either ROM or from memory. There are several methods that can be used to retain this information (as well as the .reloc sections of stripped images). Due to this possibility, the `SEC`, `PEI_CORE` and `PEIM` descriptions are different from the remaining module types. The default for all `SEC`, `PEI_CORE` and `PEIM` modules is to strip the `.reloc` section. The modules coded to use `REGISTER_FOR_SHADOW` must not have the `.reloc` section stripped.

Also of note, not all of the INF files listed in the FDF file need to be listed in the DSC file. Since the DSC file is primarily used to generate Makefiles for a build, binary only modules do not need to be listed in a DSC file, and can be listed in the FDF file.

2.1.1 Platform Configuration Database (PCD) Settings

The FD, FV and Capsule sections (and nested sections) permit setting PCD default values. All PCDs must be declared in a DEC file in order to be used. It is recommended that the PCDs set in the FDF file be for Addresses, Sizes, and/or other "fixed" information needed to define or create the flash image. Use of PCDs is permitted in the FDF file. The `Dynamic` and `DynamicEx` PCDs can be accessed or modified during execution, as result, they cannot be set within the FDF file.

Note: The PCD values set in this file are assumed to be correct on all conditions that the reset vector in SEC is executed, such as power-on, reset and ACPI S3 resume. Use of the PatchableInModule PCD access method for base addresses is permitted, but when this PCD access method is used, module implementations must always access the values through the PcdGet() and PcdSet() operations to guarantee that stale base address values are never used.

All FLASH related PCD settings MUST be set in the FDF file, not in the platform description (DSC) file. The FDF file has the final values for Flash Related PCDs. If a DSC file contains a duplicate PCD setting, the FDF file's PCD setting takes precedence and it is recommended that the build tools throw a warning message on the PCD defined in the DSC file. Default values from DEC files are not permitted in the EDK II build system for PCDs specified in the FDF file.

The PCDs used in the FDF file must be specified as:

`PcdTokenSpaceGuidCName.PcdCName`

2.1.2 Precedence of PCD Values

The values that are assigned to individual PCDs required by a build may come from different locations and different meta-data files. The following provides the precedence (high to low) to assign a value to a PCD.

- Command-line, `--pcd` flags (left most has higher priority)
- DSC file, Component INF `<Pcd*>` section statements
- FDF file, grammar describing automatic assignment of PCD values
- FDF file, SET statements within a section
- FDF file, SET statement in the [Defines] section
- DSC file, global [Pcd*] sections
- INF file, PCD sections, Default Values
- DEC file, PCD sections, Default Values

In addition to the above precedence rules, PCDs set in sections with architectural modifiers take precedence over PCD sections that are common to all architectures.

If a PCD is listed in the same section multiple times, the last one is used.

PCD RULES

There are no PCD sections defined for the FDF file. PCD values are assigned in the FDF file using two methods. They may automatically be assigned based on a specific position (as defined in the FDF specification) or by using a SET statement.

- Within the `[FD]` sections, PCDs that appear immediately following the line containing address|size or offset|size values will have their values automatically assigned to the address|size or offset|size using the same ordering. Additionally, some tokens, such as `BaseAddress`, which use a format of token = value, can have a PCD's value set at the same time. The PCD must be on the same line, and be separated from the value by field separator character.If there are two values on the line, separated by a field separator character, the PCD names that will be assigned the values must appear on the next line, and must also be separated by the field separator character.
- Specific tokens assigned to values within the FD sections can have a PCD automatically assign the same value by placing a field separator character between the value and PCD name. The PCD name must always follow the value field separator character sequence.
- Other PCDs may be assigned using the `SET` statement in the FDF file.
- PCD Values assigned in this file (using `SET PcdName = value`) override values assigned in the DSC file, or, if a value is not specified in the DSC, then they will override values assigned in an INF file, or if a value is not specified in the DSC or the INF, then they will override values assigned in the DEC file.
- Dynamic and DynamicEx PCDs cannot be set in the FDF file.

2.2 Flash Description File Format

The EDK II FDF file describes the layout of UEFI/PI compliant binary images located within hardware, removable media or update capsules. The binary files must already exist in order for the build tools to create the final images. Some content, such as PCD definitions, may be used during the creation of binary files.

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.

It is recommended that duplicate section names be merged by tools.

This description file consists of sections delineated by section names enclosed within square "[]" brackets. Section names 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:

```
[Rule.IA32.SEC, Rule.X64.SEC]
```

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 modifier.

Note: Content such as filenames, directory names, MACROs and C variable names within a section IS case sensitive. IA32, Ia32 and ia32 within a section in a directory or file name 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.

Comments are not permitted between square brackets of a section specifier.

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

The EDK II Reference build system will ignore [UserExtensions] sections in the FDF file.

The `[Rules]` and `[VTF]` sections allow the use of architectural modifiers, however the content must specific to an individual architecture or common to all architectures.

Therefore, the architectural sections take priority over common section content. The cannot be combined with a 'common' architecture.

The `[FD]`, `[FV]`, `[Capsule]` and `[OptionRom]` sections cannot specify architectural modifiers.

2.2.2 Comments

The hash "#" character indicates comments in the FDF file. In line comments terminate the processing of a line. In line comments must be placed at the end of the line.

Only `BsBaseAddress = 0x0000C1000` in the following example is processed by tools; the remainder of the line is ignored:

```
BsBaseAddress = 0x0000C100 # set boot driver base address
```

Note: Blank lines and lines that start with the hash # character must be ignored by 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, NoSuch, LTD. All rights reserved."
```

Comments are terminated by the end of line.

If a hash "#" character is required in a value field, the value field must be encapsulated by double quotation marks.

2.2.3 Valid Entries

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

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

Items in quotation marks are treated as a single token and have the highest precedence. Items encapsulated in parenthesis are also treated as tokens, with embedded tokens being processed first. 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. As a result of multiple environment support, all directory and file names are case sensitive.

- The use of special characters in directory names and file names is restricted to the dash, underscore, and period characters, respectively "-", "_", and ".".
- Period characters may not be followed by another period character. File and Directory names must not start with "./", "." or "..".
- Directory names and file names must not contain space or tab characters.
- Directory Names must only contain alphanumeric characters, underscore or dash characters and it is recommended that they start with an alpha character.

- 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, an IA32 directory name is not equivalent to Ia32 or ia32 An architecture used in a directory name must be listed in a section that uses the architecture modifier. If a common section contains filenames that have directories with architecture modifiers, the file will be processed for all architectures, not just the architecture specified in the directory name.

Space Characters in filenames: 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. The tools_def.txt file is the only file the permits the use of space characters in the directory name.

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 FDF 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.

For clarity, this specification will use all upper case letters when describing architectural keywords, and the directory names with only the first letter in upper case.

All directory paths within EDK II FDF 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 white space characters are permitted. The special characters permitted in directory names are the only special characters permitted in file names.

The relative path is relative to the directory the FDF file must be used, unless otherwise noted. Use of "..", "./" and "../" in the path of the file is strictly prohibited. All files listed in this section must reside in the directory this INF file is in or in sub-directories of this directory.

2.2.5 !include Statements

The `!include` statement may appear within an EDK II FDF file. The included file content must match the content type of the current section definition, contain complete sections, or combination of both.

The argument of this statement is a filename. The file is relative to the directory that contains this DSC file, and if not found the tool must attempt to find the file relative to paths listed in the system environment variables, `$(WORKSPACE)` , `$(PACKAGES_PATH)` , `$(EFI_SOURCE)` , `$(EDK_SOURCE)` , and `$(ECP_SOURCE)` . If the file is not found after testing for the possible combinations, the parsing tools must terminate with an error.

Macros, defined in this FDF file or in the DSC file, are permitted in the path or file name of the !include statement, as these files are included prior to processing the file for macros. The system environment variables, `$(WORKSPACE)` , `$(EDK_SOURCE)` , `$(EFI_SOURCE)` , and `$(ECP_SOURCE)` may also be used; only these system environment variables are permitted to start the path of the included file.

Statements in !include files must not break the integrity of the FDF file, the included file is read in by tools in the exact position of the file, and is functionally equivalent of copying the contents of the included file and inserting (paste) the content into the DSC file.

2.2.6 Macro Statements

Variables (or macros) used within the FDF file are typically used for path generation for locating files, used in conditional statements or values for PCDs.

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.

MACROS cannot be used to define keywords, statements, nor any other tokens defined in this spec.

All elements of a macro definition must appear on a single line; the meta-data file formats do not permit entries to span multiple lines.

Escape character sequences are only permitted within a quoted string. Quoted strings are treated as literals, escape character sequences within quoted strings will not be expanded by the tools.

Macros that appear in a double quoted string will not be expanded by parsing tools. The expectation is that these macros will be expanded by scripting tools such as make or nmake.

The format and usage for the macro statements is:

```
DEFINE MACRO = Path
```

Any portion on a path or path and filename can be defined by a macro.

When assigning a string value to a macro, the string must follow the C format for specifying a string, as shown below:

```
DEFINE MACRO1 = "SETUP"
DEFINE MACRO2 = L"SETUP"
```

When assigning a numeric value to a macro, the number may be a decimal, integer or hex value, as shown below:

```
DEFINE MACRO1 = 0xFFFFFFFF
DEFINE MACRO2 = 2.3
DEFINE MACRO3 = 10
```

The format for usage of a Macro varies. When used as a value, the Macro name must be encapsulated by "$(" and ")" as shown below:

```
$(MACRO)/filename.foo
```

When a macro is tested in a conditional directive statement, determining whether it has been defined or undefined uses the following format:

```
!ifdef MACRO
```

Note: For backward compatibility, tools may allow $(MACRO) in the !ifdef and !ifndef statements. This functionality may disappear in future releases, therefore, it is recommended that platform integrators update their DSC files if they also alter other content.

When using string comparisons of Macro elements to string literals, the format of the conditional directive must be:

```
!if $(MACRO) == "Literal String"
```

Note: For backward compatibility, tools may allow testing literal strings that are not encapsulated by double quotation marks. This functionality may disappear in future releases, therefore, it is recommended that platform integrators update their DSC files if they also alter other content.

When testing Macro against another Macro:

```
!if $(MACROALPHA) == $(MACROBETA)
```

When testing a Macro against a value:

```
!if $(MACRONUM) == 2
```

or

```
!if $(MACROBOOL) == TRUE
```

When used in either the `!if` or `!elseif` statements or in an expression used in a value field, a macro that has not been defined has a value of 0.

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 that MACRO value is local to only that section.)

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

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 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. If a macro that is not defined is used in locations that are not expressions (where the tools would just do macro expansion as in path names in an INF statement in the `[FV]` section), nothing will be emitted. If the macro, MACRO1, has not been defined, then:

```
INF $(MACRO1)GraphicsDriver.inf
```

After macro expansion, the logical result would be equal to:

```
INF GraphicsDriver.inf
```

It is recommended that tools remove any excess space characters when processing these types of lines.

Additionally, pre-defined global variables may be used in the body of the FDF file. The following is an example of using pre-defined variables:

```
FILE = $(OUTPUT_DIRECTORY)/$(TARGET)_$(TOOL_CHAIN_TAG)/FV/Microcode.bin
```

The following table lists the global variables permitted in generating a path statement as well as variables that can be passed as an argument for a rule.

Macro statements defined the FDF file are local to the file. Macro names used in values, $(Macro), must be defined in either the DSC file or the FDF file, and must be defined before they can be used. Macro values specified on the command-line over ride all definitions of that Macro.

The `EDK_GLOBAL` macros can only be defined in the DSC file, however they are considered global during the processing of the DSC, FDF and EDK INF files.

Global variables that may be used in this file are listed in the Well-known Macro Statements table while the format of the System Environment variables that may be used in EDK II DSC and FDF files are in the next table.

Table 2 Well-known Macro Statements

Exact Notation	Derivation
$(WORKSPACE)	System Environment Variable.
PACKAGES_PATH	System Environment Variable that cannot be used in EDK II meta-data Files. The build system will automatically detect if this variable is present and use directories listed in this variable as if they were listed in $(WORKSPACE)
$(EDK_SOURCE)	System Environment Variable.
$(EFI_SOURCE)	System Environment Variable.
$(EDK_TOOLS_PATH)	System Environment Variable
EDK_TOOLS_BIN	System Environment Variable that cannot be used in EDK II meta-data Files.
$(ECP_SOURCE)	System Environment Variable
$(OUTPUT_DIRECTORY)	Tool parsing from either the DSC file or via a command line option. This is typically the Build/Platform name directory created by the build system in the EDK II WORKSPACE
$(BUILD_NUMBER)	Tool parsing from either an EDK INF file or the EDK II DSC file's BUILD_NUMBER statement. The EDK II DSC file's BUILD_NUMBER takes precedence over an EDK INF file's
	BUILD_NUMBER if and only if the EDK II DSC specifies a
	BUILD_NUMBER.
	Future implementation may allow for setting the
	BUILD_NUMBER variable on the build tool's command line.
$(NAMED_GUID)	Tool parsing FILE_GUID statement in the INF file.
$(MODULE_NAME)	Tool parsing the BASE_NAME statement in the INF file.
$(INF_VERSION)	Tool parsing the VERSION_STRING statement in the INF file.
$(INF_OUTPUT)	The OUTPUT directory created by the build system for each EDK II module.
$(TARGET)	Valid values are derived from INF, DSC, target.txt and tools_def.txt. FDF parsing tools may obtain these values from command-line options.
$(TOOL_CHAIN_TAG)	Valid values are derived from INF, DSC, target.txt and tools_def.txt. FDF parsing tools may obtain these values from command-line options.
$(ARCH)	Valid values are derived from INF, DSC, target.txt and tools_def.txt. FDF parsing tools may obtain these values from command-line options.

Note: System environment variables may be referenced, however their values must not be altered.

Table 3 Using System Environment Variable

Macro Style Used in Meta-Data files	Windows Environment Variable	Linux & OS/X Environment Variable
$(WORKSPACE)	%WORKSPACE%	$WORKSPACE
$(EFI_SOURCE)	%EFI_SOURCE%	$EFI_SOURCE
$(EDK_SOURCE)	%EDK_SOURCE%	$EDK_SOURCE
$(EDK_TOOLS_PATH)	%EDK_TOOLS_PATH%	$EDK_TOOLS_PATH
$(ECP_SOURCE)	%ECP_SOURCE%	$ECP_SOURCE

The system environment variables, `PACKAGES_PATH` and `EDK_TOOLS_BIN`, are not permitted in EDK II meta-data files.

Macros defined in the FDF file are local to the FDF file. They are also positional in nature, with later definitions overriding previous definitions for the remainder of the file.

Macros may be used in other macros or in conditional directive statements. Macros can be defined or used in the `[Defines]`, `[FD]`, `[FV]`, `[Capsule]` and `[OptionROM]` sections.

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. Macros must be defined before they can be used.

Macro evaluation is done at the time the macro is used in an expression, conditional directive or value field, not when a macro is defined. Macros in quoted strings will not be expanded by parsing tools; all other macro values will be expanded, without evaluation, as other elements of the build system will perform any needed tests.

Example

```
[FV.common]
  FILE FV_IMAGE = EF41A0E1-40B1-481f-958E-6FB4D9B12E76 {
    FvAlignment           = 512K
    WRITE_POLICY_RELIABLE = TRUE
    SECTION GUIDED 3EA022A4-1439-4ff2-B4E4-A6F65A13A9AB {
      SECTION FV_IMAGE = Dxe {
        APRIORI DXE {
          INF $(WORKSPACE)/a/a.inf
          INF $(EDK_SOURCE/a/c/c.inf
          INF $(WORKSPACE)/a/b/b.inf
        }
        INF a/d/d.inf
        ...
      }
    }
  }
```

The `[Rule]` section of the FDF file allows for using macros that are also defined for the EDK II build_rule.txt file. The following table provides the list of these pre-defined macro statements. These macros should never be expanded during the initial parsing phase, as other tools use these macros to generate the UEFI and PI compliant images. Additionally, the macro names should never be set by the user, as these values are filled in by the build tools based other file and base names.

Table 4 Reserved `[Rule]` Section Macro Strings

Variable String	Description
"${src}"	Source file(s) to be built (full path)
"${s_path}"	Source file directory (absolute path)
"${s_dir}"	Source file relative directory within a module.
	Note: ${s_dir} is always equals to "." if source file is given in absolute path.
"${s_name}"	Source file name without path.
"${s_base}"	Source file name without extension and path.
"${s_ext}"	Source file extension.
"${dst}"	Destination file(s) built from ${src} (full path)

"${d_path}"	Destination file directory (absolute path)
"${d_name}"	Destination file name without path.
"${d_base}"	Destination file name without extension and path
"${d_ext}"	Destination file extension

The `SET` and `DEFINE` statements are not permitted in the `[Rule]` section.

2.2.7 PCD Names

Unique PCDs are identified using the format to identify the named PCD:

```
PcdTokenSpaceGuidCName.PcdCName
```

The PCD's Name (`PcdName`) is defined as PCD Token Space Guid C name and the PCD C name - separated by a period "." character. PCD C names are used in C code and must follow the C variable name rules.

A PCD's values are positional with in the FDF file, and may be set by either the automatic setting grammar defined in this specification, or through `SET` statements. Once the PCD's value has been defined, it may be used anywhere within the FDF file; they are not limited to sections that they are defined in. PCD values may be absolute, values defined by macros, or expressions.

Refer to the EDK II Build Specification, Pre-Build AutoGen Stage chapter for PCD processing rules.

2.2.8 Conditional Statements (!if...)

Conditional statements are used by the build tools preprocessor function to include or exclude statements in the FDF file.

Most section definitions in the EDK II meta-data files have architecture modifiers in the section tags. Use of architectural modifiers in the section tag is the recommended method for specifying architectural differences. Some sections do not have architectural modifiers and there are some unique cases where having a method for specifying architectural specific items would be valuable, hence the ability to use these values.

Statements are prefixed by the exclamation "!" character. Conditional statements may appear anywhere within the FDF file.

Note: A limited number of statements are supported. This specification does not support every conditional statement that C programmers are familiar with.

Supported statements are:

```
!ifdef, !ifndef, !if, !elseif, !else and !endif
```

Refer to the Macro Statement section for information on using Macros in conditional directives.

When using the `!ifdef` or `!ifndef` , the macro name must be used; the macro name must not be encapsulated between `$(` and `)` . (For backward compatibility, macro names encapsulated between `$(` and `)` are allowed in FDF files that have `FDF_SPECIFICATION` versions less that `0x00010016` .)

When using a marco in the `!if` or `!elseif` conditionals, the macro name must be encapsulated between `$(` and `)` .

A macro that is not defined has a default value of 0 (`FALSE`) when used in a conditional comparison statement.

It is recommended you not use PCDs in the `!ifdef` or `!ifndef` statements. Using a PCD in an `!ifdef` or `!ifndef` statement will cause the build to break with an error message.

When using a PCD in the `!if` or `!elseif` conditionals, the PCD name `(TokenSpaceGuidCName.PcdCname)` must be used; the PCD name must not be encapsulated between "$(" and ")". Do not encapsulate the PCD name in the "$(" and ")" required for macro values or in the "PCD(" and ")" used in `[FV]` or `[Capsule]` sections as shown in the example below.

```
!if ( gTokenSpaceGuid.PcdCname == 1 ) AND ( $(MY_MACRO) == TRUE )
DEFINE FOO=TRUE
!endif
```

If the PCD is a string, only the string needs to be encapsulated by double quotation marks, while a Unicode string can have the double quoted string prefixed by "L", as in the following example:

```
!if gTokenSpaceGuid.PcdCname == L"Setup"
DEFINE FOO=TRUE
!endif
```

When used in `!if` and `!elseif` conditional comparison statements, it is the value of the Macro or the PCD that is used for testing, not the name of the macro or PCD.

Strings can only be compared to strings of a like type (testing an ASCII string against a Unicode format string must fail), numbers can only be compared against numbers and boolean objects can only evaluate to `TRUE` or `FALSE`. See the Operator Precedence table, in the Expressions section below for a list of restrictions on comparisons.

Using macros in conditional directives that contain flags for use in the `[BuildOptions]` sections of DSC files is not recommended.

If a PCD is used in a conditional statement, the value must first come from the FDF file, then from the DSC file. If the value cannot be determined from these two locations, the build system should break with an error message.

Note: PCDs, used in conditional directives, must be defined and the value set in either the FDF or DSC file in order to be used in a conditional statement; values from INF or DEC files are not permitted.

The following is an example of conditional statements.

```
!if ("MSFT" in $(FAMILY)) or ("INTEL" in $(FAMILY))
... statements
!elseif $(FAMILY) == "GCC"
... statements
!endif

!ifdef FOO
  !ifndef BAR
    # FOO defined, BAR not defined
  !else
    # FOO defined, BAR is defined
  !endif
!elseif $(BARFOO)
  # FOO is not defined, BARFOO is defined as TRUE
!elseif $(BARFOO) == "FOOBAR"
  # FOO is not defined, BARFOO is defined as FOOBAR
!else
  # FOO is not defined while BARFOO is either NOT defined or does not
  # equal "FOOBAR"
!endif
```

2.2.9 Expressions

Expressions can be used in conditional directive comparison statements and in value fields for Macros and PCDs in the DSC and FDF files.

Expressions follow C relation, equality, logical and bitwise precedence and associativity. Not all C operators are supported, only operators in the following list can be used.

Note: Due to the flexibility of the build system, a new operator, "IN" has been added that can be used to test whether an element is in a list. The format for this is `<Value> IN <MACRO_LIST>`, where `MACRO_LIST` can only be one of `$(ARCH)`, `$(FAMILY)`, `$(TOOL_CHAIN_TAG)` and `$(TARGET)`.

Use of parenthesis is encouraged to remove ambiguity.

When comparing a string to a number or boolean value, a warning message will be emitted. In this case, the tools will always evaluate the expression using the "==" and "EQ" operators to `FALSE`, using the "!=" and "NE" operators to `TRUE`; other operator comparisons are not supported, and will cause the build system to terminate with an error message. Comparing a number to a boolean value (no warning message will be emitted) will be evaluated normally, however, only the numeric value of 1 will be considered a match to the "==" and "EQ" operators against a boolean value of `TRUE`.

Additional scripting style operators may be used in place of C operators as shown in the table below.

Table 5 Operator Precedence and Supported Operands

Operator	Use with Data Types	Notes	Priority
`or`, `OR`, `||`	Number, Boolean		Lowest
`and`, `AND`, `&&`	Number, Boolean		
`|`	Number, Boolean		
`^`, `xor`, `XOR`	Number, Boolean	Exclusive OR	
`&`	Number, Boolean	Bitwise AND	
`==`, `!=`, `EQ`, `NE`, `IN`	All types	The IN operator can only be used to test a unary object for membership in a list	
		Space characters must be used before and after the letter operators Strings compared to boolean or numeric values using "==" or "EQ" will always return FALSE, while using the "!=" or "NE" operators will always return TRUE	
`<=`, `>=`, `<`, `>`, `LE`, `GE`, `LT`, `GT`	All	Space characters must be used before and after the letter operators	
	Number,	Cannot be used with strings - the system does not automatically do concatenation. Tools should report a	

	Boolean	warning message if these operators are used with both a boolean and number value	
`!` , `not` , `NOT`	Number, Boolean		Highest

The `IN` operator can only be used to test a literal string against elements in the following global variables:

$(FAMILY)

`$(FAMILY)` is considered a list of families that different `TOOL_CHAIN_TAG` values belong to. The `TOOL_CHAIN_TAG` is defined in the `Conf/target.txt` or on the command-line. The `FAMILY` is associated with the `TOOL_CHAIN_TAG` in the `Conf/ tools_def.txt` file (or the `TOOLS_DEF_CONF` file specified in the `Conf/target.txt` file) file. While different family names can be defined, `ARMGCC` , `GCC` , `INTEL` , `MSFT` , `RVCT` , `RVCTCYGWIN` and `XCODE` have been predefined in the `tools_def.txt` file.

$(ARCH)

`$(ARCH)` is considered the list of architectures that are to be built, that were specified on the command line or come from the `Conf/target.txt` file.

$(TOOL_CHAIN_TAG)

`$(TOOL_CHAIN_TAG)` is considered the list of tool chain tag names specified on the command line

$(TARGET)

`$(TARGET)` is considered the list of target (such as `DEBUG` , `RELEASE` and `NOOPT`) names specified on the command line or come from the `Conf/target.txt` file.

For logical expressions, any non-zero value must be considered `TRUE` .

Invalid expressions must cause a build break with an appropriate error message.

2.3 [Defines] Section

This is an optional section.

The [Defines] section may be used for version tracking of FDF files as well as the location for global DEFINE statements.

```
[Defines]
```

The format for the non-Macro entries in this section is:

```
Name = Value
```

2.4 [FD] Sections

The `[FD]` sections are made up of the definition statements and a description of what goes into the Flash Device Image. Each FD section defines one flash "device" image. A flash device image may be one of the following: Removable media bootable image (like a boot floppy image,) a System "Flash" image (that would be burned into a system's flash) or an Update ("Capsule") image that will be used to update and existing system flash.

Multiple FD sections can be defined in a FDF file.

The section header format is `[FD.FdUiName]` where the `FdUiName` can be any value defined by the user. If only a single FD is constructed for a platform then `FdUiName` is optional, and the processing tools will use the DSC file `[Defines]` section's `PLATFORM_NAME` value for creating the FD file.

An FD section is terminated by any other section header section or the end of the file.

This section is required for platform images, and not required for OptionROM images.

2.4.1 FD TOKEN Statements

The Token statements are used to define the physical part. These include the base address of the image, the size of the image, the erase polarity and block information. Only one of each of the valid token names can be defined in any one FD section, except as noted below.

```
Token = VALUE [| PcdName]
```

Only one token statement can appear on a single line, and each token statement must be on a single line. Multi-line token statements are not permitted.

There are five valid Token names defined by this specification.

BaseAddress

The base address of the FLASH Device.

Size

The size in bytes of the FLASH Device

ErasePolarity

Either 0 or 1, depending on the erase polarity of the Flash Device.

BlockSize

One or More - Size of a block, optionally followed by number of blocks. Multiple `BlockSize` statements are legal, and the first statement represents block 0 (the first block) and subsequent `BlockSize` statements represent blocks 1 - N.

NumBlocks

Zero or one - The number of continuous blocks of size, `BlockSize`. If `NumBlocks` is not present, the number of blocks defaults to 1.

An optional `PcdName` may follow the Token statement and is separated from the statement using a pipe "|" character. The `PcdName` is assigned `$(VALUE)`. Only one `PcdName` can be assigned a Token's Value.

2.4.2 FD DEFINE statements

`DEFINE` statements are used to define `MACRO` definitions that are scoped to the individual FD sections. `DEFINE` statements are processed in order, so a later `DEFINE` statement for a given `MACRO` over-writes the previous definition. The `DEFINE` statements are typically used for creating short-cut names for directory path names but may be used for identifying other items or values that will be used in later statements.

```
DEFINE MACRO = PATH
```

The following are examples of the DEFINE statement.

```
DEFINE FV_DIR           = $(OUT_DIR)/$(TARGET)_$(TOOL_CHAIN_TAG)/$(ARCH)
DEFINE MDE_MOD_TSPG     = gEfiMdeModulePkgTokenSpaceGuid
DEFINE NV_STOR_VAR_SIZE = PcdFlashNvStorageVariableSize
DEFINE FV_HDR_SIZE      = 0x48
DEFINE VAR_STORE_SIZE   = $(MDE_MOD_TSPG).$(NV_STOR_VAR_SIZE) - $(FV_HDR_SIZE)
```

The `$(MACRO)` can be used to create a shorthand notation that can be used elsewhere within the FDF file. Macro values may be scoped to subsections of the FDF file. Macros are also positional, with later values replacing values for macros at the same level. When tools process this file, the `$(MACRO)` name will be expanded in commands or files emitted from the tools. In the following example, `$(OUTPUT_DIRECTORY)` is a variable, whose value is found in the platform's DSC file, and this file assigns `OUT_DIR` as the variable name to use, with the same value as `$(OUTPUT_DIRECTORY)`:

```
DEFINE OUT_DIR = $(OUTPUT_DIRECTORY)
DEFINE FV_DIR = $(OUT_DIR)/$(TARGET)_$(TOOL_CHAIN_TAG)/$(ARCH)/FV
```

If the DSC file declares `OUTPUT_DIRECTORY = $(WORKSPACE)/Build/Nt32`, `TARGET = DEBUG`, target.txt uses `MYTOOLS` for the tool chain, and the platform is `IA32`, then a statement later in the section that references `$(FV_DIR)` is interpreted by the tools as being:

```
$(WORKSPACE)/Build/Nt32/DEBUG_MYTOOLS/IA32/FV
```

2.4.3 FD SET statements

`SET` statements are used to define the values of PCD statements. The current PCD maps for regions include extra PCD entries that define properties of the region, so the `SET` statement can occur anywhere within an FD section.

`SET` statements are positional within the FDF file.

```
SET PcdName = VALUE
```

Additionally, a PCD Name is made up of two parts, separated by a period "." character. The format for a `PcdName` is:

```
PcdTokenSpaceGuidCName.PcdCName
```

The following is an example of the `SET` statement:

```
SET gFlashDevicePkgTokenSpaceGuid.PcdEfiMemoryMapped = TRUE
```

The `VALUE` specified must match the PCD's datum type and must be the content data.

For a PCD that has a datum type of `VOID*`, the data can be a Unicode string, as in `L"text"`, a valid C data array (it must be either a C format GUID or a hex byte array), as in `{0x20, 0x01, 0x50, 0x00, 0x32, 0xFF, 0x00, 0xAA, {0xFF, 0xF0, 0x00, 0x00, 0x00}}`. For other PCD datum types, the value may be a boolean or a hex value, as in `0x0000000F`, with a value that is consistent with the PCD's datum type.

The value may also be a macro or it may be computed, using arithmetic operations, arithmetic expressions and or logical expressions. The value portion of the `SET` statement, when using any of these computations are in-fix expressions that are evaluated left to right, with items within parenthesis evaluated before the outer expressions are evaluated. Use of parenthesis is encouraged to remove ambiguity.

2.4.4 FD Region Layout

Following the FD defines section are lists of `Regions` which correspond to the locations of different images within the flash device. Currently most flash devices have a variable number of blocks, all of identical size. When "burning" an image into one of these devices, only whole blocks can be burned into the device at any one time. This puts a constraint that all layout regions of the FD image must start on a block boundary. To accommodate future flash parts that have variable block sizes, the layout is described by the offset from the `BaseAddress` and the size of the section that is being described. Since completely filling a block is not probable, part of the last block of a region can be left empty. To ensure that no extraneous information is left in a partial block, it is recommended that the block be erased prior to burning it into the device.

Regions must be defined in ascending order and may not overlap.

A layout region start with an eight digit hex offset (leading "0x" required) followed by the pipe "|" character, followed by the size of the region, also in hex with the leading "0x" characters.

The typical layout region is terminated by the start of another region or an FV Section header.

The format for an FD Layout Region is:

```
Offset|Size
[TokenSpaceGuidCName.PcdOffsetCName | TokenSpaceGuidCName.PcdSizeCName] ?
  [RegionType] ?
```

Setting the optional PCD names in this fashion is a shortcut. The two regions listed below are identical, with the first example using the shortcut, and the second using the long method:

```
0x000000|0x0C0000
gEfiMyTokenSpaceGuid.PcdFlashFvMainBaseAddress | gEfiMyTokenSpaceGuid.PcdFlashFvMainSize
FV = FvMain

0x000000|0x0C0000
SET gEfiMyTokenSpaceGuid.PcdFlashFvMainBaseAddress = 0x000000
SET gEfiMyTokenSpaceGuid.PcdFlashFvMainSize        = 0x0C0000
FV = FvMain
```

The shortcut method is preferred, as the user does not need to maintain the values in two different locations.

The `RegionType`, if specified, must be one of the following `FV`, `DATA`, `FILE`, `INF` or `CAPSULE`. Not specifying the `RegionType` implies that the region starting at the "Offset", of length "Size" must not be touched (unless the region will be used for VPD data). This type of region is typically used for event logs that are persistent between system resets, and modified via some other mechanism (and Each FD region has a `UiName` modifier, then the output image files uses the `UiName` modifier for the file name.

Note: Although sub-regions existed in EDK FDF files, EDK II FDF does not use the concept of subregions.

2.4.4.1 FV RegionType

The `FV RegionType` is a pointer to either one of the unique FV names that are defined in the `[FV]` section. Both of these are files that contains a binary FV as defined by the PI 1.0 specification. The format for the `FV RegionType` is the following:

```
FV = UiFvName
```

The following is an example of FV region type:

```
0x000000|0x0C0000
gEfiMyTokenSpaceGuid.PcdFlashFvMainBaseAddress | gEfiMyTokenSpaceGuid.PcdFlashFvMainSize
FV = FvMain
```

2.4.4.2 DATA RegionType

The `DATA RegionType` is a region that contains is a hex byte value or an array of hex byte values. This data that will be loaded into the flash device, starting at the first location pointed to by the Offset value. The format of the `DATA RegionType` is:

DATA = { }

The following is an example of a DATA region type.

```
0x0CA000 | 0x002000
gEfiMyTokenSpaceGuid.PcdFlashNvStorageBase | gEfiMyTokenSpaceGuid.PcdFlashNvStorageSize
DATA = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x8D, 0x2B, 0xF1, 0xFF, 0x96, 0x76, 0x8B, 0x4C
}
```

The `!include` statement is valid for hex array portion of the `DATA RegionType`. The following is a valid usage for the `!include` statement.

```
0x0CA000 | 0x002000
gEfiMyTokenSpaceGuid.PcdFlashNvStorageBase | gEfiMyTokenSpaceGuid.PcdFlashNvStorageSize
DATA = {
  !include NvStoreInit.txt
}
```

2.4.4.3 FILE RegionType

The `FILE RegionType` is a pointer to a binary file that will be loaded into the flash device, starting at the first location pointed to by the `Offset` value. It should be noted that a file can be fully qualified path and filename that is outside of the current WORKSPACE (or the directories listed in PACKAGES_PATH system environment variable). The file must be a binary (.efi) or a raw binary file. The format of the `FILE RegionType` is:

```
FILE = $(FILE_DIR)/Filename.bin
```

Caution: If a fully qualified path and filename is specified, the platform integrator must ensure that all developers using the DSC and FDF file are aware of the requirements for this path.

The following is an example of the `FILE RegionType`.

```
0x0CC000|0x002000
gEfiCpuTokenSpaceGuid.PcdCpuMicrocodePatchAddress | gEfiCpuTokenSpaceGuid.PcdCpuMicrocodePatchSize
FILE = $(OUTPUT_DIRECTORY)/$(TARGET)_$(TOOL_CHAIN_TAG)/X64/Microcode.bin

# VPD Data Region
0x0026D000|0x00001000
gEfiMdeModulePkgTokenSpaceGuid.PcdVpdBaseAddress

FILE = $(OUTPUT_DIRECTORY)/$(TARGET)_$(TOOL_CHAIN_TAG)/FV/8C3D856A-9BE6468E-850A-24F7A8D38E08.bin
```

2.4.4.4 Capsule RegionType

The `CAPSULE RegionType` is a pointer to a capsule section `UiName` that will be loaded into the flash device, starting at the first location pointed to by the `Offset` value. The format of the `FILE RegionType` is:

```
CAPSULE = UiCapsuleName
```

The following is an example of the `CAPSULE RegionType`.

```
0x0CC000|0x002000
gEfiTokenSpaceGuid.PcdCapsuleOffset | gEfiTokenSpaceGuid.PcdCapsuleSize
CAPSULE = MyCapsule
```

2.4.4.5 INF Region Type

The INF statements point to EDK component and EDK II module INF files. Parsing tools will scan the INF file to determine the type of component or module. The component or module type is used to reference the standard rules defined elsewhere in the FDF file.

The format for INF statements is:

```
INF [Options] PathAndInfFileName
```

The PathAndInfFileName is the WORKSPACE (or PACKAGES_PATH) relative path and filename.

2.4.4.6 No RegionType Specified

It is permissible to define a region with no data pre-loaded. For example, event logging needs a data region to store events. This region is filled with data that matches the `ErasePolarity` bit during the initial installation of the firmware or through UEFI or operating system commands and services.

An example of no region type specified is:

```
0x0CE000|0x002000
gEfiMyTokenSpaceGuid.PcdFlashNvStorageEventLogBase | gEfiMyTokenSpaceGuid.PcdFlashNvStorageEventLogBase
```

2.5 [FV] Sections

The `[FV]` sections are required for platform images, are optional for Capsule images, and are not required for Option ROM only images. The `[FV]` section defines what components or modules are placed within a flash device file. These sections define the order the components and modules are positioned within the image. The `[FV]` section consists of define statements, set statements and module statements. A single `[FV]` section is terminated by the start of another section header or the end of the file. The `[FV]` section has one required modifier, a user defined section name. The format for `[FV]` section header is:

```
[FV.UiFvName]
```

The FV `UiFvName` must be unique for every declared user defined name within the file. The `UiFvName` is used for specifying the file that will be used in and `[FD]` section.

Nesting of `[FV]` sections is permitted, making it possible to create a tree structure containing multiple FV sections within a single `[FV]` section. Nested sections are specified using either the syntax "FILE FV_IMAGE = Name" or "SECTION FV_IMAGE = Name" within the top `[FV]` Section.

Use of the `UiFvName` modifier permits the user to include, by `UiFvName`, previously defined sections within another FV section. This eliminates the need to re-specify components or modules in multiple places. When the `FvNameString` entry is present and set to TRUE in an `[FV]` section, the tools will generate an `FvNameString` entry in FV EXT header using the `UiFvName`.

This section also specifies how to define content for PI FV Extensions which provides a mapping between a GUID and an OEM file type. The size of `EFI_FIRMWARE_VOLUME_EXT_HEADER` and `EFI_FIRMWARE_VOLUME_EXT_ENTRY` sizes will be calculated based on content, while the `EFI_FIRMWARE_VOLUME_EXT_ENTRY` type must be defined by the platform integrator based on the PI specification, volume 3 The content is limited to the contents of a binary file specified by a FILE statement or a data array specified by a `DATA` statement.

The `EFI_FIRMWARE_VOLUME_EXT_ENTRY_OEM_TYPE` (using `TYPE=0x0001`) is only support by including a file or data structure that completes the remainder of the OEM type entry, where the first entry would have to be a `UINT32` representing the TypeMask. Additional types defined by the PI spec will be supported in this manner as well.

The build system permits a recovery feature that allows placing two copies of a `PEI.fv` in the flash top. If the top one is corrupt, backup one will be swapped to the top and work.

The backup must be the same as the top PEI.fv although it is placed into another place. Therefore, the backup FV must be rebased to run at another address. The `FvBaseAddress` and the optional `FvForceRebase` attributes must be above `FvAlignment` attribute.

2.5.1 DEFINE Statements

`DEFINE` statements are used to define Macro definitions that are scoped to the individual `[FV]` sections. `DEFINE` statements are processed in order, so a later `DEFINE` statement for a given `MACRO` over-writes the previous definition for the remainder of the section or sub-section. The `DEFINE` statements are typically used for creating short-cut names for directory path names, but may be used for identifying other items or values that will be used in later statements.

```
DEFINE MACRO = VALUE
```

The following are examples of the `DEFINE` statement.

```
DEFINE EDKMOD           = $(WORKSPACE)/EdkModulePkg/
DEFINE MDE_MOD_TSPG     = gEfiMdeModulePkgTokenSpaceGuid
DEFINE NV_STOR_VAR_SIZE = PcdFlashNvStorageVariableSize
```

```
DEFINE FV_HDR_SIZE      = 0x48
DEFINE VAR_STORE_SIZE   = $(MDE_MOD_TSPG).$(NV_STOR_VAR_SIZE) - $(FV_HDR_SIZE)
```

2.5.2 Block Statements

`BLOCK` statements are used to define block size and the number of blocks that are needed for FV images that do NOT get put into a physical flash part, such as the recovery image that gets put on a floppy or USB key.

```
BLOCK_SIZE = VALUE
NUM_BLOCKS = VALUE
```

The following is an example of the BLOCK statement:

```
BLOCK_SIZE = 512
```

2.5.3 FV SET Statements

`SET` statements are used to define the values of PCDs. These statements are positional within the FDF file. `SET` statements set a `PcdName` to a `VALUE` for this FV.

```
SET <PcdName> = VALUE
```

The following is an example of the `SET` statement:

```
SET gEfiMyTokenSpaceGuid.PcdDisableOnboardVideo = TRUE
```

The `VALUE` specified must match the Pcd's datum type and must be the content data.

For a PCD that has a datum type of `VOID*`, the data can be a Unicode string, as in `L"text"`, a valid C data array (it must be either a C format GUID or a hex byte array), as in `{0x20, 0x00, 0x20, 0x00, 0x32, 0xFF, 0x00, 0xAA, {0xFF, 0xF0, 0x00, 0x00, 0x00}}`. Other PCD datum types are either boolean values or a hex value, as in `0x0000000F`, with a value that is consistent with the PCD's datum type

The value may also be a macro or it may be computed, using arithmetic operations, arithmetic expressions and or logical expressions. The value portion of the `SET` statement, when using any of these computations are in-fix expressions that are evaluated left to right, with items within parenthesis evaluated before the outer expressions are evaluated. Use of parenthesis is encouraged to remove ambiguity.

2.5.4 APRIORI Scoping

Within some firmware volumes, an `APRIORI` file can be created which is a GUID named list of modules in the firmware volume. The modules will be invoked or dispatched in the order they appear in the `APRIORI` file. Within a Firmware Volume, only one PEI and one DXE Apriori file are permitted. Since nested Firmware Volumes are permitted, Apriori files are limited to specifying the files, not FVs that are within the scope of the FV image in which it is located. (It is permissible for nested FV images to have one PEI and one DXE Apriori file per FV.) Scoping is accomplished using the curly "{}" braces.

The following example demonstrates an example of multiple APRIORI files.

```
[Fv.Root]
  DEFINE NT32     = $(WORKSPACE)/EdkNt32Pkg
  DEFINE BuildDir = $(OUTPUT_DIRECTORY)/$(PLATFORM_NAME)/$(TARGET)_$(TOOL_CHAIN_TAG)
  APRIORI DXE {
    FILE DXE_CORE = B5596C75-37A2-4b69-B40B-72ABD6DD8708 {
      SECTION COMPRESS {
        SECTION PE32 = $(BuildDir)/X/Y/Z/B5596C75-37A2-4b69-B40B-72ABD6DD8708-DxeCore.efi
        SECTION VERSION "1.2.3"
      }
    }
```

```
    INF VERSION = "1" ${NT32)/Dxe/WinNtThunk/Cpu/Cpu.inf
  }

  FILE FV_IMAGE = EF41A0E1-40B1-481f-958E-6FB4D9B12E76 {
    SECTION GUIDED 3EA022A4-1439-4ff2-B4E4-A6F65A13A9AB {
      SECTION FV_IMAGE = Dxe {
        APRIORI DXE {
          INF a/a/a.inf
          INF a/c/c.inf
          INF a/b/b.inf
        }
        INF a/d/d.inf
        ...
      }
    }
  }
```

In the example above, There are three FFS files in the `Fv.Root` and one Encapsulated FV image, with the build tools creating an `APRIORI` file that will dispatch the `DXE_CORE` first, then the CPU module second. In the FV image, named Dxe, there will be at least five FFS files, the `APRIORI` file, listing the GUID names of `a.inf`, `c.inf` and `b.inf`, which will be dispatched in this order. Once complete, the `d.inf` module will be dispatched.

2.5.5 INF Statements

The INF statements point to EDK component and EDK II module INF files. Parsing tools will scan the INF file to determine the type of component or module. The component or module type is used to reference the standard rules defined elsewhere in the FDF file.

The format for INF statements is:

`INF [Options] PathAndInfFileName`

The `PathAndInfFileName` is the `WORKSPACE` (or `PACKAGES_PATH`)relative path and filename.

Using an INF statement will cause the build tools to implicitly build an FFS file with the `EFI_FV_FILETYPE` based on the INF module's `MODULE_TYPE` and content. For example, specifying the following lines in an FV section will generate an FFS file with an `EFI_FV_FILETYPE_DRIVER` with three sections, the `EFI_SECTION_PE32`, an `EFI_SECTION_VERSION`, and an `EFI_SECTION_DXE_DEPEX`. While there is no version file defined in the INF - it has been specified by the `VERSION` option; and there is a dependency file specified in the INF file's source file list.

```
DEFINE MMP_U_MT = MdeModulePkg/Universal/MemoryTest
INF VERSION = "1.1" $(MMP_U_MT)/NullMemoryTestDxe/NullMemoryTestDxe.inf
```

Valid options for the INF file line are:

RuleOverride = RuleName

This permits the platform integrator with a method to override the default rules built into tools, specified in the EDK II Build Specification which follows the UEFI and PI specifications for EFI FileType construction. If the module is a binary module, the default rules are implied by the processor, module type and `BINARY` rule name. Using the explicit named rule here may compromise the platform's PI specification compliance. The RuleName is either the reserved word, `BINARY` (that only applies to INF files that contain only binary content), or the RuleUiName of a `[Rule]` section in this FDF file.

USE = ARCH

The `USE = ARCH` option is used to differentiate if a single INF file is built different ways, for example a single INF file is called out multiple times in the DSC file when building the same module for more than one processor architecture.

VERSION = "String"

The `VERSION` option is used to create an `EFI_SECTION_VERSION` section with the FFS file.

UI = "String"

The UI option is used to create an `EFI_SECTION_USER_INTERFACE` section for an INF that may not have specified one.

When `RuleOverride` is not specified for binary module INF, **GenFds** tool will find the FFS rule with `BINARY` rule name. If `RuleOverride` is specified, **GenFds** tool will still find FFS rule with the specified rule name.

The following are examples of INF statements:

```
DEFINE IFMP = IntelFrameworkModulePkg
INF USE = IA32 $(EDK_SOURCE)/Sample/Universal/Network/Ip4/Dxe/Ip4.inf
INF $(EDK_SOURCE)/Sample/Universal/Network/Ip4Config/Dxe/Ip4Config.inf
INF RULE_OVERRIDE = MyCompress $(IFMP)/Bus/Pci/IdeBusDxe/IdeBusDxe.inf
```

2.5.6 FILE Statements

`FILE` statements are provided so that a platform integrator can include complete EFI FFS files, as well as a method for constructing FFS files using curly "{}" brace scoping.

FFS file specification syntax is one of the following:

```
FILE Type $(NAMED_GUID) [Options] FileName
```

OR

```
FILE Type $(NAMED_GUID) [Options] {
  SECTION SECTION_TYPE = FileName
  SECTION SECTION_TYPE = FileName
}
```

The first statement is commonly used with `EFI_FV_FILETYPE_RAW` files, while the second type is used for most other file types. The FileName is typically a binary file, and the consumer of this type of file must have an a priori knowledge of the format.

The following describes the information that can be specified a File:

Type

EFI FV File Types - one and only one of the following:

- `RAW` - Binary data
- `FREEFORM` - Sectioned binary data
- `SEC` - Sectioned data consisting of an optional pad section, a terse section and an optional raw section.
- `PEI_CORE` - Sectioned data consisting of one PE32, one user interface and one version section.
- `DXE_CORE` - Sectioned data containing one or more other sections.
- `PEIM` - Dispatched by PEI Core
- `DRIVER` - Dispatched by DXE core
- `COMBO_PEIM_DRIVER` - Combined PEIM/DXE driver containing PEI and DXE depex sections as well as PE32 and version sections.[1]
- `SMM_CORE` - Sectioned data containing one or more other sections.

- `DXE_SMM_DRIVER` - Dispatched by the SMM Core
- `APPLICATION` - Application, so will not be dispatched
- `FV_IMAGE` - File contains an FV image
- `DISPOSABLE` - This section type is not supported by the EDK II build system
- `0x00 - 0xFF` - Hex values are legal too. See PI specification Volume 3 for details

NAMED_GUID

The `$(NAMED_GUID)` is usually constructed from an INF file's `[Defines]` section `FILE_GUID` element.

Options

The `Fixed` and `Checksum` attributes are boolean flags, both default to `FALSE`, specifying "Fixed" enables the flag to `TRUE`.

The Alignment attribute requires the "= value".

- `Fixed` - File can not be moved, default (not specified) is relocate-able.
- `Alignment` - Data (value is one of: 1, 2 4, 8, 16, 32, 64 128, 512, 1K, 2K, 4K, 8K, 16K, 32K, 64K) byte aligned
- `Checksum` - It is recommended that this be controlled on an entire FV basis not at the file level, however, we are including this attribute for completeness.

UEFI and PI Specifications have rules for file type construction that, by default, will be used by the tools.

In addition to the arguments on the `FILE` line, for EFI FV File types that are not `RAW`, additional EFI section information must be specified.

To specify additional section information for a file, the EFI Encapsulation Sections must be contained within curly "{}" braces that follow the `FILE` line, while leaf sections are denoted by an `EFI_SECTION` type keyword. Encapsulation and leaf section types are described below.

Caution: If a fully qualified path and filename are specified, the platform integrator must ensure that all developers using the DSC and FDF file are aware of the requirements for this path.

The following is an example for using additional sections:

```
#Encapsulation - Compress
FILE FOO = 12345678-0000-AAAA-FFFF-0123ABCD12BD {
  SECTION COMPRESS {
    SECTION PE32 = $(WORKSPACE)/EdkModulePkg/Core/Dxe/DxeMain.inf
    SECTION VERSION = "1.2.3"
  }
}

# Encapsulation - GUIDED
FILE FV_IMAGE = 87654321-FFFF-BBBB-2222-9874561230AB {
  SECTION GUIDED gEfiTianoCompressionScheme {
    SECTION PE32 = $(WORKSPACE)/EdkModulePkg/Core/Dxe/DxeMain.inf
  }
}

# LEAF Section
FILE DXE_CORE = B5596C75-37A2-4b69-B40B-72ABD6DD8708 {
  SECTION VERSION $(BUILD_DIR)/$(ARCH)/D6A2CB7F-6A18-4E2F-B43B-9920A733700A-DxeMain.ver
}
```

1

[1]. The EDK II build system does not support creation of `COMBO_PEIM_DRIVER` FV type. ↩

2.5.6.1 EFI Encapsulation Sections

There are two types of encapsulation sections, a `COMPRESSION` section and the `GUIDED` section. The `DISPOSABLE` encapsulation section is not supported by the EDK II build system.

The `COMPRESS` encapsulation section uses the following format.

```
SECTION COMPRESS [type] {
  SECTION EFI_SECTION_TYPE = FILENAME
  SECTION EFI_SECTION_TYPE = "string"
}
```

The `[type]` argument is optional, only `EFI_STANDARD_COMPRESSION` is supported by the PI specification. The current EDK enumerations for compression are a violation of the PI specification, and `SECTION GUIDED` must be used instead.

The `EFI_SECTION_TYPE` and `FILENAME` are required sub-elements within the compression encapsulation section. for most sections. However both the `VERSION` (`EFI_SECTION_VERSION`) and `UI` (`EFI_SECTION_USER_INTEFACE`) may specify a string, that will be used to create an EFI section.

The `GUIDED` encapsulation section uses one of the following formats.

```
SECTION GUIDED $(GUID_CNAME) [auth] {
  SECTION EFI_SECTION_TYPE = FILENAME
  SECTION EFI_SECTION_TYPE = "string"
}

SECTION GUIDED $(GUID_CNAME) [auth] FILENAME
```

The required argument is the `GUIDED` name followed by an optional "auth" flag. If the argument "auth" flag is specified, then the attribute `EFI_GUIDED_SECTION_AUTH_STATUS_VALID` must be set.

For non-scoped statements (the second `SECTION` statement of the two listed above,) if filename exists the Attribute `EFI_GUIDED_SECTION_PROCESSING_REQUIRED` must be set to `TRUE`. The file pointed to by filename is the data. If filename does not exist `EFI_GUIDED_SECTION_PROCESSING_REQUIRED` is cleared and normal leaf sections must be used.

2.5.6.2 EFI Leaf Sections

Leaf sections are identified using the `EFI_SECTION` Type, as specified in the UEFI specification. Arguments to the `EFI_SECTION` Type include information that will be used to build a leaf section. Nesting of leaf sections within leaf sections is not permitted, as a leaf section is defined as UEFI's smallest entity.

The LEAF section is specified using the following format.

`SECTION LEAF_SECTION [build #] [Align=X] [Unicode String][Filename]`

The following keywords are used for valid `LEAF_SECTION` types.

- `PE32`
- `PIC`
- `TE`
- `DXE_DEPEX`
- `SMM_DEPEX`
- `PEI_DEPEX`
- `VERSION` -- Contains either a 16-bit build number or a Unicode string
- `UI` -- Unicode String

- `COMPAT16`
- `FV_IMAGE`
- `SUBTYPE_GUID` -- A GUID value with content defined by the GUID to be used with the section type of `EFI_SECTION_FREEFORM_SUBTYPE_GUID`.

The argument, `build #`, is only valid for `VERSION` leaf section. The number may be specified in the platform description (DSC) file's `[Defines]` section, `BUILD_NUMBER` element. EDK INF files may specify a `BUILD_NUMBER` in the defines section. However, this value is only used if the EDK II DSC file does not contain a `BUILD_NUMBER` statement.

The **Filename** is only optional for `VERSION` and `UI`.

A Unicode string is only valid for `VERSION` or `UI` if the Filename is not present, and is of the form `L"string"`.

The remaining leaf section types require the **Filename** argument. The file must contain the data for the section.

2.6 [Capsule] Sections

The optional `[Capsule]` sections are used to define the components and modules that make up formatted, variable-length data structure. Capsules were designed for and are intended to be the major vehicle for delivering firmware volume changes to an existing implementation. An update capsule is commonly used to update the firmware flash image or for an operating system to have information persist across a system reset. A `[Capsule]` header section requires one modifier, the `UiCapsuleName` modifier. Zero or more `[Capsule]` sections can be present in a FDF file.

The following is the format for the `[Capsule]` section header:

```
[Capsule.UiCapsuleName]
```

The first elements of a `[Capsule]` section are required Token elements, using the following format.

```
Token = VALUE
```

2.6.1 UEFI Implementation

The UEFI specification defines the `EFI_CAPSULE_HEADER` structure in the runtime services chapter. The header consists of the following elements. The following tokens are required in a capsule conforming to the UEFI specification.

EFI_CAPSULE_GUID

The GUID that defines the contents of a capsule, used by the EFI system table, which must point to one or more capsules that have the same `EFI_CAPSULE_GUID` value.

EFI_CAPSULE_HEADER_SIZE

Size in bytes of the capsule header. If the size specified here is larger than the size of the `EFI_CAPSULE_HEADER`, then the capsule `GUID` value implies extended header entries.

EFI_CAPSULE_FLAGS

Currently, three bit flags have been defined:

```
PersistAcrossReset  = CAPSULE_FLAGS_PERSIST_ACROSS_RESET
InitiateReset       = CAPSULE_FLAGS_INITIATE_RESET and
PopulateSystemTable = CAPSULE_FLAGS_POPULATE_SYSTEM_TABLE
```

The value of the `EFI_CAPSULE_IMAGE_SIZE`, which is the size in bytes of the capsule, is determined by the tools.

In order to use the `InitiateReset` flag, the `PersistAcrossReset` flag must also be set.

2.6.2 Capsule SET Statements

`SET` statements are used to define the values of PCD statements. These statements are positional in the FDF file. `SET` statements are set for the `[Capsule]` section, those set under the `[Capsule]` section header are global for all sub-sections within the `[Capsule]` section.

```
SET PcdName = VALUE
```

The following is an example of the `SET` statement.

```
SET gEfiMyTokenSpaceGuid.PcdSecStartLocalApicTimer = TRUE
```

The `VALUE` specified must match the PCD's datum type and must be the content data.

For a PCD that has a datum type of `VOID*`, the data can be a Unicode string, as in `L"text"`, a valid C data array (it must be either a C format GUID or a hex Byte array), as in `{0x20002000, 0x32FF, 0x00AA, {0xFF, 0xF0, 0x00, 0x00, 0x00, 0xF0, 0x00, 0x00, 0x00, 0xEF, 0x1A, 0x55}}`. Other PCD datum types are either boolean values or a hex value, as in `0x0000000F`, with a value that is consistent with the PCD's datum type.

2.6.3 Capsule Data

`EFI_CAPSULE_DATA` follows the `EFI_CAPSULE_HEADER` token definitions in the `[Capsule]` section or sub-sections. The content consists of one or more files, FD UiName, FV UiName or the following.

2.6.3.1 INF Statements

The INF statement syntax is common to the syntax used for `[FV]` sections. Refer to section 2.4.4 above.

2.6.3.2 FILE Statements

FILE statement syntax is common to the syntax used for `[FV]` sections. Refer to Section 2.5.5.

2.7 [VTF] Sections

The optional `[VTF]` sections specify information regarding the IPF Boot Strap File (BSF) or the IA32 Volume Top File (VTF). Both the `ARCH` and the `UiName` modifier fields are required. The `[VTF]` section terminates with either the start of another section, or the end of the file.

The `[VTF]` section modifier usage is shown below.

```
[VTF.ARCH.UiName]
```

Underneath the `[VTF]` section are specific statements defining information about the VTF file. EDK Bsf.inf files use two different sections, an `[OPTIONS]` section and a `[COMPONENTS]` section. For EDK II, the grammar of the `[VTF]` section statements defines these sections, rather than having separate sub-sections within the `[VTF]` section.

The format for statements within the section is illustrated below.

```
STATEMENT_NAME = Value
```

The component version number (`COMP_VER`) values are binary coded decimal (1 byte for the major number and 1 byte for the minor number). As a result, the maximum value is "99.99".

2.7.1 Options Statement

One and only one options statement, "IA32_RST_BIN", is permitted within any one `[VTF]` section. This value is a path and name of `IA32_BIOS` reset vector binary (16 byte) file. If needed, this binary can be put into the VTF file.

2.7.2 Component Statements

Within the section, a components sub-section starts with the "COMP_NAME" statement, and terminates with either the start of another sub-section, major section or the end of the file. Certain values for component statements are enumerated values or values that are within a given, specification defined, range.

Each of the component sections is used to complete a data structure, using the following sequence.

```
Name = Region,
       Type,
       Version,
       Checksum Flag,
       Path of Binary File,
       Path of the Symbol File,
       Preferred Size;
```

2.8 [Rule] Sections

The optional `[Rule]` sections in the FDF file are used for combining binary images, not for compiling code. Rules are use with the `[FV]` section's module INF type to define how an FFS file is created for a given INF file. The EDK II Build Specification defines the default rules that are implicitly used for creating FFS files. The implicit rules follow the PI Specification and UEFI Specification.

The `[Rule]` section of the FDF file is used to define custom rules, which may be applied to a given INF file listed in an `[FV]` section. This section is also used to define rules for module types that permit the user to define the content of the FFS file - when an FFS type is not specified by either PI or UEFI specifications.

The Rules can have multiple modifiers as shown below.

```
[Rule.ARCH.MODULE_TYPE.TEMPLATE_NAME]
```

If no `TEMPLATE_NAME` is given then the match is based on `ARCH` and `MODULE_TYPE` modifiers. `BINARY` is a reserved `TEMPLATE_NAME` as a default rule name for binary modules. The `TEMPLATE_NAME` must be unique to the `ARCH` and `MODULE_TYPE`. It is permissible to use the same `TEMPLATE_NAME` for two or more `[Rule]` sections if the `ARCH` or the `MODULE_TYPE` listed are different for each of the sections.

A `[Rule]` section is terminated by another section header or the end of file.

The content of the `[Rule]` section is based on the `FILE` and section grammar of the FV section. The difference is the `FILE` referenced in the `[RULE]` is a `MACRO`. The section grammar is extended to include an optional argument, `Optional`. The `Optional` argument is used to say a section is optional. That is to say, if it does not exist, then it is O.K.

Note: The `!include` statement is valid for any part of the `[Rule]` section, including an entire `[Rule]` section.

The generic form of the entries for leaf sections is:

```
<SectionType> <FileType> [Options] [{<Filename>} {<Extension>}]
```

When processing the FDF file, the following rules apply (in order):

1. If `<SectionType>` not defined or not a legal name, then error
2. If `<FileType>` not defined or not a legal name, then error
3. If `[FilePath/FileName]`, then: Add one section to FFS with a section type of `<SectionType>`
4. Else: Find all files defined by the INF file whose file type is `<FileType>` and add each one to the FFS with a section type of `<SectionType>` in alphabetical order. Add files defined in `[Sources]` followed by files defined in `[Binaries]`
5. If > 1 `UI` section in final FFS, then error
6. If > 1 `VER` section in final FFS, then error
7. If > 1 `PEI_DEPEX` section in final FFS, then error
8. If > 1 `DXE_DEPEX` section in final FFS, then error
9. If > 1 `SMM_DEPEX` section in final FFS, then error

If a rule specifies a file type, instead of specifying specific file names, the files that match the extension must be processed in alphabetical order.

Example

```
[Rule.Common.ACPITABLE]
  FILE FREEFORM = $(NAMED_GUID) {
    RAW ACPI Optional |.acpi
    RAW ASL  Optional |.aml
  }
```

Tools must add the processed .acpi files alphabetically, followed by the .aml files which must also be added alphabetically.

The file would contain:

`<SOF>a1.acpi, a2.acpi, b1.acpi, b2.acpi, a.aml, b.aml<EOF>`

where, start of file is `<SOF>` and end of file is `<EOF>`

Refer to the EDK II INF File Specification for a description of the `FileType` for binary files.

2.9 [OptionRom] Sections

The EDK II `[OptionRom]` sections allow for extending the FDF for processing of standalone legacy PCI Option ROM images or stand-alone UEFI PCI Option ROM images. A required modifier, DriverName, will specify where in the build's FV directory, the OptionROM file will be placed.

If the user is only creating PCI Option ROM images, then the `[FV]` and `[FD]` sections are not required. If an FD and FV sections exist, then the tools will create the FD image as well as the Option ROM image. If they are not in the FDF file, then they will only generate the Option ROM image.

Each `[OptionRom]` section terminates with either the start of another section, or the end of the file. The `[OptionRom]` section header usage is shown below.

`[OptionRom.ROMName]`

If more than architecture (for example, IA32 and EBC) for the driver is to be bundled in an option rom file, then more than one INF entry (specified by the USE option) can be used to include the other architecture.

Having different sections for the same option rom driver for different architectures is not permitted.

This is an optional section for platform images.

3 EDK II FDF FILE FORMAT

This section of the document describes the content of the EDK II FDF sections using an Extended Backus-Naur Form. Binary Only modules not listed in the platform DSC file, can be specified in the FDF file. There may also be modules listed in the DSC file that are not required in the FDF file. When a module listed in the DSC and is excluded from FDF file, then a UEFI-compliant binary will be generated, but the binary will not be put into any firmware volume.

3.1 General Rules

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

Note: Path and Filename elements within the FDF 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 FDF file are not casesensitive. The use of "..", "../" and "./" in paths and filenames is strictly prohibited.

- Multiple FDF files may exist in a directory.
- Text in section tags is case in-sensitive.
- A section terminates with either another section definition or the end of the file.
- Token statements terminate with the start of a new token statement, such as the start of a new section 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.
- Field separators for lines that contain more than one field are pipe "`|`" characters. 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 is used to uniquely identify the PCD.

- A line terminates with either an end of line character sequence or a comment.
- When processing numeric values, either integer or hex, leading zeros specified in the entry may be ignored. For example, 0x000000000000000000000001 can be a valid value for a `UINT8` data type, as the actual value is 1.

3.1.1 Line-Extension 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 FDF 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 filenames

Note that for specifying the path for a file name, if the path value starts with a dollar sign "`$`" character, either a local MACRO or system environment variable is being specified. If the path value starts with one of "letter:\", "/", "\" or "\" the path must be a fully qualified URI location. If it does not, the specified path is relative to the `WORKSPACE` (or `PACKAGES_PATH`) environment variable.

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

For all FDF files, the specified directory path must use the forward slash character for separating directories. For example, MdePkg/Include/ is Valid.

Note: If the platform integrator is working on a Microsoft Windows* environment and will not be working on a non-windows platform, then the DOS-style directory separator can be used. The forward slash Unix-style directory separator is mandatory for distributions where the build environment is unknown.

Unless otherwise noted, all file names and paths are relative the system environment variable, `WORKSPACE` (or relative to a directory listed in the `PACKAGES_PATH` system environment variable). A directory name that starts with a word is assumed by the build tools to be located in the `WORKSPACE` directory (or a directory listed in the `PACKAGES_PATH` system environment variable). Search paths for locating the files are `WORKSPACE` then the ordered list (left to right) of directories listed in the optional `PACKAGES_PATH` environment variable.

Each module may have one or more INF files that can be used by tools to generate images. Specifically, the EDK Compatibility Package may 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.

3.2 FDF Definition

The FDF definitions define the final properties for a flash image - PCD settings in this file override any other PCD settings that may have been set in the platform description (DSC) file. The `[Defines]` section, when specified, must appear before any other section except the header. (The header, when specified, is always the first section of an FDF file.) The remaining sections may be specified in any order within the FDF file.

Summary

The EDK II Flash Description (FDF) file has the following format (using the EBNF).

```
<EDK_II_FDF> ::= [<Header>]
                 [<Defines>]
                 <FD>*
                 <FV>*
                 <Capsule>*
                 <FmpPayload>*
                 <VTF>*
                 <Rules>*
                 <OptionRom>*
                 <UserExtensions>*
```

Note: Assignments set as command-line arguments to the parsing tools take precedence over all assignments defined in the FDF file. If a variable/value assignment is specified on the build tool's command-line, that value will override any variable/value assignment defined in the FDF file.

Note: Conditional statements may be used anywhere within the FDF file, with the ability to group any item within a section as well as entire sections.

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 white space 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)* Alphanumeric characters. white
                          space characters are not permitted.
<FileSep>             ::= "/"
<Extension>           ::= (a-zA-Z0-9_-)+ One or more alphanumeric characters.
<File>                ::= <Word> ["." <Extension>]
<PATH>                ::= [<MACROVAL> <FileSep>] <RelativePath>
<RelativePath>        ::= <DirName> [<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 - 0x5B)} {(0x5D - 0x7E)}
                            {<EscapeSequence>}]*
<DblQuote>              ::= 0x22
<EscapeSequence>        ::= "\" {"n"} {"t"} {"f"} {"r"} {"b"} {"0"}
                            {"\"} {<DblQuote>}
<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>
<CString>               ::= ["L"] <QuotedString>
<NormalizedString>      ::= <DblQuote> [{<Word>} {<Space>}]+ <DblQuote>
<GlobalComment>         ::= <WS> "#" [<AsciiString>] <EOL>+
<Comment>               ::= "#" <AsciiString> <EOL>+
<UnicodeString>         ::= "L" <QuotedString>
<HexDigit>              ::= (a-fA-F0-9)
<HexByte>               ::= {"0x"} {"0X"} [<HexDigit>] <HexDigit>
<HexNumber>             ::= {"0x"} {"0X"} <HexDigit>+
<HexVersion>            ::= "0x" [0]* <Major> <Minor>
<Major>                 ::= <HexDigit>? <HexDigit>? <HexDigit>?
                            <HexDigit>
<Minor>                 ::= <HexDigit> <HexDigit> <HexDigit> <HexDigit>
<DecimalVersion>        ::= {"0"} {(1-9) [(0-9)]*} ["." (0-9)+]
<VersionVal>            ::= {<HexVersion>} {(0-9)+ "." (0-99)}
<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> <HexByte>
                            [ <Cs> <HexByte> [<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"}
<BoolType>             ::= {<TRUE>} {<FALSE>}
<MACRO>                ::= (A-Z)(A-Z0-9_)*
<MACROVAL>             ::= "$(" <MACRO> ")"
<PcdName>              ::= <TokenSpaceGuidCName> "." <PcdCName>
<PcdCName>             ::= <CName>
<TokenSpaceGuidCName>  ::= <CName>
<PCDVAL>               ::= "PCD(" <PcdName> ")"
<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)
<NumValUint8>          ::= {<ShortNum>} {<UINT8>}
<NumValUint16>         ::= {<IntNum>} {<UINT16>}
<NumValUint32>         ::= {<LongNum>} {<UINT32>}
<NumValUint64>         ::= {<LongLongNum>} {<UINT64>}
<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>
<MembershipExpression> ::= {<TargetExpress>} {<ArchExpress>} {<ToolExpress>}
<NotOp>                ::= <UnaryOperator> <MTS>
<InOp>                 ::= <MTS> [<NotOp>] {"IN"} {"in"} <MTS>
<TargetExress>         ::= (A-Z) (A-Z0-9)* <InOp> "$(TARGET)"
<ArchExpress>          ::= <DblQuote> <Arch> <DblQuote> <InOp> "$(ARCH)"
<Arch>                 ::= {"IA32"} {"X64"} {"IPF"} {"EBC"} {<OA>}
<ToolExpress>          ::= (A-Z) (a-zA-Z0-9)* <InOp> "$(TOOL_CHAIN_TAG)"
<Boolean>              ::= {<BoolType>} {<Expression>}
<EOL>                  ::= <TS> 0x0D 0x0A
<OA>                   ::= (a-zA-Z)(a-zA-Z0-9)*
<arch>                 ::= {"IA32"} {"X64"} {"IPF"} {"EBC"} {<OA>}
                           {"common"}
<FvAlignmentValues>    ::= {"1"} {"2"} {"4"} {"8"} {"16"} {"32"}
                           {"64"} {"128"}{"256"} {"512"} {"1K"} {"2K"}
                           {"4K"} {"8K"} {"16K"} {"32K"} {"64K"}
                           {"128K"} {"256K"} {"512K"} {"1M"} {"2M"} {"4M"}
                           {"8M"} {"16M"} {"32M"} {"64M"}
                           {"128M"} {"256M"} {"512M"} {"1G"} {"2G"}
<FfsAlignmentValues>   ::= {"Auto"} {"8"} {"16"} {"32"} {"64"} {"128"}
                           {"512"} {"1K"} {"4K"} {"32K"} {"64K"}
```

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 FDF 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.

Parameter Definitions

Expression

Refer to the EDK II Expression Syntax Specification for more information.

""UnicodeString

When the `<UnicodeString>` element (these characters are string literals as defined by the C99 specification: 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-2LE 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.

FileSep

FileSep refers to either the back slash "\" or forward slash "/" characters that are used to separate directory names. All EDK II FDF files must use the "/" forward slash character when specifying the directory portion of a filename. Microsoft operating systems, that normally use a back slash character for separating directory names, will interpret the forward slash character correctly.

TokenSpaceGuidCName

A word that is a valid C variable that specifies the name space for a particular PCD.

PcdCName

A word that is a valid C variable that specifies the name of the token number which a member of the name space specified by the TokenSpaceGuidCName.

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.

EOL

The DOS End Of Line: "0x0D 0x0A" character sequence must be used for all EDK II meta-data 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 or may be defined on a command line of a parsing tool.

Define statements are processed according to the following precedence.

Highest Priority

1. Command-line option -D MACRO=Value
2. Most Recent in file
3. Macros defined in the FDF file's `[Defines]` section
4. Macros defined in the DSC file's `[Defines]` section Lowest Priority

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 may not be referenced before they are defined.

Macros may be inherited from the DSC file specifying this FDF file.

All content for a macro statement must appear on a single line.

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>           ::= {<Number>} {<BoolType>} {<CFormatGUID>} {<PATH>}
                      {<CString>} {<UnicodeString>} {<CArray>}
                      {<Expression>} {<CFlags>} {<Filename>}
```

Restrictions

System Environment Variables

System environment variable may not be re-defined in this file. System environment variables cannot be overridden by the build system tools.

Parameters

Expression

C-style expression using C relational, equality and logical numeric and bitwise operators and/or arithmetic and bitwise operators that evaluate to a value (for PCDs, the value must match the Datum Type of the PCD). Precedence and associativity follow C standards. Along with absolute values, macro names and PCDs may be used within an expression. For both macro names and PCDs, the element must be previously defined before it can be used.

VALUE

The `<EQ> <VALUE>` is optional, and if not included, uses a default of TRUE.

Note: Some MACRO and PCD values may be defined in the Platform DSC file.

Examples:

```
DEFINE SECCORE    = MyPlatform/SecCore
DEFINE GEN_SKU    = MyPlatform/GenPei
DEFINE SKU1       = MyPlatform/Sku1/Pei
DEFINE FLASH_SIZE = 0x00280000
DEFINE MY_MACRO
```

EDK_GLOBAL

The macro names defined using the `EDK_GLOBAL` statement in the DSC file may be used in paths, value fields and conditional statements in this file. The `EDK_GLOBAL` statement itself, cannot be specified in this file.

3.2.3 Conditional Directive Blocks

Use of conditional directive blocks is optional.

Summary

Conditional statements may appear anywhere within the file. Conditional directive blocks can be nested. Conditional directive processing must emulate a C pre-processor.

- All conditional directives can use MACRO, FixedAtBuild or FeatureFlag PCD values, which must evaluate to either '`True`' or '`False`.'
- Directives must be the only statement on a line.
- String evaluations are permitted, and are case sensitive; the two string values must be an exact match to evaluate to '`True`'.
- The expression immediately following the '`!if`' statement controls whether the content after the line is processed or not. `TRUE` is any non-zero and/or non-NULL value other than zero.
- Each '`!if`' within the source code must be matched with a closing '`!endif`'.
- Zero or more `!elseif` statements are permitted; only one `!else` statement is permitted.
- Conditional directive blocks may be nested.
- Directives can be used to encapsulate entire sections or elements within a single section, such that they do not break the integrity of the section definitions.
- Directives are in-fix expressions that are evaluated left to right; content within parenthesis is evaluated before the outer statements are evaluated. Use of parenthesis is recommended to remove ambiguity.
- The values of the FixedAtBuild and FeatureFlag PCDs used in the conditional statements must be set in the `[PcdsFixedAtBuild]` or `[PcdsFeatureFlag]` section(s) of the DSC file or included in `SET` statements.
- Default values from the DEC file are not permitted. Values used in directive statement in the FDF files must be define in either the DSC file or the FDF file.

Conditional directives may appear before a Macro, FixedAtBuild or FeatureFlag PCD has been defined. Therefore, the reference build tools may be required to perform two passes on this file to resolve all directive statements:

1. Obtain the values of the Macros, FixedAtBuild or FeatureFlag PCDs used for theconditional directives
2. Evaluate the conditional statements for inclusion in the build.

If the value of a FixedAtBuild or FeatureFlag PCD cannot be determined, the build will break.

If the value of a FixedAtBuild or FeatureFlag PCD used in a conditional directive cannot be determined during the first pass, the build should break. Macros, FixedAtBuild and FeatureFlag PCDs used in conditional statements in the first pass must not be located within conditional directives. It is permissible to have a Macro that is undefined after the first pass. It is permissible to have macros that are undefined used in !ifdef and `!ifndef` statements. FixedAtBuild or FeatureFlag PCDs in the first pass must not be located within a conditional directive.

Macro and PCD values may be inherited from the DSC file.

Note: PCDs, used in conditional directives, must be defined and the value set in either the FDF or DSC file in order to be used in a conditional statement; values from INF or DEC files are not permitted.

Prototype

```
<Conditional>         ::= <IfStatement>
                          <ElseIfConditional>*
                          [<ElseConditional>]
                          <TS> "!endif" <EOL>
<IfStatement>         ::= {<TS> "!if" <MTS> <ExpressionStatement>}
                          {<TS> "!ifdef" <MTS> <MACRO> <EOL>}
                          {<TS> "!ifndef" <MTS> <MACRO> <EOL>}
                          <Statements>*
<Statements>          ::= {<Statements>} {<Conditional>}
<ElseIfConditional>   ::= <TS> "!elseif" <MTS> <ExpressionStatement>
                          <EOL>
                          <Statements>*
<ElseConditional>     ::= <TS> "!else" <EOL> <Statements>*
<ExpressionStatement> ::= <Expression> <EOL>
```

Restrictions

MACRO and PCD Values

When a MACRO is used in conditional directives `!if` or `!elseif`, the `<MACROVAL>` - $(MACRO) - format is used. When a PCD is used in a conditional directive (or in an expression) the `<PCDVAL>` - $(PcdName) - format is used.

Number values

For Numeric expressions, numbers must not be encapsulated by double quotation marks

Strings

Strings in `<PCDVAL>` elements must be `NULL` terminated. The `NULL` character is not part of the string that is tested. All other string comparisons do not include the double quotation marks encapsulating the string. If the string is "myapple", the only characters that would be tested are myapple. When using strings in the expression statements, there must be a comparison operator.

Parameters

Statements

Any valid section, section statement or set of statements defined in this specification may be within the scope of the conditional statements. The encapsulated statements must not break the integrity of the file. All statements within the encapsulation must end with `<EOL>`.

MACRO Usage in Expression Statements

If a MACRO is used in expression statements, the MACRO must be encapsulated between "$(" and ")" character sets (matching C format). String comparisons are case sensitive and must exactly equal, character for character - leading and trailing white space characters included.

PcdFeatureFlag

The FeatureFlag PCD is a boolean PCD that is set to either `True` (1) or `False` (0). The PCD datum type for a FeatureFlag PCD is always `BOOLEAN`. It may be used in a logical expression.

FixedPcdName

A FixedAtBuild PCD will have a set value at build time, and that cannot be modified in the binary image, nor modified at runtime. For directives, the PCD datum type is limited to `UINT8`, `UINT16`, `UINT32`, `UINT64`, `UINTN` and `BOOLEAN`. Using a FixedAtBuild PCD that has a datum type of `VOID*` is limited to text-based comparisons in directives. Using a PCD that has a value of a byte array is not permitted. FixedAtBuild PCDs may be used in a logical expression.

Numeric Expression

This is a test of numbers, using relational or equality operators, that evaluates to `TRUE` or `FALSE`

Logical Expression

This is a test where the expression, MACRO value or PCD value (include `<MACROVAL>` or `<PCDVAL>` used in an expression) must evaluate to either `TRUE` (1) or `FALSE` (0), no operators are required, however logical operators, as well full expressions can be used. (expressions that do not evaluate to `TRUE` or `FALSE` must break the build).

String Expressions

The strings must be exactly identical in order to match. Literal strings must be encapsulated by double quotation marks. There must be a comparison operator between two strings (using a string without an operator is not permitted). Also permitted are the membership expressions, for architectures, targets and tool chain tag names.

All Expression

C-style expression using C relational, equality and logical numeric and bitwise operators that evaluate to either `TRUE` (1) or `FALSE` (0). Values other than zero or one are invalid. Precedence and associativity follow C standards. Along with absolute values, macro names and PCDs may be used within an expression. For both macro names and PCDs, the element must be previously defined before it can be used. A new operator, "in" is also permitted for testing membership of an item in a list of one or more items.

Example

```
!if $(MyPlatformTspGuid.IPF_VERSION_1) && NOT $(MyPlatformTspGuid.IPF_VERSION_2)
  [VTF.IPF.MyBsf]
    !ifdef IA32RESET
      # IPF_VERSION is 1 and IA32RESET defined
      IA32_RST_BIN           = IA32_RST.BIN
    !endif
```

```
    COMP_NAME = PAL_A
    COMP_LOC  = MyVtfVF | F
    COMP_TYPE = 0xF
    COMP_VER  = 7.01
    COMP_CS   = 1
    !if ($(PROCESSOR_NAME) == "M1")
      COMP_BIN = M1PalCode/PAL_A_M1.BIN
      COMP_SYM = M1PalCode/PAL_A_M1.SYM
    !elseif ($(PROCESSOR_NAME) == "M2")
      COMP_BIN = M2PalCode/PAL_A_M2.BIN
      COMP_SYM = M2PalCode/PAL_A_M2.SYM
    !else
      COMP_BIN = GenPal/PAL_A_GEN.bin
      COMP_SYM = GenPal/PAL_A_GEN.sym
    !endif
    COMP_SIZE = -
!elseif $(MyPlatformTspGuid.IPF_VERSION_2)
  [VTF.IPF.MyBsf]
    !ifdef IA32RESET
      IA32_RST_BIN = IA32_RST.BIN
    !endif
    COMP_NAME = PAL_A
    COMP_LOC  = MyVtfFv | F
    COMP_TYPE = 0xF
    COMP_VER  = 7.01
    COMP_CS   = 1
    COMP_BIN  = GenPal/PAL_A_GEN.bin
    COMP_SYM  = GenPal/PAL_A_GEN.sym
    COMP_SIZE = -
    COMP_NAME = PAL_B
    COMP_LOC  = MyVtfFv | S
    COMP_TYPE = 0x01
    COMP_VER  = -
    COMP_CS   = 1
    COMP_BIN  = GenPal/PAL_B_GEN.bin
    COMP_SYM  = GenPal/PAL_B_GEN.sym
    COMP_SIZE = -
!else
  [VTF.X64.MyVtf]
      IA32_RST_BIN = IA32_RST.BIN
!endif
!ifndef MY_MACRO
DEFINE MY_MACRO
!endif
```

3.2.4 !include Statements

Use of this statement is optional.

Summary

Defines the `!include` statement in FDF files. This statement is used to include, at the statement's line, a file which is processed at that point, as though the text of the included file was actually in the FDF file. Statements in the `!include` file(s) are additions to the FDF file, and do not replace information in the FDF file. The included file's content must match the content of the section that the `!include` statement resides, or it may contain completely new sections of the same section type. If the included file contains new sections, then the section being processed in the Platform FDF file is considered to have been terminated.

If the `<Filename>` contains "$" characters, then macros defined in the DSC file, FDF file, and the system environment variables, `$(WORKSPACE)`, `$(EDK_SOURCE)`, `$(EFI_SOURCE)`, and `$(ECP_SOURCE)` are substituted into `<Filename>`.

The tools look for `<Filename>` relative to the directory the FDF file resides. If the file is not found, and the directory containing this FDF file is not the same directory as the directory containing the DSC file, the tools must attempt to locate the file relaitive to the directory that contains the DSC file.

If none of these methods find the file, and a directory separator is in `<Filename>`, the tools attempt to find the file in a WORKSPACE (or a directory listed in the PACKAGES_PATH) relative path. If the file cannot be found, the build system must exit with an appropriate error message.

Prototype

```
<IncludeStatement> ::= <TS> "!include" <MTS> <Filename> <EOL>
```

Example (EDK II FDF)

```
!include myPlatform/NvRamDefs.txt
!include myFeatures.mak
!include $(WORKSPACE)/PackageDir/Features.dsc
!include $(MACRO1)/AnotherDir/$(MACRO2)/Features.dsc
```

3.3 Header Section

This is an optional 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:

```
## @file Nt32.fdf
# Abstract
#
# Description
#
# Copyright
#
# License
#
##
```

Prototype

```
<Header>      :: = <Comment>*
                   "##" <Space> [<Space>] @file" <EOL>
                   [<Abstract>]
                   [<Description>]
                   <Copyright>+
                   "#" <EOL>
                   <License>+
                   "##" <EOL>
<Filename>    :: = <Word> "." <Extension>
<Abstract>    :: = "#" <MTS> <AsciiString> <EOL>
                   ["#" <EOL>]
<Description> :: = ["#" <MTS> <AsciiString> <EOL>]+
                   ["#" <EOL>]
<Copyright>   ::= "#" <MTS> <CopyName> <Date> "," <CompInfo> <EOL>
<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>"]
<License>     ::= ["#" <MTS> <AsciiString> <EOL>]+
                  ["#" <EOL>]
```

Example

```
## @file
# Emulation Platform Pseudo Flash Part
#
# The Emulation Platform can be used to debug individual modules,
# prior to creating a real platform. This also provides an example for
# how to create an FDF file.
#
# Copyright (c) 2006 - 2008, NoSuch 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 an optional section. This section, if present, must be the first section following comment blocks at the beginning of the file.

Summary

This section describes the defines section content in the FDF files. This file can be created by a developer and is an input to the EDK II build tool parsing utilities. Elements may appear in any order within this section.

The code for this version of the FDF specification is "0x0001001B". New versions of this specification must increment the minor (001B) portion of the specification code for backward-compatible changes, and increment the major specification number for non-backward-compatible changes.

This revision of the specification adds new features. Any FDF file that uses these new features must use the value `0x0001001B` in the `FDF_SPECIFICATION` statement. Older FDF files that do not use these new feature do not need to update the value.

Conditional statements may be used anywhere within this section.

Prototype

```
<Defines>    ::= "[Defines]" <EOL>
                 [<TS> "FDF_SPECIFICATION" <Eq> <SpecVer> <EOL>]
                 [<TS> "FDF_VERSION" <Eq> <DecimalVersion> <EOL>] <DefStmts>*
<DefStmts>   ::= {<MacroDefinition>} {<SetStmts>} {<IncludeStatement>}
<UiNameType> ::= <AsciiString>
<SpecVer>    ::= {<HexVersion>} {(0-9)+ "." (0-9)+}
<SetStmts>   ::= <TS> "SET" <MTS> <PcdName> <Eq> [<VALUE>] <EOL>
<VALUE>      ::= {<Number>} {<Boolean>} {<GUID>} {<CArray>}
                 {<CString>} {<UnicodeString>} {<Expression>}
```

Parameters

Expression

Refer to the EDK II Expression Syntax Specification for more information.

FDF_VERSION

The version number for this flash definition; the value is not used by build tools, but the version element is provided for user tracking capabilities that may be used by other user interface tools.

FDF_SPECIFICATION

For this specification, the version value is 0x0001001B. Tools that process this version of the FDF file can successfully process earlier versions of the FDF files (this is a backward compatible update). If an FDF file with an earlier version of the `FDF_SPECIFICATION` is modified to use a feature added in the 1.27 version of this specification must be updated to 0x0001001B. There is no requirement to change existing entries if no other content changes. This value may also be specified as decimal value, such as 1.27.

PcdNames

PCDs defined in this section take precedence over PCD values specified in other meta-data files. Note that PCDs defined via the SET statements in sub-sections of the document can override the values set here as well as in other EDK II meta-data files.

Example

```
[Defines]
  FDF_SPECIFICATION                          = 0x0001001B
  DEFINE BIG_STUFF                           = False
  SET gEfiMyPlatformTokenSpaceGuid.MyUsbFlag = True
```

3.5 [FD] Sections

This is a required section for platform flash parts, and is not required for simple Option ROM creation.

Summary

This describes the `[FD]` section tag, which is required in all FDF files. This file is created by the platform integrator and, along with the platform DSC file, is an input to the parsing utilities.

All FD files will be created in the `$(OUTPUT_DIRECTORY)/$(TARGET)_$(TAGNAME)/FV` directory using the values from the individual instance of the build tools. (Build tools get these values after parsing DSC, INF, `target.txt`, `tools_def.txt` files and command line options).

Conditional statements may be used anywhere within this section.

Prototype

```
<FD>              ::= "[FD" [<FdUiName>] "]" <EOL>
                      <TokenStatements>
                      <FdStatements>*
<FdStatements>    ::= {<GlobalStmts>} {<SetStatements>}
                      {<RegionLayout>}
<GlobalStmts>     ::= {<MacroDefinition>} {<IncludeStatement>}
<FdUiName>        ::= "." (a-zA-Z)(a-zA-Z0-9_)*
<TokenStatements> ::= <TS> "BaseAddress" <Eq> <UINT64> [<SetPcd>]
                      <EOL>
                      <TS> "Size" <Eq> <UINT64> [<SetPcd>] <EOL>
                      <TS> "ErasePolarity" <Eq> {"0"} {"1"} <EOL>
                      <BlockStatements>+
<SetPcd>          ::= <FS> <PcdName>
<BlockStatements> ::= <TS> "BlockSize" <Eq> <UINT32> [<SetPcd>]
                      <EOL>
                      [<TS> "NumBlocks" <Eq> <UINT32> <EOL>]
<SetStatements>   ::= <TS> "SET" <PcdName> <Eq> <VALUE> <EOL>
<VALUE>           ::= {<Number>} {<Boolean>} {<GUID>} {<CArray>}
                      {<CString>} {<UnicodeString>} {<Expression>}
<RegionLayout>    ::= <TS> <Offset> <FS> <Size> <EOL>
                      [<TS> <PcdOffset> [<FS> <PcdSize>] <EOL>]
                      [<RegionType>]
<Offset>          ::= {<HexNumber>} {<Expression>}
<Size>            ::= {<HexNumber>} {<Expression>}
<RegionType>      ::= {<FvType>} {<FileType>} {<CapsuleRegion>}
                      {<DataType>} {<InfRegion>}
<InfRegion>       ::= <TS> "INF" <MTS> [<InfOptions>] <InfFile> <EOL>
<InfOptions>      ::= [<Use>] [<Rule>] [<SetVer>] [<SetUi>]
<Use>             ::= "USE" <Eq> <TargetArch> <MTS>
<TargetArch>      ::= <arch>
<Rule>            ::= "RuleOverride" <Eq> {<Word>} {"BINARY"} <MTS>
<SetVer>          ::= "VERSION" <Eq> <CString> <MTS>
<SetUi>           ::= "UI" <Eq> <CString> <MTS>
<InfFile>         ::= <PATH> <Word> ".inf" [<FS> <RelocFlags>]
<RelocFlags>      ::= {"RELOCS_STRIPPED"} {"RELOCS_RETAINED"}
<CapsuleRegion>   ::= <TS> "CAPSULE" <Eq> UiCapsuleName <EOL>
<PcdOffset>       ::= <PcdName>
<PcdSize>         ::= <PcdName>
<FvType>          ::= <TS> "FV" <Eq> <FvNameOrFilename> <EOL>
<FileType>        ::= <TS> "FILE" <Eq> <BinaryFile> <EOL>
<DataType>        ::= <TS> "DATA" <Eq>
                      "{" [<EOL>]
                      [<DataContent>]
                      <TS> "}" <EOL>
<DataContent>     ::= <TS> {<RawData>} {<CFormatGUID>} {<UINT8z>} [<EOL>]
<BinaryFile>      ::= [<Location>] <File>
<Location>        ::= {<BuildId>} {<PATH>}
<BuildId>         ::= <VarLoc> {"FV"} {<arch>} "/"
```

```
<VarLoc>           ::= "$(OUTPUT_DIRECTORY)/$(TARGET)_($TOOL_CHAIN_TAG)/"
<FvNameOrFilename> ::= {<FvUiName>} {<FvFilename>}
<FvUiName>         ::= {<Word>} {"common"}
<FvFilename>       ::= [<PATH>] <Word> "." {"fv"} {"Fv"} {"FV"}
```

Restrictions

For the `FvFilename`, the PATH is relative to the EDK II environment variable `$(WORKSPACE)` or relative to a path listed in the `PACKAGES_PATH` environment variable. If the path is not specified, the `PATH` defaults to the following location, where `$(OUTPUT_DIRECTORY)` is specified in the EDK II Platform DSC file's `[Defines]` section. If a path is not present, or the ".fv" file extensions do not appear in the value, the build system will use a filename based on the UiFvFilename specified in the FDF file:

`$(OUTPUT_DIRECTORY)/$(TARGET)_$(TOOL_CHAIN_TAG)/FV`

For the Binary File, the `PATH` must be relative to the EDK II environment variable: `$(WORKSPACE)` or relative to a path listed in the `PACKAGES_PATH` environment variable. If not specified, the `PATH` defaults to the directory location of the EDK II Platform DSC file. If not found, the tools must test the directory location of this FDF file, if different from the directory containing the Platform's DSC file.

If a GUID is used (either the C format or Registry Format) in the data content, tools will be required to process the GUID into a byte array.

Raw Data arrays in FDF files are always listed as byte arrays, using little-endian format. Numbers listed in the DataContent section must be zero filled, in order to determine the size of the element. For example, a `UINT16` value of 1 must be specified as `0x0001`.

Parameters

FdUiName

This name is used by the *GenFds* tool to generate FD image files. If not present, only one FD section is allowed and the *GenFds* tool will use the name of the active platform as the name of the FD image.

UiCapsuleName

The `UiCapsuleName` must be specified in a `[Capsule]` section header defined in this the file.

FvUiName

The `FvUiName` must be specified in a `[FV]` section header defined in this the file.

PcdValue

The PCD Value may be a specific numeric value, an array of numeric bytes, a GUID, a quoted string, an L quoted string (representing a unicode string), an arithmetic expression, a logic expression or a macro from a previously defined macro statement.

Expression

Refer to the EDK II Expression Syntax Specification for more information.

Location

For BINARY ONLY files, the location specified in the FILE element of this section must be relative to the directory identified by the WORKSPACE or relative to a path listed in the PACKAGES_PATH system environment variable.

Example

```
[FD.FdMain]
BaseAddress   = 0xFFF00000 |gEfiMyPlatformTokenSpaceGuid.PcdFlashAreaBaseAddress
```

```
Size          = 0x102000
ErasePolarity = 1
BlockSize     = 0x10000
NumBlocks     = 16
BlockSize     = 0x1000
NumBlocks     = 2

# Offset:Size
0x000000|0x0C0000
gEfiMyPlatformTokenSpaceGuid.PcdFlashFvMainBase|gEfiMyPlatformTokenSpaceGuid.PcdFlashFvMainSize
FV   = FvMain

0x0C0000|0x00A000
gEfiMyPlatformTokenSpaceGuid.PcdFlashNvStorageBase|gEfiMyPlatformTokenSpaceGuid.PcdFlashNvStorageSize
Data = { # Variable Store
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x8D, 0x2B, 0xF1, 0xFF, 0x96, 0x76, 0x8B, 0x4C,
0xA9, 0x85, 0x27, 0x47, 0x07, 0x5B, 0x4F, 0x50,
0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00,
0x5F, 0x46, 0x56, 0x48, 0xFF, 0x8E, 0xFF, 0xFF,
0x5A, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
}

0x0CA000 | 0x002000
gEfiCpuTokenSpaceGuid.PcdCpuMicrocodePatchAddress|gEfiCpuTokenSpaceGuid.PcdCpuMicrocodePatchSize
FILE = FV/Microcode.bin

0x0CC000 | 0x002000 # Event Log
gEfiMyPlatformTokenSpaceGuid.PcdFlashNvStorageEventLogBase|gEfiMyPlatformTokenSpaceGuid.PcdFlashNvStorageEventLogSize

0x0CE000 | 0x002000 # FTW Working Area
gEfiMyPlatformTokenSpaceGuid.PcdFlashNvStorageFtwWorkingBase|gEfiMyPlatformTokenSpaceGuid.PcdFlashNvStorageFtwWorkingSize
Data = {
0x8D, 0x2B, 0xF1, 0xFF, 0x96, 0x76, 0x8B, 0x4C,
0xA9, 0x85, 0x27, 0x47, 0x07, 0x5B, 0x4F, 0x50,
0x85, 0xAE, 0x2D, 0xBF, 0xFE, 0xFF, 0xFF, 0xFF,
0xE4, 0x1F, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF
}

0x0D0000 | 0x010000 # FTW Spare Block
gEfiMyPlatformTokenSpaceGuid.PcdFlashNvStorageFtwSpareBase|gEfiMyPlatformTokenSpaceGuid.PcdFlashNvStorageFtwSpareSize

0x0E0000 | 0x020000
gEfiMyPlatformTokenSpaceGuid.PcdFlashFvRecoveryBase|gEfiMyPlatformTokenSpaceGuid.PcdFlashFvRecoverySize
FV = FV/FvRecovery.fv
```

3.6 [FV] Sections

One or more of theses sections is required for platform images, and not required for simple Option ROM generation.

Summary

This describes the `[FV]` section tag, which is required in all platform FDF files. This file is created by the platform integrator and is an input to the one or more parsing utilities.

Note that `FvAttributes`, listed below, may be set via PCD settings. Setting the attribute via PCD takes precedence over the `FvAttributes` settings in this FDF file. If the `FvAttribute` is not set by either a PCD or an `FvAttribute` line in the FDF file, then the default value is `FALSE` - the corresponding bit in the `EFI_FV_ATTRIBUTE` of the `EFI_FIRMWARE_VOLUME_PROTOCOL.GetVolumeAttributes()` is set per the `FvAttributesSet` or `FvAttributesClear`; items specified in `FvAttributesSet` are default "TRUE", while items in `FvAttributesClear` are default "FALSE".

If FV files are created, they will be created in the `$(OUTPUT_DIRECTORY)/$(TARGET)_$(TAGNAME)/FV` directory using the values from the individual instance of the build tools. (Build tools get these values after parsing DSC, INF, `target.txt`, `tools_def.txt` files and command line options).

Conditional statements may be used anywhere within this section.

Prototype

```
<FV>               ::= "[FV" <FvUiName> "]" <FvStmts>*
<FvStmts>          ::= {<ExtendedFvEntry>} {<FvStatements>} {<GlobalStmts>}
<FvUiName>         ::= "." (a-zA-Z)(a-zA-Z0-9_)*
<ExtendedFvEntry>  ::= <TS> "FV_EXT_ENTRY_TYPE" <MTS> "TYPE" <Eq>
                       <Hex4>
                       "{" [<EOL>]
                       {<TS> "FILE" <Eq> <BinaryFile> [<EOL>]}
                       {<TS> "DATA" <Eq> "{" <DataContent> "}" }
                       "}" <EOL>
<DataContent>      ::= <TS> {<RawData>} {<CFormatGUID>} {<UINT8z>}
                       {<UINT16z>} {<UINT32z>} {<UINT64z>} [<EOL>]
<BinaryFile>       ::= <PATH> <File> [<EOL>]
<FvStatements>     ::= [<BlockStatements>]
                       [<FvAlignment>]
                       [<FvAttributes>]
                       [<FileSystemGuid>]
                       [<FvNameGuid>]
                       [<FvNameString>]
                       [<PeiAprioriSection>]
                       [<DxeAprioriSection>]
                       <InfStatements>*
                       <FileStatements>*
<GlobalStmts>      ::= {<MacroDefinition>} {<IncludeStatement>}
<BlockStatements>  ::= <FixedBlocks>
<FixedBlocks>      ::= [<TS> "BlockSize" <Eq> <UINT32> <EOL>]
                       [<TS> "NumBlocks" <Eq> <UINT32> <EOL>]
<SetStatements>    ::= <TS> "SET" <MTS> <PcdName> <Eq> <VALUE> <EOL>
<VALUE>            ::= {<Number>} {<Boolean>} {<GUID>} {<CArray>}
                       {<CString>} {<UnicodeString>} {<Expression>}
<FvAlignment>      ::= [<TS> "FvBaseAddress" <Eq> <UINT64> <EOL>]
                       [<TS> "FvForceRebase" <Eq> <TrueFalse> <EOL>]
                       <TS> "FvAlignment" <Eq>
                       <FvAlignmentValues> <EOL>
<FvAttributes>     ::= [<TS> "MEMORY_MAPPED" <Eq> <TrueFalse> <EOL>]
                       [<TS> "LOCK_CAP" <Eq> <TrueFalse> <EOL>]
                       [<TS> "LOCK_STATUS" <Eq> <TrueFalse> <EOL>]
                       [<TS> "WRITE_LOCK_CAP" <Eq> <TrueFalse> <EOL>]
                       [<TS> "WRITE_LOCK_STATUS" <Eq> <TrueFalse> <EOL>]
```

```
                        [<TS> "WRITE_ENABLED_CAP" <Eq> <TrueFalse> <EOL>]
                        [<TS> "WRITE_DISABLED_CAP" <Eq> <TrueFalse> <EOL>]
                        [<TS> "WRITE_STATUS" <Eq> <TrueFalse> <EOL>]
                        [<TS> "STICKY_WRITE" <Eq> <TrueFalse> <EOL>]
                        [<TS> "WRITE_POLICY_RELIABLE" <Eq> <TrueFalse> <EOL>]
                        [<TS> "READ_LOCK_CAP" <Eq> <TrueFalse> <EOL>]
                        [<TS> "READ_LOCK_STATUS" <Eq> <TrueFalse> <EOL>]
                        [<TS> "READ_ENABLED_CAP" <Eq> <TrueFalse> <EOL>]
                        [<TS> "READ_DISABLED_CAP" <Eq> <TrueFalse> <EOL>]
                        [<TS> "READ_STATUS" <Eq> <TrueFalse> <EOL>]
                        [<TS> "ERASE_POLARITY" <Eq> {"0"} {"1"} <EOL>]
<FileSystemGuid>    ::= <TS> "FileSystemGuid" <Eq> <NamedGuid> <EOL>
<FvNameGuid>        ::= <TS> "FvNameGuid" <Eq> <NamedGuid> <EOL>
<FvNameString>      ::= <TS> "FvNameString" <Eq> <TrueFalse> <EOL>
<PeiAprioriSection> ::= <TS> "APRIORI" <MTS> "PEI" <MTS>
                        "{" <EOL>
                        <MacroDefinition>*
                        <InfStatements>*
                        <FileStatements>*
                        <TS> "}" <EOL>
<DxeAprioriSection> ::= <TS> "APRIORI" <MTS> "DXE" <MTS>
                        "{" <EOL>
                        <MacroDefinition>*
                        <InfStatements>*
                        <FileStatements>*
                        <TS> "}" <EOL>
<InfStatements>     ::= <TS> "INF" <MTS> [<InfOptions>] <InfFile>
<InfOptions>        ::= [<Use>] [<Rule>] [<SetVer>] [<SetUi>]
<Use>               ::= "USE" <Eq> <TargetArch> <MTS>
<TargetArch>        ::= <arch>
<Rule>              ::= "RuleOverride" <Eq> {<RuleUiName>} {"BINARY"}
                        <MTS>
<SetVer>            ::= "VERSION" <Eq> <CString> <MTS>
<SetUi>             ::= "UI" <Eq> <CString> <MTS>
<InfFile>           ::= if (MODULE_TYPE == "SEC"
                        || MODULE_TYPE == "PEI_CORE"
                        || MODULE_TYPE == "PEIM"):
                        <PATH> <Word> ".inf" [<FS> <RelocFlags>] <EOL> else:
                        <PATH> <Word> ".inf" <EOL>
<RelocFlags>        ::= {"RELOCS_STRIPPED"} {"RELOCS_RETAINED"}
<FileStatements>    ::= {<type1>} {<type2>} {<type3>} {<type4>}
                        {<type5>} <EOL>
<type1>             ::= <TS> "FILE" <MTS> <FvType1> <Eq> <NamedGuid> <Options1>
<type2>             ::= <TS> "FILE" <MTS> <FvType2> <Eq> <NamedGuid> <Options2>
<type3>             ::= <TS> "FILE" <MTS> "RAW" <Eq> <NamedGuidOrPcd>
                        <Options2>
<type4>             ::= <TS> "FILE" <MTS> "NON_FFS_FILE" <Eq> [<NamedGuid>]
                        <Options2>
<type5>             ::= <TS> "FILE" <MTS> "FV_IMAGE" <Eq>
                        <NamedGuidOrPcd> <Options2>
<FvType1>           ::= {"SEC"} {"PEI_CORE"} {"PEIM"}
<FvType2>           ::= {"FREEFORM"} {"PEI_DXE_COMBO"} {"DRIVER"}
                        {"DXE_CORE"} {"APPLICATION"} {"SMM_CORE"} {"SMM"}
<NamedGuidOrPcd>    ::= {<NamedGuid>} {"PCD(" <PcdName> ")"}
<NamedGuid>         ::= {<RegistryFormatGUID>} {"$(NAMED_GUID)"} {<GuidCName>}
<Options1>          ::= [<Use>] [<FileOpts>] <RelocFlags> <MTS>
                        "{" [<EOL>]
                        {<Filename>} {<SectionData>} <TS> <TS> "}" [<EOL>]
<Options2>          ::= [<Use>] [<FileOpts>] <MTS>
                        "{" [<EOL>]
                        <TS> {<Filename>} {<FileList>+} {<SectionData> <EOL>}
                        "}" <EOL>
<FileList>          ::= <TS> [<FfsAlignment>] <NormalFile> <EOL>
<FileOpts>          ::= ["FIXED" <MTS>] ["CHECKSUM" <MTS>]
                        [<FfsAlignment>]
<FfsAlignment>      ::= "Align" <Eq> <FfsAlignmentValues> <MTS>
<Filename>          ::= <TS> {<FvImage>} {<FdImage>} {<NormalFile>} <EOL>
<FvImage>           ::= <TS> "FV" <Eq> <FvUiName> <EOL>
<FdImage>           ::= <TS> "FD" <Eq> <FdUiName> <EOL>
<FdUiName>          ::= {<Word>} {"common"}
<NormalFile>        ::= <PATH> <Word> "." <Word> <EOL>
<SectionData>       ::= <MacroDefinition>*
```

```
                        [<PeiAprioriSection>]
                        [<DxeAprioriSection>]
                        <EncapSec>*
                        <LeafSections>*
<LeafSections>      ::= {<VerSection>} {<UiSec>} {<FvImgSection>}
                        {<DataSection>} {<DepexExpSection>}
<VerSection>        ::= <TS> "SECTION" <MTS> [<VerArgs>] "VERSION" <VerUniArg>
<UiSec>             ::= <TS> "SECTION" <MST> [<FfsAlignment>] "UI" <VerUniArg>
<FvImgSection>      ::= <TS> "SECTION" <MTS> [<FfsAlignment>] "FV_IMAGE"
                        <FvImgArgs>
<VerArgs>           ::= [<FfsAlignment>] [<Build>]
<Build>             ::= "BUILD_NUM" <Eq> <BuildVal> <MTS>
<BuildVal>          ::= {<HexDigit> <HexDigit> <HexDigit> <HexDigit>}
                        {"$(BUILD_NUMBER)"}
<VerUniArg>         ::= <Eq> {<StringData>} {<NormalFile>} <EOL>
<StringData>        ::= {<UnicodeString>} {<QuotedString>}
<FvImgArgs>         ::= <Eq> <FvUiName> <MTS>
                        "{" <EOL>
                        <MacroDefinition>*
                        <ExtendedFvEntry>*
                        [<FvAlignment>]
                        [<FvAttributes>]
                        [<PeiAprioriSection>]
                        [<DxeAprioriSection>]
                        <InfStatements>*
                        <FileStatements>*
                        <TS> "}" <EOL>
<DataSection>       ::= {<KnownSection>} {<SubTypeGuid>}
<KnownSection>      ::= <TS> "SECTION" <MTS> [<FfsAlignment>] <SecData>
<SecData>           ::= <LeafSecType> <MTS> [<ChkReloc> <MTS>]
                        <SectionOrFile>
<LeafSecType>       ::= {"COMPAT16"} {"PE32"} {"PIC"} {"TE"} {"RAW"}
                        {"FV_IMAGE"} {"DXE_DEPEX"} {"SMM_DEPEX"}
                        {"UI"} {"PEI_DEPEX"} {"VERSION"}
<SubTypeGuid>       ::= <TS> "SECTION" <MTS> [<FfsAlignment>] <STG_Data>
<STG_Data>          ::= "SUBTYPE_GUID" <MTS> <GuidValue> <Eq>
                        <NormalFile> <EOL>
<ChkReloc>          ::= if ((LeafSectionType == "PE32"
                        || LeafSectionType == "TE")
                        && (MODULE_TYPE == "SEC"
                        || MODULE_TYPE == "PEI_CORE"
                        || MODULE_TYPE == "PEIM")): [<RelocFlags>]
<SectionOrFile>     ::= {<Eq> <NormalFile> <EOL>} {<EncapSec>}
<EncapSec>          ::= <TS> "SECTION" <MTS> [<FfsAlignment>] <EncapSection>
                        <EOL>
<EncapSection>      ::= {<CompressSection>} {<GuidedSection>}
<CompressSection>   ::= <TS> "COMPRESS" <MTS> [<CompType>]
                        "{" <EOL>
                        <EncapSec>*
                        <LeafSections>*
                        <TS> "}" [<EOL>]
<CompType>          ::= {"PI_STD"} {"PI_NONE"} <MTS>
<GuidedSection>     ::= "GUIDED" <MTS> <NamedGuid> <MTS>
                        [<GuidedOptions>]
                        "{" <EOL>
                        <EncapSec>*
                        <LeafSections>*
                        <TS> "}" [<EOL>]
<GuidedOptions>     ::= [<GuidAttrPr>] [<GuidAttrASV>]
                        [<GuidHeaderSize>]
<GuidAttrPr>        ::= "PROCESSING_REQUIRED" <Eq> <TrueFalse> <MTS>
<GuidAttrASV>       ::= "AUTH_STATUS_VALID" <Eq> <TrueFalse> <MTS>
<GuidHeaderSize>    ::= "EXTRA_HEADER_SIZE" <Eq> <Number> <MTS>
<DepexExpSection>   ::= if ( COMPONENT_TYPE == "LIBRARY"
                        || LIBRARY_CLASS is declared in defines section of the
                        INF
                        || MODULE_TYPE == "USER_DEFINED" ):
                        [<Depex>]
                        else if ( MODULE_TYPE == "PEIM"
                        || MODULE_TYPE == "DXE_DRIVER"
                        || MODULE_TYPE == "DXE_RUNTIME_DRIVER"
                        || MODULE_TYPE == "DXE_SAL_DRIVER" || MODULE_TYPE ==
```

```
                        "DXE_SMM_DRIVER" ):
                        <Depex>
                        else if ( MODULE_TYPE == "UEFI_APPLICATION"
                        && MODULE_TYPE == "UEFI_DRIVER"
                        && MODULE_TYPE == "PEI_CORE"
                        && MODULE_TYPE == "DXE_CORE"
                        && MODULE_TYPE == "SMM_CORE"
                        && MODULE_TYPE == "SEC" ):
                        No DEPEX section is permitted
<Depex>             ::= if (MODULE_TYPE == "PEIM"):
                        <PeiDepexExp> else if (MODULE_TYPE ==
                        "DXE_SMM_DRIVER"):
                        <SmmDepexExp> [<DxeDepexExp>] else:
                        <DxeDepexExp>
<PeiDepexExp>       ::= <TS> "SECTION" <MTS> [<FfsAlignment>] "PEI_DEPEX_EXP"
                        <Eq> "{" [<EOL>] <PeiDepex> "}" <EOL>
<PeiDepex>          ::= [<TS> <BoolStmt> {<EOL>} {<MTS>}]
                        [<TS> <DepInstruct> {<EOL>} {<MTS>}]
                        [<TS> "end" {<EOL>} {<MTS>}]
<BoolStmt>          ::= {<Bool>} {<BoolExpress>}
                        {<GuidCName>} {<EOL>} {<MTS>}
<Bool>              ::= {"TRUE"} {"FALSE"} {<GuidCName>}
<GuidCName>         ::= <CName> # A Guid C Name
<BoolExpress>       ::= [<Not>] <GuidCName> [<OP> [<Not>] <GuidCName> ]*
<Not>               ::= "NOT" <MTS>
<OP>                ::= <MTS> {"AND"} {"OR"} <MTS>
<DepInstruct>       ::= "push" <MTS> <Filename>
<DxeDepexExp>       ::= <TS> "SECTION" <MTS> [<FfsAlignment>] "DXE_DEPEX_EXP"
                        <Eq> "{" [<EOL>] <DxeDepex> "}" <EOL>
<DxeDepex>          ::= [<TS> <SorStmt> {<EOL>} {<MTS>}]
                        [<TS> <GuidStmt> {<EOL>} {<MTS>}]
                        [<TS> <BoolStmt> {<EOL>} {<MTS>}]
                        [<TS> <DepInstruct> {<EOL>} {<MTS>}]
                        [<TS> "END" {<EOL>} {<MTS>}]
<SorStmt>           ::= "SOR" <BoolStmt>
<GuidStmt>          ::= {"before"} {"after"} <MTS> <Filename>
<SmmDepexExp>       ::= <TS> "SECTION" <MTS> [<FfsAlignment>] "SMM_DEPEX_EXP"
                        <Eq> "{" [<EOL>] <DxeDepex> "}" <EOL>
```

Restrictions

Filename

For BINARY ONLY content (`UEFI_DRIVER` and `UEFI_APPLICATION` .efi files) the file names specified in the elements (`FILE` and `SECTION`) of this section must be relative to the directory identified by the `WORKSPACE` or relative to a path listed in the `PACKAGES_PATH` system environment variable.

TargetArch

Only specific architectures are permitted - use of "common" is prohibited.

FvBaseAddress

The `FvBaseAddress` , if present, must be listed before the `FvAlignment` element. If present, the `FvForceRebase` must immediately follow the `FvBaseAddress` .

Parameters

FvBaseAddress

A `UINT64` value that will be used to rebase the code to run at a different address than the address specified by the location of the FV in the FD section.

SUBTYPE_GUID

This is short hand notation refering to content that will be placed in a Section of type: EFI_SECTION_FREEFORM_SUBTYPE_GUID. A single

`EFI_SECTION_FREEFORM_SUBTYPE_GUID` section is permitted in an FFS File of type `EFI_FV_FILETYPE_FREEFORM`

GuidCName

A word that is a valid C variable for a GUID.

Expression

Refer to the EDK II Expression Syntax Specification for more information.

COMPRESS

Compression sections that use `PI_STD` compression do not have `PROCESSING_REQUIRED = TRUE` flag, it is only required for `GUIDED` sections.

User Interface (UI) entries

There are three possible methods for specifying a User Interface string. 1) Specify the string value in the FDF file, 2) specify a ASCII plain text file that has an extension of ".ui" or 3) specify a Unicode file with an extension of ".uni" that contains a single Unicode string.

Paths

Unless otherwise noted, all file paths are relative to the WORKSPACE directory or relative to a directory listed in the PACKAGESPATH. In some cases, the tools will search well known paths for some files, for example, for FD filenames, the output will typically be located in the `$(OUTPUT_DIRECTORY)/$(TARGET)$(TAGNAME)/FV` directory.

Related Definitions

Note that no space characters are permitted on the left side of the expression (before the equal sign).

Target

This value must match a target identifier in the EDK II tools_def.txt file - the first field, where fields are separated by the underscore character. Wildcard characters are not permitted.

TagName

This must match a tag name field in the EDK II `tools_def.txt` file - second field. Wildcard characters are not permitted.

Example

```
[Fv.Root]
  FvAlignment        = 64
  ERASE_POLARITY     = 1
  MEMORY_MAPPED      = TRUE
  STICKY_WRITE       = TRUE
  LOCK_CAP           = TRUE
  LOCK_STATUS        = TRUE
  WRITE_DISABLED_CAP = TRUE
  WRITE_ENABLED_CAP  = TRUE
  WRITE_STATUS       = TRUE
  WRITE_LOCK_CAP     = TRUE
  WRITE_LOCK_STATUS  = TRUE
  READ_DISABLED_CAP  = TRUE
  READ_ENABLED_CAP   = TRUE
  READ_STATUS        = TRUE
  READ_LOCK_CAP      = TRUE
  READ_LOCK_STATUS   = TRUE

  INF VERSION = "1" $(WORKSPACE)/EdkNt32Pkg/Dxe/WinNtThunk/Cpu/Cpu.inf

  FILE DXE_CORE = B5596C75-37A2-4b69-B40B-72ABD6DD8708 {
```

```
    SECTION COMPRESS {
      DEFINE DC    = $(WORKSPACE)/Build/Nt32/DEBUG_IA32
      SECTION PE32 = $(DC)/B5596C75-37A2-4b69-B40B-72ABD6DD8708-DxeCore.efi
    }
  }

  FILE FREEFORM = 85C3EBE1-F58F-4820-8AD3-F2FB62DC3A23 {
    FvAlignment = 512K
    SECTION SUBTYPE_GUID AFC13561-9A65-4754-9C93-E133B3B8767C = $(WORKSPACE)/MyPackage/MyNewType/Binary/newform.bin
  }
  FILE FV_IMAGE = EF41A0E1 - 40B1 - 481f - 958E-6FB4D9B12E76 {
    FvAlignment = 512K
    WRITE_POLICY_RELIABLE = TRUE
    SECTION GUIDED 3EA022A4-1439-4ff2-B4E4-A6F65A13A9AB {
      SECTION FV_IMAGE = Dxe {
        APRIORI DXE {
          INF $(WORKSPACE)/a/a.inf
          INF $(EDK_SOURCE/a/c/c.inf
          INF $(WORKSPACE)/a/b/b.inf
        }
        INF a/d/d.inf
        ...
      }
    }
  }
DEFINE SAMPLE = $(EDK_SOURCE)/Sample

INF $(SAMPLE)/Universal/Network/Ip4/Dxe/Ip4.inf
INF $(SAMPLE)/Universal/Network/Ip4Config/Dxe/Ip4Config.inf
INF $(SAMPLE)/Universal/Network/Udp4/Dxe/Udp4.inf
INF $(SAMPLE)/Universal/Network/Tcp4/Dxe/Tcp4.inf
INF $(SAMPLE)/Universal/Network/Dhcp4/Dxe/Dhcp4.inf
INF $(SAMPLE)/Universal/Network/Mtftp4/Dxe/Mtftp4.inf
INF $(SAMPLE)/Universal/Network/SnpNt32/Dxe/SnpNt32.inf

FILE RAW = 197DB236-F856-4924-90F8-CDF12FB975F3 {
  $(OUTPUT_DIRECTORY)/$(TARGET)_$(TOOL_CHAIN_TAG)/$PLATFORM_ARCH)/File.bin
}

FILE RAW = 197DB236-F856-4924-90F8-CDF12FB975F3 {
  Align=16 $(PLATFORM_PACKAGE)/Binaries/File1.pdb
  Align=16 $(PLATFORM_PACKAGE)/Binaries/File2.pdb
  Align=16 $(PLATFORM_PACKAGE)/Binaries/File3.pdb
}
```

3.7 [Capsule] Sections

These sections are optional.

Summary

If capsule files are created, they will be created in the `$(OUTPUT_DIRECTORY)/$(TARGET)_$(TAGNAME)/FV` directory using the values from the individual instance of the build tools. (Build tools get these values after parsing DSC, INF, `target.txt`, `tools_def.txt` files and command line options.)

Conditional statements may be used anywhere within this section.

Prototype

```
<Capsule>           ::= "[Capsule" <UiCapsuleName> "]" <EOL>
                        <UefiTokens>
                        <CapsuleStmts>*
<UiCapsuleName>     ::= "." <Word>
<SetStatements>     ::= <TS> "SET" <MTS> <PcdName> <Eq> <VALUE> <EOL>
<VALUE>             ::= {<Number>} {<Boolean>} {<GUID>} {<CArray>}
                        {<CString>} {<UnicodeString>} {<Expression>}
<UefiTokens>        ::= <TS> "CAPSULE_GUID" <Eq> <GuidValue> <EOL>
                        [<TS> "CAPSULE_HEADER_SIZE" <Eq> <Bytes> <EOL>] [<TS>
                        "CAPSULE_FLAGS" <Eq> <Flags> <EOL>]
                        [<TS> "CAPSULE_HEADER_INIT_VERSION" <Eq> <Hex2> <EOL>]
<CapsuleStmts>      ::= {<MacroDefinition>} {<SetStatements>}
                        {<CapsuleData>}
<GuidValue>         ::= {<GuidCName>} {<GuidStructure>}
<GuidCName>         ::= <CName>
<GuidStructure>     ::= {<RegistryFormatGUID>} {<CFormatGUID>}
<Flags>             ::= <FlagName>
<FlagName>          ::= {"PersistAcrossReset"}
                        {"PersistAcrossReset" "," "InitiateReset"}
                        {"PersistAcrossReset" "," "PopulateSystemTable"}
                        {"PersistAcrossReset" "," "PopulateSystemTable"
                        "," "InitiateReset"}
                        {"PersistAcrossReset" "," InitiateReset"
                        "," "PopulateSystemTable"}
                        {"PopulateSystemTable"}
                        {"PopulateSystemTable" "," "PersistAcrossReset"}
                        {"PopulateSystemTable" "," "PersistAcrossReset"
                        "," "InitiateReset"}
                        {"PopulateSystemTable" "," "InitiateReset" ","
                        "PersistAcrossReset"}
                        {"InitiateReset" "," "PersistAcrossReset"}
                        {"InitiateReset" "," "PersistAcrossReset"
                        "," "PopulateSystemTable"}
                        {"InitiateReset" "," "PopulateSystemTable" ","
                        "PersistAcrossReset"}
<CapsuleData>       ::= <InfStatements>*
                        <FileStatements>*
                        <FvStatements>*
                        <FdStatenents>*
                        <FmpFileStatement>*
                        <FmpPayload>* <Afile>*
<InfStatements>     ::= <TS> "INF" <MTS> [<InfOptions>] <InfFile> <EOL>
<InfOptions>        ::= [<Use>] [<Rule>] [<SetVer>] [<SetUi>]
<Use>               ::= "USE" <Eq> <TargetArch> <MTS>
<TargetArch>        ::= <arch>
<Rule>              ::= "RuleOverride" <Eq> {<RuleUiName>} {"BINARY"} <MTS>
<SetVer>            ::= "VERSION" <Eq> <CString> <MTS>
<SetUi>             ::= "UI" <Eq> <CString> <MTS>
<InfFile>           ::= if (MODULE_TYPE == SEC
                        || MODULE_TYPE == PEI_CORE
                        || MODULE_TYPE == PEIM):
```

```
                        <PATH> <Word> ".inf" [<FS> <RelocFlags>] else:
                        <PATH> <Word> ".inf"
<RelocFlags>        ::= {"RELOCS_STRIPPED"} {"RELOCS_RETAINED"}
<KeyString>         ::= <Target> "_" <TagName> "_" <TargetArch>
<Target>            ::= {Target} {"$(TARGET)"}
<TagName>           ::= {TagName} {"$(TOOL_CHAIN_TAG)"}
<FileStatements>    ::= <TS> {<type1>} {<type2>} {<type3>} {<type4>}
<type1>             ::= "FILE" <FvType1> <Eq> <NamedGuid> <Options1>
<type2>             ::= "FILE" <FvType2> <Eq> <NamedGuid> <Options2>
<type3>             ::= "FILE" "RAW" <Eq> <NamedGuidOrPcd>
                        <Options2>
<type4>             ::= "FILE" "NON_FFS_FILE" <Eq> [<NamedGuid>] <Options2>
<type5>             ::= "FILE" "FV_IMAGE" <Eq> <NamedGuidOrPcd>
                        <Options2>
<FvType1>           ::= {"SEC"} {"PEI_CORE"} {"PEIM"}
<FvType2>           ::= {"FREEFORM"} {"PEI_DXE_COMBO"} {"DRIVER"}
                        {"DXE_CORE"} {"APPLICATION"} {"SMM_CORE"} {"SMM"}
<NamedGuid>         ::= {<RegistryFormatGUID>} {"$(NAMED_GUID)"}
<NamedGuidOrPcd>    ::= {<NamedGuid>} {"PCD(" <PcdName> ")"}
<Options1>          ::= [<Use>] [<FileOpts>] [<RelocFlags>]
                        "{" [<EOL>]
                        <TS> {<Filename>} {<SectionData>} [<EOL>] <TS> "}"
                        <EOL>
<Options2>          ::= [<Use>] [<FileOpts>]
                        "{" [<EOL>]
                        {<Filename>} {<FileList>+} {<SectionData> <EOL>}
                        <TS> "}" <EOL>
<FileList>          ::= <TS> [<FfsAlignment>] <NormalFile> <EOL>
<FileOpts>          ::= ["FIXED" <MTS>] ["CHECKSUM" <MTS>]
                        [<FfsAlignment>]
<FfsAlignment>      ::= "Align" <Eq> <FfsAlignmentValues>
<FvAlignment>       ::= [<TS> "FvBaseAddress" <Eq> <UINT64> <EOL>]
                        [<TS> "FvForceRebase" <Eq> <TrueFalse> <EOL>]
                        "FvAlignment" <Eq> <FvAlignmentValues> <EOL>
<Filename>          ::= <TS> {<FvImage>} {<FdImage>} {<NormalFile>} <EOL>
<FvImage>           ::= "FV" <Eq> <FvUiName> <EOL>
<FdImage>           ::= "FD" <Eq> <FdUiName> <EOL>
<FdUiName>          ::= {<Word>} {"common"}
<NormalFile>        ::= <PATH> <Word> "." <Word> <EOL>
<SectionData>       ::= <MacroDefinition>*
                        [<PeiAprioriSection>]
                        [<DxeAprioriSection>]
                        <EncapSec>*
                        <LeafSections>*
<PeiAprioriSection> ::= "APRIORI PEI" <MTS>
                        "{" <EOL>
                        [<DefineStatements>]
                        <InfStatements>*
                        <FileStatements>* <TS> "}" <EOL>
<DxeAprioriSection> ::= "APRIORI DXE" <MTS>
                        "{" <EOL>
                        [<DefineStatements>]
                        <InfStatements>*
                        <FileStatements>*
                        <TS> "}" <EOL>
<LeafSections>      ::= {<VerSection>} {<UiSec>} {<FvImgSection>}
                        {<DataSection>} {<DepexExpSec>}
<VerSection>        ::= "SECTION" <MTS> [<VerArgs>] "VERSION" <MTS> <UniArg>
<UiSection>         ::= "SECTION" <MTS> [<FfsAlignment>] "UI" <MTS> <UniArg>
<FvImgSection>      ::= "SECTION" <MTS> [<FfsAlignment>] "FV_IMAGE" <MTS>
                        <FvImgArgs>
<VerArgs>           ::= [<FfsAlignment>] [<Build>]
<Build>             ::= "BUILD_NUM" <Eq> <BuildVal> <MTS>
<BuildVal>          ::= {[a-fA-F0-9]{4}} {"$(BUILD_NUMBER)"}
<UniArg>            ::= <Eq> {<StringData>} {<NormalFile>} <EOL>
<StringData>        ::= {<UnicodeString>} {<QuotedString>}
                        {<NamedMacro>}
<NamedMacro>        ::= if (VerSection): "$(INF_VERSION)" else if (UiSection):
                        {"$(INF_VERSION)"} {"$(MODULE_NAME)"}
<FvImgArgs>         ::= <Eq> <FvUiName>
                        "{" <EOL>
                        <MacroDefinition>*
```

```
                        <ExtendedFvEntry>*
                        [<FvAlignment>]
                        <FvAttributes>*
                        [<FileSystemGuid>]
                        [<PeiAprioriSection>]
                        [<DxeAprioriSection>]
                        <InfStatements>*
                        <FileStatements>*
                        <TS> "}" <EOL>
<ExtendedFvEntry>   ::= <TS> "FV_EXT_ENTRY_TYPE" <MTS> <TypeValue>
                        "{" [<EOL>]
                        <TS>
                        {"FILE" <Eq> <BinaryFile>}
                        {"DATA" <Eq> "{" <DataContent> "}"}
                        [<EOL>]
                        <TS> "}" <EOL>
<DataContent>       ::= {<RawData>} {<CFormatGUID>} {<UINT8z>} {<UINT16z>}
                        {<UINT32z>} {<UINT64z>}
<TypeValue>         ::= "TYPE" <Eq> <Hex4> <MTS>
<Afile>             ::= "APPEND" <Eq> <BinaryFile> <EOL>
<BinaryFile>        ::= [<PATH>] <Word> ["." {<bin>} {<dat>}] [<EOL>]
<bin>               ::= {"bin"} {"BIN"} {"Bin}
<dat>               ::= {"dat"} {"DAT"} {"Dat"} {"data"} {"DATA"}
                        {"Data"}
<DataSection>       ::= {<KnownSection>} {<SubTypeGuid>}
<KnownSection>      ::= "SECTION" <MTS> [<FfsAlignment>] <SecData>
<SecData>           ::= <LeafSecType> [<ChkReloc>] <SectionOrFile>
<LeafSecType>       ::= {"COMPAT16"} {"PE32"} {"PIC"} {"TE"} {"RAW"}
                        {"FV_IMAGE"} {"DXE_DEPEX"} {"SMM_DEPEX"}
                        {"UI"} {"PEI_DEPEX"} {"VERSION"}
<SubTypeGuid>       ::= <TS> "SECTION" <MTS> [<FfsAlignment>] <SgData>
<SgData>            ::= "SUBTYPE_GUID" <MTS> <GuidValue> <Eq>
                        <NormalFile> <EOL>
<GuidValue>         ::= {<GuidCName>} {<GuidStructure>}
<GuidCName>         ::= <CName>
<GuidStructure>     ::= {<RegistryFormatGUID>} {<CFormatGUID>}
<ChkReloc>          ::= if ((LeafSecType == "PE32"
                        || LeafSecType == "TE")
                        && (MODULE_TYPE == "SEC"
                        || MODULE_TYPE == "PEI_CORE"
                        || MODULE_TYPE == "PEIM")): [<RelocFlags>]
<SectionOrFile>     ::= {<Eq> <NormalFile> <EOL>} {<EncapSec>}
<EncapSec>          ::= "SECTION" <MTS> [<FfsAlignment>] <EncapSection>
<EncapSection>      ::= {<CompressSection>} {<GuidedSection>}
<CompressSection>   ::= "COMPRESS" <MTS> [<CompType>]
                        "{" <EOL>
                        <MacroDefinition>*
                        [<PeiAprioriSection>]
                        [<DxeAprioriSection>]
                        <EncapSec>*
                        <LeafSections>*
                        <TS> "}" <EOL>
<CompType>          ::= {"PI_STD"} {"PI_NONE"} <MTS>
<GuidedSection>     ::= "GUIDED" <NamedGuid> [<GuidedOptions>]
                        "{" <EOL>
                        <MacroDefinition>*
                        [<PeiAprioriSection>]
                        [<DxeAprioriSection>]
                        <EncapSec>*
                        <LeafSections>*
                        <TS> "}" <EOL>
<GuidedOptions>     ::= [<GuidAttrPR>] [<GuidAttrASV>] [<GuidHeaderSize>]
<GuidAttrPR>        ::= "PROCESSING_REQUIRED" <Eq> <TrueFalse> <MTS>
<GuidAttrASV>       ::= "AUTH_STATUS_VALID" <Eq> <TrueFalse> <MTS>
<GuidHeaderSize>    ::= "EXTRA_HEADER_SIZE" <Eq> <Number> <MTS>
<FvUiName>          ::= {<Word>} {"common"}
<FvStatements>      ::= "FV" <Eq> <FvNameOrFilename> <EOL>
<FvNameOrFilename>  ::= {<FvUiName>} {<FvFilename>}
<FvFilename>        ::= [<PATH>] <Word> "." "fv"
<FdStatements>      ::= "FD" <Eq> <FdNameOrFilename> <EOL>
<FdNameOrFilename>  ::= {<FdUiName>} {<FdFilename>}
<FdFilename>        ::= [<PATH>] <Word> "." "fd"
```

```
<AnyFile>           ::= "FILE" <MTS> "DATA" <Eq> <NormalFile> <EOL>
<FvAttributes>      ::= [<TS> "MEMORY_MAPPED" <Eq> <TrueFalse> <EOL>]
                        [<TS> "LOCK_CAP" <Eq> <TrueFalse> <EOL>]
                        [<TS> "LOCK_STATUS" <Eq> <TrueFalse> <EOL>]
                        [<TS> "WRITE_LOCK_CAP" <Eq> <TrueFalse> <EOL>]
                        [<TS> "WRITE_LOCK_STATUS" <Eq> <TrueFalse> <EOL>]
                        [<TS> "WRITE_ENABLED_CAP" <Eq> <TrueFalse> <EOL>]
                        [<TS> "WRITE_DISABLED_CAP" <Eq> <TrueFalse> <EOL>]
                        [<TS> "WRITE_STATUS" <Eq> <TrueFalse> <EOL>]
                        [<TS> "STICKY_WRITE" <Eq> <TrueFalse> <EOL>]
                        [<TS> "WRITE_POLICY_RELIABLE" <Eq> <TrueFalse> <EOL>]
                        [<TS> "READ_LOCK_CAP" <Eq> <TrueFalse> <EOL>]
                        [<TS> "READ_LOCK_STATUS" <Eq> <TrueFalse> <EOL>]
                        [<TS> "READ_ENABLED_CAP" <Eq> <TrueFalse> <EOL>]
                        [<TS> "READ_DISABLED_CAP" <Eq> <TrueFalse> <EOL>]
                        [<TS> "READ_STATUS" <Eq> <TrueFalse> <EOL>]
<FileSystemGuid>    ::= "FileSystemGuid" <Eq> <NamedGuid>
<DepexExpSection>   ::= if ( COMPONENT_TYPE == "LIBRARY"
                        || LIBRARY_CLASS is declared in defines section of the
                        INF
                        || MODULE_TYPE == "USER_DEFINED" ):
                        [<Depex>]
                        else if ( 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 == "UEFI_APPLICATION"
                        || MODULE_TYPE == "UEFI_DRIVER"
                        || MODULE_TYPE == "PEI_CORE"
                        || MODULE_TYPE == "DXE_CORE"
                        || MODULE_TYPE == "SMM_CORE"
                        || MODULE_TYPE == "SEC"):
                        No DEPEX section is permitted
<Depex>             ::= if (MODULE_TYPE == PEIM): <PeiDepexExp> elif
                        (MODULE_TYPE == "DXE_SMM_DRIVER"): <SmmDepexExp>
                        [<DxeDepexExp>] else:
                        <DxeDepexExp>
<PeiDepexExp>       ::= "SECTION" <MTS> [<FfsAlignment>]
                        "PEI_DEPEX_EXP"
                        <Eq> "{" [<EOL>] <PeiDepex> "}" <EOL>
<PeiDepex>          ::= [<BoolStmt> {<EOL>} {<MTS>}]*
                        [<DepInstruct> {<EOL>} {<MTS>}]*
                        ["end"] [<EOL>]
<BoolStmt>          ::= {<Boolean>} {<BoolExpress>}
                        {<GuidCName>} <EOL>
<Boolean>           ::= {"TRUE"} {"FALSE"} {<GuidCName>}
<GuidCName>         ::= <CName> # A Guid C Name
<BoolExpress>       ::= <GuidCName> [<OP> ["NOT"] <GuidCName> ]*
<OP>                ::= <MTS> {"AND"} {"OR"} <MTS>
<DepInstruct>       ::= "push" <Filename>
<DxeDepexExp>       ::= "SECTION" <MTS> [<FfsAlignment>] <DxeExp>
<DxeExp>            ::= "DXE_DEPEX_EXP" <Eq> <MTS>
                        "{" [<EOL>] <DxeDepex>* "}" <EOL>
<DxeDepex>          ::= [<SorStmt> {<EOL>} {<MTS>}]*
                        [<GuidStmt> {<EOL>} {<MTS>}]*
                        [<BoolStmt> {<EOL>} {<MTS>}]*
                        [<DepInstruct> {<EOL>} {<MTS>}]*
                        ["END"] {<EOL>} {<MTS>}]
<SorStmt>           ::= "SOR" <MTS> <BoolStmt>
<GuidStmt>          ::= {"before"} {"after"} <MTS> <Filename>
<SmmDepexExp>       ::= "SECTION" <MTS> [<FfsAlignment>] "SMM_DEPEX_EXP"
                        <Eq> "{" [<EOL>] <DxeDepex> "}" <EOL>
<FmpPayload>        ::= <TS> "FMP_PAYLOAD" <Eq> <UiFmpName> <EOL>
<UiFmpName>         ::= <Word>
<FmpFileStatement>  ::= <TS> "FILE" <Space> "DATA" <Eq> <Filename> <EOL>
```

Restrictions

Filename

For BINARY ONLY content (`UEFI_DRIVER` and `UEFI_APPLICATION` .efi files) the file names specified in the elements (`FILE` and `SECTION`) of this section must be relative to the directory identified by the `WORKSPACE` system environment variable or relative to a path listed in the `PACKAGES_PATH` system environment variable.

TargetArch

Only specific architectures are permitted - use of "common" is prohibited.

GuidValue

When specifying the CAPSULE_GUID value for an FMP Capsule, the GUID value must be set to 6dcbd5ed-e82d-4c44-bda1-7194199ad92a.

Parameters

UiCapsuleName

Filename that will be used to create an FV file.

CreateFile

Filename to create instead of using the `UiCapsuleName` .

FvBaseAddress

The `FvBaseAddress` , if present, must be listed before the `FvAlignment` element.

The `FvForceRebase` flag, if present, must immediately follow the `FvBaseAddress` .

SUBTYPE_GUID

This is short hand notation refering to content that will be placed in a Section of type: `EFI_SECTION_FREEFORM_SUBTYPE_GUID` . A single

`EFI_SECTION_FREEFORM_SUBTYPE_GUID` section is permitted in an FFS File of type `EFI_FV_FILETYPE_FREEFORM`

Depex

Depex sections are prohibited for modules with a `MODULE_TYPE` of `UEFI_DRIVER` , `UEFI_APPLICATION` , `PEI_CORE` , `DXE_CORE` or `SEC` . modules with `MODULE_TYPE` of `USER_DEFINED` and all Library instances may or may not have a `DEPEX` section.

Modules that use `DXE_RUNTIME_DRIVER` as the `MODULE_TYPE` require a `DEPEX` section if and only if they are pure DXE Runtime drivers - UEFI Runtime Drivers that use the `DXE_RUNTIME_DRIVER` `MODULE_TYPE` must not have a `DEPEX` section.

If a library instance is required by a module that prohibits depex sections, the libraries' depex section is ignored. For modules that do require a depex section, the depex section of all dependent libraries is AND'ed with the depex section of the module.

Expression

Refer to the EDK II Expression Syntax Specification for more information.

Paths

Unless otherwise specified, all file specified paths are relative to the `WORKSPACE` directory or relative to a directory listed in the `PACKAGES_PATH` . In some cases, the tools will search well known paths for some files, for example, for FD filenames, the output will typically be located in the `$(OUTPUT_DIRECTORY)/$(TARGET)_$(TAGNAME)/FV` directory.

COMPRESS

Compression sections that use `PI_STD` compression do not have `PROCESSING_REQUIRED = TRUE` flag, it is only required for GUIDED sections.

User Interface (UI) entries

There are three possible methods for specifying a User Interface string. 1) Specify the string value in the FDF file, 2) specify a plain ASCII text file that has an extension of ".ui" or 3) specify a Unicode file with an extension of ".uni" that contains a single Unicode string.

Append

The APPEND element is used to specify a workspace relative path (or relative to a directory listed in the PACKAGES_PATH) and file name for a raw binary file. The order that files will be appended is the order in which they are listed in the section. Any driver that needs to access these files must have a prior knowledge of the content - for example, a new payload image - as these files are not processed by the EDK II tools. They have no section types or any other kind of identifier that has been defined by UEFI/PI specifications.

Related Definitions

Target

Must match a target identifier in the EDK II tools_def.txt file - the first field, where fields are separated by the underscore character. Wildcard characters are not permitted.

TagName

Must match a tag name field in the EDK II `tools_def.txt` file - second field. Wildcard characters are not permitted

Example

```
[Capsule.Fob]
CAPSULE_GUID        = 42857F0A-13F2-4B21-8A23-53D3F714B840
CAPSULE_HEADER_SIZE = 32

FILE FV_IMAGE = EF41A0E1-40B1-481f-958E-6FB4D9B12E76 {
  SECTION GUIDED 3EA022A4-1439-4ff2-B4E4-A6F65A13A9AB {
    SECTION FV_IMAGE = Dxe {
      APRIORI DXE {
        INF a/a/a.inf
        INF a/c/c.inf
        INF a/b/b.inf
      }
      INF a/d/d.inf
     ...
    }
  }
}

[Capsule.FmpCapsuleImage]
  # normal header for FMP capsule content
  # special Guid
  CAPSULE_GUID = 6dcbd5ed-e82d-4c44-bda1-7194199ad92a
  # normal header
  CAPSULE_FLAGS = PersistAcrossReset, InitiateReset
  # normal header
  CAPSULE_HEADER_SIZE = 0x20
  # The following identifies this as an FMP capsule header
  CAPSULE_HEADER_INIT_VERSION = 0x1

  FILE DATA   = Driver1.efi
  FILE DATA   = Driver2.efi  # zero or more
  FMP_PAYLOAD = Payload1     # zero or more
```

3.8 [FmpPayload] Sections

These are optional sections that describes the FMP payload content for FMP Capsule files.

There must be at least one and at most two `<FmpFileData>` statements. The `<FmpFileData>` statements start with `FILE DATA`. The first statement provides the information for UpdateImage in an `EFI_FIRMWARE_MANAGEMENT_CAPSULE_IMAGE_HEADER`. The second statement, if present, provides the information for VendorCode in an `EFI_FIRMWARE_MANAGEMENT_CAPSULE_IMAGE_HEADER`.

Prototype

```
<FmpPayload>       ::= "[FmpPayload" "." <UiFmpName> "]" <EOL>
                       <FmpTokens>
                       <FmpFileData>{1,2}
<UiFmpName>        ::= <Word>
<FmpTokens>        ::= [<TS> "IMAGE_HEADER_INIT_VERSION" <Eq> <Hex2> <EOL>]
                       <TS> "IMAGE_TYPE_ID" <Eq> <RegistryFormatGUID> <EOL>
                       [<TS> "IMAGE_INDEX" <Eq> <Hex2> <EOL>]
                       [<TS> "HARDWARE_INSTANCE" <Eq> <Hex2> <EOL>]
                       [<TS> "MONOTONIC_COUNT"<Eq> <NumValUint64> <EOL>]
                       [<TS> "CERTIFICATE_GUID"<Eq> <RegistryFormatGUID><EOL>]
<FmpFileData>      ::= <FileStatements>*
                       <FvStatements>*
                       <FdStatenents>*
<FileStatements>   ::= <TS> "FILE" <Space> "DATA" <Eq> <Filename> <EOL>
<FvStatements>     ::= "FV" <Eq> <FvNameOrFilename> <EOL>
<FvNameOrFilename> ::= {<FvUiName>} {<FvFilename>}
<FvUiName>         ::= {<Word>} {"common"}
<FvFilename>       ::= [<PATH>] <Word> "." "fv"
<FdStatements>     ::= "FD" <Eq> <FdNameOrFilename> <EOL>
<FdNameOrFilename> ::= {<FdUiName>} {<FdFilename>}
<FdUiName>         ::= {<Word>} {"common"}
<FdFilename>       ::= [<PATH>] <Word> "." "fd"
```

Note: The `CERTIFICATE_GUID` and `MONOTONIC_COUNT` must work as a pair. If `CERTIFICATE_GUID` is provided, the FMP payload is processed as UEFI FMP Authentication format, and `MONOTONIC_COUNT` **MUST** be provided. If `CERTIFICATE_GUID` is not provided, the FMP payload is processed as UEFI FMP non-Authentication format, and `MONOTONIC_COUNT` **MUST NOT** be provided.

Example

```
[FmpPayload.Payload1]
  # FMP payload header
  IMAGE_HEADER_INIT_VERSION = 0x02
  # FMP payload header
  IMAGE_TYPE_ID     = 938A6F2E-9711-49CE-90D5-7ED68AC96501
  IMAGE_INDEX       = 0x1 # FMP payload header
  HARDWARE_INSTANCE = 0x0 # FMP payload header

  FILE DATA = UpdateImage.bin
  FILE DATA = VendorCodeBytes.bin # optional
```

3.9 [Rule] Sections

These are optional sections that describes the `[Rule]` content found in FDF files.

Summary

This section is similar to the `[FV]` section, with the a few exceptions. The INF statements are not permitted within a rules section.

Rules are used as templates, normally using variable names instead of fully qualified names, while INF statements are always are fully qualified file names.

The following list specifies the allowed variables that may be used exactly as typed:

`$(WORKSPACE)` , `$(EDK_SOURCE)` , `$(EFI_SOURCE)` , `$(TARGET)` , `$(TOOL_CHAIN_TAG)` , `$(ARCH)` , `$(MODULE_NAME)` , `$(OUTPUT_DIRECTORY)` , `$(BUILD_NUMBER)` , `$(INF_VERSION)` , `$(NAMED_GUID)` , `$(INF_OUTPUT)`

Conditional statements may be used anywhere within this section.

Prototype

```
<Rules>             ::= "[Rule" <RuleArgs> "]" <EOL> <FileStatements>
<RuleArgs>          ::= "." <arch> "." <ModuleType>
                        [<TemplateName>]
<ModuleType>        ::= {<EdkComponentType>} {<Edk2ModuleType>}
<Edk2ModuleType>    ::= {"SEC"} {"PEI_CORE"} {"PEIM"} {"SMM_CORE"}
                        {"DXE_CORE"} {"DXE_DRIVER"}
                        {"DXE_SAL_DRIVER"} {"DXE_SMM_DRIVER"}
                        {"DXE_RUNTIME_DRIVER"} {"UEFI_DRIVER"}
                        {"UEFI_APPLICATION"} {"USER_DEFINED"}
<EdkComponentType>  ::= {"LIBRARY"} {"APPLICATION"} {"AcpiTable"}
                        {"BINARY"} {"BS_DRIVER"} {"LOGO"}
                        {"Legacy16"} {"Microcode"} {"PE32_PEIM"}
                        {"RAWFILE"} {"RT_DRIVER"} {"SAL_RT_DRIVER"}
                        {"SECURITY_CORE"} {"COMBINED_PEIM_DRIVER"}
                        {"PIC_PEIM"} {"RELOCATABLE_PEIM"}
                        {"PEI_CORE"}
<TemplateName>      ::= "." <RuleUiName>
<RuleUiName>        ::= {<Word>} {"BINARY"}
<FileStatements>    ::= if (MODULE_TYPE == "SEC"
                        || MODULE_TYPE == "PEI_CORE"
                        || MODULE_TYPE == "PEIM"
                        || COMPONENT_TYPE == "PEI_CORE"
                        || COMPONENT_TYPE == "PIC_PEIM"
                        || COMPONENT_TYPE == "RELOCATABLE_PEIM" ||
                        COMPONENT_TYPE == "SECURITY_CORE"
                        || COMPONENT_TYPE == "PE32_PEIM" ):
                        <TS> "FILE" <MTS> <FvType1> <Eq>
                        <FileStatement1>
                        elif (MODULE_TYPE == "DXE_CORE" || MODULE_TYPE ==
                        "DXE_DRIVER"
                        || MODULE_TYPE == "DXE_SAL_DRIVER" || MODULE_TYPE ==
                        "SMM_CORE"
                        || MODULE_TYPE == "DXE_SMM_DRIVER"
                        || MODULE_TYPE == "UEFI_DRIVER"
                        || MODULE_TYPE == "UEFI_APPLICATION"
                        || MODULE_TYPE == "USER_DEFINED"
                        || COMPONENT_TYPE == "BS_DRIVER"
                        || COMPONENT_TYPE ==
                        "COMBINED_PEIM_DRIVER"
                        || COMPONENT_TYPE == "APPLICATION"):
                        {<FileStatement2>} {<FileStatement3>}
                        elif (MODULE_TYPE == "FV_IMAGE"):
                        <FileStatement4> else:
                        <TS> "FILE" <MTS> "NON_FFS_FILE" <Eq>
```

```
                        [<NamedGuid>] [<Options>] <EOL>
<FileStatement1>    ::= <NamedGuid> [<RelocFlags> <MTS>] [<Options>] <EOL>
<FileStatement2>    ::= <TS> "FILE" <MTS> <FvType2> <Eq> <NamedGuid>
                        [<Options>] <EOL>
<FileStatement3>    ::= <TS> "FILE" <MTS> "RAW" <Eq> <NamedGuidOrPcd>
                        [<Options>] <EOL>
<FileStatement4>    ::= <TS> "FILE" <MTS> "FV_IMAGE" <Eq>
                        <NamedGuidOrPcd> [<Options>] <EOL>
<NamedGuid>         ::= {"$(NAMED_GUID)"} {<RegistryFormatGUID>}
                        {<Sym>} <MTS>
<Sym>               ::= "$(" <Word> ")"
<NamedGuidOrPcd>    ::= <NamedGuid> <MTS>
                        {"PCD(" <PcdName> ")"} {<GuidValue>} <MTS>
<GuidValue>         ::= {<GuidCName>} {<GuidStructure>}
<GuidCName>         ::= <CName>
<GuidStructure>     ::= {<RegistryFormatGUID>} {<CFormatGUID>}
<FvType1>           ::= {"SEC"} {"PEI_CORE"} {"PEIM"} {"PEI_DXE_COMBO"}
<FvType2>           ::= {"FREEFORM"} {"DRIVER"} {"DXE_CORE"}
                        {"APPLICATION"} {"SMM_CORE"} {"SMM"}
<RelocFlags>        ::= {"RELOCS_STRIPPED" <MTS>}
                        {"RELOCS_RETAINED" <MTS>}
<Options>           ::= [<UseLocal>] [<FileOpts>] <FileSpec>
<UseLocal>          ::= <KeyString> ["," <KeyString>]
<KeyString>         ::= <Target> "_" <TagName> "_" <ToolArch>
<Target>            ::= {<Word>} {"$(TARGET)"}
<TagName>           ::= {<Word>} {"$(TOOL_CHAIN_TAG)"}
<ToolArch>          ::= {<Arch>} {"$(ARCH)"}
<FileOpts>          ::= ["Fixed" <MTS>] ["Checksum" <MTS>]
                        [<FfsAlignment>]
<FfsAlignment>      ::= "Align" <Eq> <FfsAlignmentValues> <MTS>
<FileSpec>          ::= {<SimpleFile>} {<ComplexFile>} {<SbtGuid>}
<SimpleFile>        ::= <LeafSecType> [<FileOpts>] <VarFile> <EOL>
<LeafSecType>       ::= {"COMPAT16"} {"PE32"} {"PIC"} {"TE"}
                        {"FV_IMAGE"} {"RAW"} {"DXE_DEPEX"} {"UI"}
                        {"PEI_DEPEX"} {"SMM_DEPEX"} {"VERSION"}
<SbtGuid>           ::= "SUBTYPE_GUID" <MTS> <GuidValue> <MTS> <FName> <EOL>
<VarFile>           ::= {<FilenameVariable>} {<FName>} {<Ext>}
<FName>             ::= [<PATH>] <Word> "." <Word>
<FilenameVariable>  ::= "$(INF_OUTPUT)/$(MODULE_NAME)" "." <Word>
<ComplexFile>       ::= "{" <EOL>
                        <EncapSection>*
                        <LeafSections>*
                        <TS> "}" <EOL>
<EncapSection>      ::= <TS> {<CompressSection>} {<GuidedSection>}
<CompressSection>   ::= "COMPRESS" <MTS> [<CompType>]
                        "{" [<EOL>]
                        <EncapSec>*
                        <LeafSections> <EOL>*
                        <TS> "}" <EOL>
<CompType>          ::= {"PI_STD"} {"PI_NONE"} <MTS>
<GuidedSection>     ::= "GUIDED" <MTS> "$(NAMED_GUID)" <MTS> [<GAttr>]
                        "{" <EOL>
                        <EncapSec>*
                        <LeafSections>*
                        <TS> "}" <EOL>
<GAttr>             ::= [<GuidAttrPR>] [<GuidAttrASV>] [<GuidHeaderSize>]
<GuidAttrPR>        ::= "PROCESSING_REQUIRED" <Eq> <BoolType> <MTS>
<GuidAttrASV>       ::= "AUTH_STATUS_VALID" <Eq> <BoolType> <MTS>
<GuidHeaderSize>    ::= "EXTRA_HEADER_SIZE" <Eq> <Number> <MTS>
<LeafSections>      ::= <TS> {<C16Sec>} {<PeSec>} {<PicSec>} {<TeSec>}
                        {<FvSec>} {<RawSec>} {<DxeDepSec>}
                        {<UiSec>} {<VerSec>} {<PeiDepSec>}
                        {<SmmDepSec>} {<SubtypeGuidSec>}
<C16Sec>            ::= "COMPAT16" <MTS> <C16FileType> [<FileOrExt>] <EOL>
<PeSec>             ::= "PE32" <MTS> <Pe32FileType> [<FileOrExt>] <EOL>
<PicSec>            ::= "PIC" <MTS> <PicFileType> [<FileOrExt>] <EOL>
<TeSec>             ::= "TE" <MTS> <TeFileType> [<FileOrExt>] <EOL>
<RawSec>            ::= "RAW" <MTS> <RawFileType> [<FileOrExtOrPcd>] <EOL>
<DxeDepSec>         ::= "DXE_DEPEX" <MTS> <DdFileType> [<FileOrExt>] <EOL>
<SmmDepSec>         ::= "SMM_DEPEX" <MTS> <DdFileType> [<FileOrExt>] <EOL>
<UiSec>             ::= "UI" <MTS> <UiFileType> [<FileOrExt>] <EOL>
<VerSec>            ::= "VERSION" <MTS> <VerFileType> [<FileOrExt>] <EOL>
```

```
<PeiDepSec>          ::= "PEI_DEPEX" <MTS> <PdFileType>
                         [<FileOrExt>] <EOL>
<SubtypeGuidSec>     ::= "SUBTYPE_GUID" <MTS> <GuidValue> <MTS>
                         <File> <EOL>
<FileOrExt>          ::= {<VarFile>} {<Ext>} <MTS>
<FileOrExtOrPcd>     ::= {<VarFile>} {<Ext>} {"PCD(" <PcdName> ")"}
<Ext>                ::= <FS> "." [a-zA-Z][a-zA-Z0-9]{0,}
<C16FileType>        ::= "COMPAT16" [<FfsAlignment>]
<Pe32FileType>       ::= "PE32" <MTS> [<FfsAlignment>]
                         [<ChkReloc>]
<ChkReloc>           ::= if (MODULE_TYPE == "SEC"
                         || MODULE_TYPE == "PEI_CORE"
                         || MODULE_TYPE == "PEIM"): [<RelocFlags>]
<PicFileType>        ::= "PIC" <MTS> [<FfsAlignment>]
<TeFileType>         ::= "TE" <MTS> [<FfsAlignment>] [<ChkReloc>]
<RawFileType>        ::= {<BinTypes>} {<AcpiFileTypes>} <MTS>
                         [<FfsAlignment>]
<BinTypes>           ::= {"BIN"} {"RAW"}
<AcpiFileTypes>      ::= {"ACPI"} {"ASL"} <MTS> ["Optional" <MTS>]
<DdFileType>         ::= {"DXE_DEPEX"} {"SMM_DEPEX"} [<DpxAlign>]
<DpxAlign>           ::= ["Optional" <MTS>] [<FfsAlignment>]
<UiFileType>         ::= {<UiFile>} {<UiString>}
<UiFile>             ::= "UI" <MTS> [<UiOpts>]
<UiString>           ::= "STRING" <Eq> <StringVal> [<FfsAlignment>]
<StringVal>          ::= {<UnicodeString>} {<QuotedString>}
                         {<NamedMacros>}
<NamedMacros>        ::= {"$(INF_VERSION)"} {"$(MODULE_NAME))"
<UiOpts>             ::= ["Optional" <MTS>] [<FfsAlignment>]
<VerFileType>        ::= {<VerFile>} {<VerString>}
<VerFile>            ::= "VERSION" <MTS> [<VerOpts>]
<VerOpts>            ::= ["Optional" <MTS>] [<BuildAlign>]
<VerString>          ::= "STRING" <Eq> <VerStringVal> <MTS> [<BuildAlign>]
<VerStringVal>       ::= {<UnicodeString>} {<QuotedString>}
                         {"$(INF_VERSION)"}
<BuildAlign>         ::= ["Optional" <MTS>] [<BuildArg>]
                         [<FfsAlignment>]
<BuildArg>           ::= "BUILD_NUM = $(BUILD_NUMBER)" <MTS>
<PdFileType>         ::= "PEI_DEPEX" <MTS> [<DpxAlign>]
<FvSec>              ::= "FV_IMAGE" <MTS> {<FvBin>} {<FvImageSection>}
<FvBin>              ::= "FV" <MTS> [<FfsAlignment>] [<FileOrExt>] <EOL>
<FvImgSection>       ::= "FV_IMAGE" <MTS> <FvImgArgs>
<FvImgArgs>          ::= "{" <EOL>
                         <MacroDefinition>*
                         [<FvAlignment>]
                         <FvAttributes>*
                         [<AprioriSection>]
                         <FileStatements>*
                         <TS> "}" <EOL>
<FvAlignment>        ::= [<TS> "FvBaseAddress" <Eq> <UINT64> <EOL>]
                         <TS> "FvAlignment" <Eq> <FvAlignmentValues> <EOL>
<FvAttributes>       ::= [<TS> "MEMORY_MAPPED" <Eq> <BoolType> <EOL>]
                         [<TS> "LOCK_CAP" <Eq> <BoolType> <EOL>]
                         [<TS> "LOCK_STATUS" <Eq> <BoolType> <EOL>]
                         [<TS> "WRITE_LOCK_CAP" <Eq> <BoolType> <EOL>]
                         [<TS> "WRITE_LOCK_STATUS" <Eq> <BoolType> <EOL>]
                         [<TS> "WRITE_ENABLED_CAP" <Eq> <BoolType> <EOL>]
                         [<TS> "WRITE_DISABLED_CAP" <Eq> <BoolType> <EOL>]
                         [<TS> "WRITE_STATUS" <Eq> <BoolType> <EOL>]
                         [<TS> "STICKY_WRITE" <Eq> <BoolType> <EOL>]
                         [<TS> "WRITE_POLICY_RELIABLE" <Eq> <BoolType> <EOL>]
                         [<TS> "READ_LOCK_CAP" <Eq> <BoolType> <EOL>]
                         [<TS> "READ_LOCK_STATUS" <Eq> <BoolType> <EOL>]
                         [<TS> "READ_ENABLED_CAP" <Eq> <BoolType> <EOL>]
                         [<TS> "READ_DISABLED_CAP" <Eq> <BoolType> <EOL>]
                         [<TS> "READ_STATUS" <Eq> <BoolType> <EOL>]
<PeiAprioriSection> ::= "APRIORI PEI" <MTS>
                         "{" <EOL>
                         <MacroDefinition>*
                         <FileStatements>* <TS> "}" <EOL>
<DxeAprioriSection> ::= "APRIORI DXE" <MTS>
                         "{" <EOL>
                         <MacroDefinition>*
```

```
                        <FileStatements>*
                        <TS> "}" <EOL>
```

Restrictions

FName

For BINARY ONLY content (`UEFI_DRIVER` and `UEFI_APPLICATION` .efi files) the file names specified in the `SECTION` element of this section must be relative to the directory identified by the `WORKSPACE` system environment variable (or relative to a directory listed in the `PACKAGES_PATH`).

Parameters

$(INF_VERSION)

This value refers to the `VERSION_STRING` element in the INF file's `[Defines]` section, not the `INF_VERSION` element, which is assigned based on the INF specification revision.

SUBTYPE_GUID

This is short hand notation refering to content that will be placed in a Section of type: `EFI_SECTION_FREEFORM_SUBTYPE_GUID` . A single

`EFI_SECTION_FREEFORM_SUBTYPE_GUID` section is permitted in an FFS File of type `EFI_FV_FILETYPE_FREEFORM`

RuleUiName

A unique single word identifier. The word `"BINARY"` is reserved; it is recommended that it be used for the rules that process INF modules that only contain binary content.

COMPRESS

Compression sections that use `PI_STD` compression do not have `PROCESSING_REQUIRED = TRUE` flag, it is only required for GUIDED sections.

User Interface (UI) entries

There are three possible methods for specifying a User Interface string. 1) Specify the string value in the FDF file, 2) specify a plain ASCII text file that has an extension of ".ui" or 3) specify a Unicode file with an extension of ".uni" that contains a single Unicode string.

Related Definitions

Target

Must match a target identifier in the EDK II tools_def.txt file - the first field, where fields are separated by the underscore character. Wildcard characters are not permitted.

TagName

Must match a tag name field in the EDK II tools_def.txt file - second field. Wildcard characters are not permitted

Example

```
[Rule.IA32.SEC]
  FILE SEC = $(NAMED_GUID) Fixed Align=32 |.efi

[Rule.Common.PEIM]
  FILE PEIM = $(NAMED_GUID) {
    TE TE |.te
    PEI_DEPEX PEI_DEPEX Optional |.Depex
```

```
    VERSION STRING    = "$(INF_VERSION)" Optional BUILD_NUM = $(BUILD_NUM)
    UI UNI_UI Optional | .uni
  }

[Rule.Common.PEIM.PE32]
  FILE PEIM = $(NAMED_GUID) {
    PEI_DEPEX PEI_DEPEX Optional | .dxs
    COMPRESS {
      PE32 PE32 |.efi
      VERSION UNI_VER Optional BUILD_NUM = $(BUILD_NUM) | .ver
      UI UI Optional | .ui
    }
  }
```

3.10 [VTF] Section

This describes the optional `[VTF]` section tag found in FDF files.

Summary

If VTF files will be created, they will be created in the `$(OUTPUT_DIRECTORY)/$(TARGET)_$(TAGNAME)/FV` directory using the values from the individual instance of the build tools. (Build tools get these values after parsing DSC, INF, `target.txt`, `tools_def.txt` files and command line options.)

The following sequence describes each component:

```
Name = Region,
       Type,
       Version,
       CheckSum_Flag,
       Path_of_Binary_File,
       Path_of_SYM_File,
       Preferred_Size;
```

Where,

Name:

Name of the component

Region:

Location in the firmware. Valid locations are:

```
PH - Protected Block region, merged towards the higher address
PL - Protected Block region, merged towards the lower address
H - Flashable region, merged towards the higher address
L - Flashable region, merged towards the lower address
F - First VTF File
N - Not in VTF File
S - Second VTF File
```

Type:

Component Type. Predefined values are:

```
0x00 : FIT Header entry
0x01 : PAL_B
0x02 - 0x0E : Reserved
0x0F : PAL_A
0x10 - 0x7E : OEM-defined
0x7F : Unused
```

Version:

Component Version number (XX.YY)

```
- major version number (decimal number, range of 0 to 99)
- minor version number (decimal number, range of 0 to 99)
```

Checksum_Flag:

Checksum Flag (equivalent to CV bit)

```
0 - Checksum Byte always equals 0, CV=0
1 - calculate Checksum Byte, CV=1
```

Checksum:

Byte sum of component + Checksum Byte = modulus 0x100

Path_of_Binary_File:

Path of the Binary file of the component

Path_of_SYM_File:

Path of the .SYM symbol file of the component

Preferred_Size:

User preferred component size, overrides actual component file size. Valid is equal or greater than the actual file size.

Prototype

```
<VTF>                ::= "[VTF" <Modifiers> "]" <EOL>
                         [<OptionStatement>]
                         <ComponentStatements>*
<Modifiers>          ::= "." <arch> "." <UiName> [<ArchList>]
<ArchList>           ::= "," <Arch>
<Arch>               ::= {"IA32"} {"X64"} {"IPF"}
<UiName>             ::= <Word>
<OptionStatement>    ::= <TS> "IA32_RST_BIN" <Eq> <Filename> <EOL>
<ComponentStatements> ::= <TS> "COMP_NAME" <Eq> <WORD> <EOL>
                         <TS> "COMP_LOC" <Eq> <Location> <EOL>
                         <TS> "COMP_TYPE" <Eq> <CompType> <EOL>
                         <TS> "COMP_VER" <Eq> <Version> <EOL>
                         <TS> "COMP_CS" <Eq> {"1"} {"0"} <EOL>
                         <TS> "COMP_BIN" <Eq> <BinFile> <EOL>
                         <TS> "COMP_SYM" <Eq> <SymFile> <EOL> <TS> "COMP_SIZE"
                         <Eq> <Size> <EOL>
<Location>           ::= {<FvUiName>} {"NONE"} {"None"} {"none"} [<FS>
                         <Region>]
<Region>             ::= {"F"} {"N"} {"S"} {"H"} {"L"} {"PH"} {"PL"}
<CompType>           ::= {"FIT"} {"PAL_B"} {"PAL_A"} {"OEM"} {<Byte>}
<Byte>               ::= "0x" <HexDigit>? <HexDigit>
<Version>            ::= {"-"} {<Major> "." <Minor>} {<BcdHex>}
<Major>              ::= [(0-9)](0-9)
<Minor>              ::= [(0-9)](0-9)
<BcdHex>             ::= "0x" <Major> (0-9) (0-9)
<BinFile>            ::= {"-"} {[<PATH>] <Filename>}
<SymFile>            ::= {"-"} {[<PATH>] <Filename>}
<Size>               ::= {"-"} {<Integer>} {<HexNumber>}
```

Restrictions

FName

All file specified paths are relative to the WORKSPACE directory (or a directory

listed in the PACKAGESPATH). In some cases, the tools will search well known paths for some files, for example, for FD filenames, the output will typically be located in the $(OUTPUT_DIRECTORY)/$(TARGET)$(TAGNAME)/FV directory.

Parameters

- Filename

If a filename is given, the file must have an extension for a binary type file, such as ".bin" or ".BIN". Filenames are case sensitive, so the correct case must be used for all filenames.

- Filename

If a filename is given, the file must have an extension for a symbol type file, such as ".sym" or ".SYM". Filenames are case sensitive, so the correct case must be used for all filenames.

Example

```
[VTF.IPF.MyBsf]
IA32_RST_BIN = IA32_RST.BIN

COMP_NAME = PAL_A          # Component Name
COMP_LOC  = FvRecovery | F # In the first VTF file
COMP_TYPE = 0xF            # Component Type (PAL_A=0x0F, defined in SAL Spec.)
COMP_VER  = 7.01           # Version will come from header of PAL_A binary
COMP_CS   = 1              # Checksum_Validity (CV bit)
COMP_BIN  = PAL_A_GEN.BIN  # Path of binary
COMP_SYM  = PAL_A_GEN.SYM  # Path of SYM symbol
COMP_SIZE = -              # Preferred component size in bytes

COMP_NAME = PAL_B     # Component Name
COMP_LOC  = F         # In the first VTF file
COMP_TYPE = 0x01      # Component Type (PAL_A=0x0F, defined in SAL Spec.)
COMP_VER  = -         # Version will come from header of PAL_A binary
COMP_CS   = 1         # Checksum_Validity (CV bit)
COMP_BIN  = PAL_B.BIN # Path of binary
COMP_SYM  = PAL_B.Sym # Path of SYM symbol
COMP_SIZE = -         # Preferred component size in bytes
```

3.11 PCI OptionRom Section

This is an optional section.

Summary

This section is used to specify the content of a PCI Option ROM container. A PCI Option ROM image may contain zero or more PCI ROM image files - binary only, and zero or more UEFI driver images, specified by either binary or INF files, that are to be packaged into a single Option ROM image. Additionally, support for a single EFI driver with both native (IA32, X64, IPF, etc). and EBC images in the same PCI Option ROM container is provided.

Conditional statements may be used anywhere within this section.

Prototype

```
<OptionRom>    ::= "[OptionRom" "." <DriverName> "]" <EOL> <Components>*
<DriverName>   ::= (a-zA-Z)(a-zA-Z0-9)*
<Components>   ::= {<InfComponent>} {<Binary>}
<InfComponent> ::= <TS> "INF" <MTS> <UseArch> <InfFile>
                   [<Overrides>] <EOL>
<UseArch>      ::= "USE" <Eq> <TargetArch> <MTS>
<TargetArch>   ::= <arch>
<InfFile>      ::= [<PATH>] <Word> ".inf"
<Overrides>    ::= <MTS> "{" <EOL>
                   [<TS> "PCI_VENDOR_ID" <Eq> <UINT16> <EOL>]
                   [<TS> "PCI_CLASS_CODE" <Eq> <UINT8> <EOL>]
                   [<TS> "PCI_DEVICE_ID" <Eq> <UINT16> <EOL>]
                   [<TS> "PCI_REVISION" <Eq> <UINT8> <EOL>]
                   [<TS> "PCI_COMPRESS" <Eq> <TrueFalse> <EOL>]
                   <TS> "}" <EOL>
<Binary>       ::= {<EfiBinary>} {<OtherBinary>}
<EfiBinary>    ::= <TS> "FILE" <MTS> "EFI" <EfiFileName>
                   [<Overrides>] <EOL>
<EfiFileName>  ::= <MTS> [<PATH>] <Word> {".efi"} {".EFI"} {".Efi"}
<OtherBinary>  ::= <TS> "FILE" <MTS> "BIN" <Filename> <EOL>
```

Restrictions

TargetArch

Only specific architectures are permitted - use of "common" or the wildcard character is prohibited.

Paths

For BINARY ONLY content (`UEFI_DRIVER` and `UEFI_APPLICATION` .efi files) the file names specified in `<EfiFileName>` of this section must be relative to the directory identified by the `WORKSPACE` system environment variable (or relative to a directory listed in the PACKAGESPATH system environment variable). In some cases, the tools will search well known paths for some files, for example, for FD filenames, the output will typically be located in the `$(OUTPUT_DIRECTORY)/ $(TARGET)$(TAGNAME)/FV` directory.

Related Definitions

DriverName

Specifies the name of the created PCI Option ROM image that will be placed in the build's FV directory.

USE

Specifies the architecture to use to create a PCI Option ROM.

Filename

Filenames must match the actual case of the file; three variations are shown for the .efi extension in the ENBF above.

Example

```
[OptionRom.AtapiPassThru]
  INF USE = IA32 OptionRomPkg/AtapiPassThruDxe/AtapiPassThruDxe.inf {
    PCI_REVISION = 0x0020
  }
  INF USE = EBC OptionRomPkg/AtapiPassThruDxe/AtapiPassThruDxe.inf
```

APPENDIX A NT32PKG FLASH DESCRIPTION FILE

This section provides a sample FDF using the Nt32Pkg/Nt32Pkg.fdf file.

Note: This file must NOT be used as is, as data structures and definitions do not exist.

```
## @file
# This is NT32 FDF file with UEFI HII features enabled
#
# 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.
#
######################################################################

######################################################################
#
# FD Section
# The [FD] Section is made up of the definition statements and a
# description of what goes into the Flash Device Image. Each FD section
# defines one flash "device" image. A flash device image may be one of
# the following: Removable media bootable image (like a boot floppy
# image,) an Option ROM image (that would be "flashed" into an add-in
# card,) a System "Flash" image (that would be burned into a system's
# flash) or an Update ("Capsule") image that will be used to update and
# existing system flash.
#
######################################################################
[FD.Nt32]
  # The base address of the FLASH Device.
  BaseAddress = 0x0|gEfiNt32PkgTokenSpaceGuid.PcdWinNtFdBaseAddress
  # The size in bytes of the FLASH Device
  Size      = 0x002a0000 ErasePolarity = 1
  BlockSize = 0x10000
  NumBlocks = 0x2a
  #
  #
  # Following are lists of FD Region layout which correspond to the
  # locations of different images within the flash device.
  #
  # Regions must be defined in ascending order and may not overlap.
  #
  # A Layout Region start with a eight digit hex offset (leading "0x"
  # required) followed by the pipe "|" character, followed by the size of
  # the region, also in hex with the leading "0x" characters. Like:
  # Offset|Size
  # PcdOffsetCName|PcdSizeCName
  # RegionType <FV, DATA, or FILE>
  #
  ######################################################################
  0x00000000|0x00280000
  gEfiNt32PkgTokenSpaceGuid.PcdWinNtFlashFvRecoveryBase|gEfiNt32PkgTokenSpaceGuid.PcdWinNtFlashFvRecoverySize
  FV = FvRecovery
  0x00280000|0x0000c000
  gEfiNt32PkgTokenSpaceGuid.PcdWinNtFlashNvStorageVariableBase|gEfiMdeModulePkgTokenSpaceGuid.PcdFlashNvStorageVariableSize
```

```
  #NV_VARIABLE_STORE
  DATA = {
  ## This is the EFI_FIRMWARE_VOLUME_HEADER
  # ZeroVector []
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  # FileSystemGuid: gEfiSystemNvDataFvGuid =
  # { 0xFFF12B8D, 0x7696, 0x4C8B,
  # { 0xA9, 0x85, 0x27, 0x47, 0x07, 0x5B, 0x4F, 0x50 }}
  0x8D, 0x2B, 0xF1, 0xFF, 0x96, 0x76, 0x8B, 0x4C,
  0xA9, 0x85, 0x27, 0x47, 0x07, 0x5B, 0x4F, 0x50,
  # FvLength: 0x20000
  0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00,
  #Signature "_FVH" #Attributes
  0x5f, 0x46, 0x56, 0x48, 0xff, 0xfe, 0x04, 0x00,
  #HeaderLength #CheckSum #ExtHeaderOffset #Reserved #Revision
  0x48, 0x00, 0x36, 0x09, 0x00, 0x00, 0x00, 0x02,
  #Blockmap[0]: 2 Blocks * 0x10000 Bytes / Block
  0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00,
  #Blockmap[1]: End
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  ## This is the VARIABLE_STORE_HEADER`
  #Signature: gEfiVariableGuid =
  # { 0xddcf3616, 0x3275, 0x4164,
  # { 0x98, 0xb6, 0xfe, 0x85, 0x70, 0x7f, 0xfe, 0x7d }}
  0x16, 0x36, 0xcf, 0xdd, 0x75, 0x32, 0x64, 0x41,
  0x98, 0xb6, 0xfe, 0x85, 0x70, 0x7f, 0xfe, 0x7d,
  #Size: 0xc000
  # (gEfiMdeModulePkgTokenSpaceGuid.PcdFlashNvStorageVariableSize) -
  # 0x48 (size of EFI_FIRMWARE_VOLUME_HEADER) = 0xBFB8
  # This can speed up the Variable Dispatch a bit.
  0xB8, 0xBF, 0x00, 0x00,
  #FORMATTED: 0x5A #HEALTHY: 0xFE #Reserved: UINT16 #Reserved1: UINT32
  0x5A, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
}
0x0028c000 | 0x00002000
#NV_EVENT_LOG
gEfiNt32PkgTokenSpaceGuid.PcdWinNtFlashNvStorageEventLogBase|gEfiNt32PkgTokenSpaceGuid.PcdWinNtFlashNvStorageEventLogSize

0x0028e000 | 0x00002000
gEfiNt32PkgTokenSpaceGuid.PcdWinNtFlashNvStorageFtwWorkingBase|gEfiMdeModulePkgTokenSpaceGuid.PcdFlashNvStorageFtwWorkingSize

#NV_FTW_WORKING
DATA = {
  # EFI_FAULT_TOLERANT_WORKING_BLOCK_HEADER->Signature =
  # gEfiSystemNvDataFvGuid = { 0xFFF12B8D, 0x7696, 0x4C8B,
  # { 0xA9, 0x85, 0x27, 0x47, 0x07, 0x5B, 0x4F, 0x50 }}
  0x8D, 0x2B, 0xF1, 0xFF, 0x96, 0x76, 0x8B, 0x4C,
  0xA9, 0x85, 0x27, 0x47, 0x07, 0x5B, 0x4F, 0x50,
  # Crc:UINT32
  # WorkingBlockValid:1, WorkingBlockInvalid:1, Reserved
  #
  0x77, 0x13, 0x9B, 0xD7, 0xFE, 0xFF, 0xFF, 0xFF,
  # WriteQueueSize: UINT64
  0xE0, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
}

0x00290000 | 0x00010000
#NV_FTW_SPARE
gEfiNt32PkgTokenSpaceGuid.PcdWinNtFlashNvStorageFtwSpareBase|gEfiMdeModulePkgTokenSpaceGuid.PcdFlashNvStorageFtwSpareSize

################################################################################
#
# FV Section
#
# [FV] section is used to define what components or modules are placed
# within a flash device file. This section also defines order the
# components and modules are positioned within the image. The [FV] # section consists of define statements, set statements and
 module # statements.
#
################################################################################
[FV.FvRecovery]
```

```
FvBaseAddress = 0x0 # FV Base Address for the Backup copy of FV
FvAlignment   = 16  # FV alignment

#FV attributes setting.
ERASE_POLARITY     = 1
MEMORY_MAPPED      = TRUE
STICKY_WRITE       = TRUE
LOCK_CAP           = TRUE
LOCK_STATUS        = TRUE
WRITE_DISABLED_CAP = TRUE
WRITE_ENABLED_CAP  = TRUE
WRITE_STATUS       = TRUE
WRITE_LOCK_CAP     = TRUE
WRITE_LOCK_STATUS  = TRUE
READ_DISABLED_CAP  = TRUE
READ_ENABLED_CAP   = TRUE
READ_STATUS        = TRUE
READ_LOCK_CAP      = TRUE
READ_LOCK_STATUS   = TRUE
FvNameGuid         = 6D99E806-3D38-42c2-A095-5F4300BFD7DC

##############################################################################
#
# The INF statements point to EDK component and EDK II module INF files,
# which will be placed into this FV image.
# Parsing tools will scan the INF file to determine the type of component
# or module.
# The component or module type is used to reference the standard rules
# defined elsewhere in the FDF file.
#
# The format for INF statements is:
# INF $(PathAndInfFileName)
#
##############################################################################
  ##
  # PEI Phase modules
  ##
  # PEI Apriori file example, more PEIM module added later.
  ##

DEFINE MdeModUni = MdeModulePkg/Universal
DEFINE WNOEMHOOK = Nt32Pkg/WinNtOemHookStatusCodeHandlerPei

DEFINE RSCR_P    = ReportStatusCodeRouter/Pei

APRIORI PEI {
  INF $(MdeModUni)/PCD/Pei/Pcd.inf
  INF $(MdeModUni)/$(RSCR_P)/ReportStatusCodeRouterPei.inf
  INF $(MdeModUni)/StatusCodeHandler/Pei/StatusCodeHandlerPei.inf
  INF $(WNOEMHOOK)/WinNtOemHookStatusCodeHandlerPei.inf
  }

APRIORI DXE {
  INF MdeModulePkg/Universal/PCD/Dxe/Pcd.inf
  INF Nt32Pkg/MetronomeDxe/MetronomeDxe.inf
  }

INF MdeModulePkg/Core/Pei/PeiMain.inf
INF MdeModulePkg/Universal/PCD/Pei/Pcd.inf
INF $(MdeModUni)/$(RSCR_P)/ReportStatusCodeRouterPei.inf
INF $(MdeModUni)/StatusCodeHandler/Pei/StatusCodeHandlerPei.inf
INF $(WINOEMHOOK)/WinNtOemHookStatusCodeHandlerPei.inf
INF Nt32Pkg/BootModePei/BootModePei.inf
INF Nt32Pkg/StallPei/StallPei.inf
INF Nt32Pkg/WinNtFlashMapPei/WinNtFlashMapPei.inf
INF Nt32Pkg/WinNtAutoScanPei/WinNtAutoScanPei.inf
INF Nt32Pkg/WinNtFirmwareVolumePei/WinNtFirmwareVolumePei.inf
INF MdeModulePkg/Universal/Variable/Pei/VariablePei.inf
INF Nt32Pkg/WinNtThunkPPIToProtocolPei/WinNtThunkPPIToProtocolPei.inf
INF MdeModulePkg/Core/DxeIplPeim/DxeIpl.inf

  ##
```

```
  # DXE Phase modules
  ##
INF MdeModulePkg/Core/Dxe/DxeMain.inf
INF MdeModulePkg/Universal/PCD/Dxe/Pcd.inf
INF Nt32Pkg/MetronomeDxe/MetronomeDxe.inf
INF Nt32Pkg/RealTimeClockRuntimeDxe/RealTimeClockRuntimeDxe.inf
INF Nt32Pkg/ResetRuntimeDxe/ResetRuntimeDxe.inf
INF MdeModulePkg/Core/RuntimeDxe/RuntimeDxe.inf
INF Nt32Pkg/FvbServicesRuntimeDxe/FvbServicesRuntimeDxe.inf
INF MdeModulePkg/Universal/SecurityStubDxe/SecurityStubDxe.inf
INF MdeModulePkg/Universal/SmbiosDxe/SmbiosDxe.inf
INF MdeModulePkg/Universal/EbcDxe/EbcDxe.inf
INF $(MdeModUni)/MemoryTest/NullMemoryTestDxe/NullMemoryTestDxe.inf
INF MdeModulePkg/Universal/HiiDatabaseDxe/HiiDatabaseDxe.inf
INF Nt32Pkg/WinNtThunkDxe/WinNtThunkDxe.inf
INF Nt32Pkg/CpuRuntimeDxe/CpuRuntimeDxe.inf
INF IntelFrameworkModulePkg/Universal/BdsDxe/BdsDxe.inf
INF $(MdeModUni)/FaultTolerantWriteDxe/FaultTolerantWriteDxe.inf
INF Nt32Pkg/MiscSubClassPlatformDxe/MiscSubClassPlatformDxe.inf
INF Nt32Pkg/TimerDxe/TimerDxe.inf

DEFINE RSCR_RD = ReportStatusCodeRouter/RuntimeDxe
INF $(MdeModUni)/$(RSCR_RD)/ReportStatusCodeRouterRuntimeDxe.inf

DEFINE SCH_RD = StatusCodeHandler/RuntimeDxe
INF $(MdeModUni)/$(SCH_RD)/StatusCodeHandlerRuntimeDxe.inf

DEFINE NTWINNTOEMHOOK = Nt32Pkg/WinNtOemHookStatusCodeHandlerDxe
INF $(NTWINNTOEMHOOK)/WinNtOemHookStatusCodeHandlerDxe.inf

INF MdeModulePkg/Universal/Variable/RuntimeDxe/VariableRuntimeDxe.inf
INF MdeModulePkg/Universal/WatchdogTimerDxe/WatchdogTimer.inf

DEFINE MCRD = MonotonicCounterRuntimeDxe
INF $(MdeModUni)/$(MCRD)/MonotonicCounterRuntimeDxe.inf

INF MdeModulePkg/Universal/CapsuleRuntimeDxe/CapsuleRuntimeDxe.inf
INF MdeModulePkg/Universal/Console/ConPlatformDxe/ConPlatformDxe.inf
INF MdeModulePkg/Universal/Console/ConSplitterDxe/ConSplitterDxe.inf
INF $(MdeModUni)/Console/GraphicsConsoleDxe/GraphicsConsoleDxe.inf
INF MdeModulePkg/Universal/Console/TerminalDxe/TerminalDxe.inf
INF MdeModulePkg/Universal/DevicePathDxe/DevicePathDxe.inf
INF MdeModulePkg/Universal/Disk/DiskIoDxe/DiskIoDxe.inf
INF MdeModulePkg/Universal/Disk/PartitionDxe/PartitionDxe.inf
INF MdeModulePkg/Universal/SetupBrowserDxe/SetupBrowserDxe.inf
INF MdeModulePkg/Universal/PrintDxe/PrintDxe.inf

DEFINE DUC = Disk/UnicodeCollation
INF RuleOverride = TIANOCOMPRESSED $(MdeModUni)/$(DUC)/EnglishDxe/EnglishDxe.inf

INF MdeModulePkg/Bus/Pci/PciBusDxe/PciBusDxe.inf
INF MdeModulePkg/Bus/Scsi/ScsiBusDxe/ScsiBusDxe.inf
INF MdeModulePkg/Bus/Scsi/ScsiDiskDxe/ScsiDiskDxe.inf
INF IntelFrameworkModulePkg/Bus/Pci/IdeBusDxe/IdeBusDxe.inf
INF Nt32Pkg/WinNtBusDriverDxe/WinNtBusDriverDxe.inf
INF Nt32Pkg/WinNtBlockIoDxe/WinNtBlockIoDxe.inf
INF Nt32Pkg/WinNtSerialIoDxe/WinNtSerialIoDxe.inf
INF Nt32Pkg/WinNtGopDxe/WinNtGopDxe.inf
INF Nt32Pkg/WinNtSimpleFileSystemDxe/WinNtSimpleFileSystemDxe.inf
INF $(MdeModUni)/PlatformDriOverrideDxe/PlatformDriOverrideDxe.inf
INF MdeModulePkg/Universal/DriverSampleDxe/DriverSampleDxe.inf

INF MdeModulePkg/Universal/Network/DpcDxe/DpcDxe.inf
INF MdeModulePkg/Universal/Network/ArpDxe/ArpDxe.inf
INF MdeModulePkg/Universal/Network/Dhcp4Dxe/Dhcp4Dxe.inf
INF MdeModulePkg/Universal/Network/Ip4ConfigDxe/Ip4ConfigDxe.inf
INF MdeModulePkg/Universal/Network/Ip4Dxe/Ip4Dxe.inf
INF MdeModulePkg/Universal/Network/MnpDxe/MnpDxe.inf
INF MdeModulePkg/Universal/Network/VlanConfigDxe/VlanConfigDxe.inf
INF MdeModulePkg/Universal/Network/Mtftp4Dxe/Mtftp4Dxe.inf
INF MdeModulePkg/Universal/Network/Tcp4Dxe/Tcp4Dxe.inf
INF MdeModulePkg/Universal/Network/Udp4Dxe/Udp4Dxe.inf
```

```
INF Nt32Pkg/SnpNt32Dxe/SnpNt32Dxe.inf
INF MdeModulePkg/Universal/Network/UefiPxeBcDxe/UefiPxeBcDxe.inf
INF MdeModulePkg/Universal/Network/IScsiDxe/IScsiDxe.inf

######################################################################
#
# FILE statements are provided so that a platform integrator can include
# complete EFI FFS files, as well as a method for constructing FFS files
# using curly "{}" brace scoping. The following three FILEs are
# for binary shell, binary fat and logo module.
#
######################################################################
FILE APPLICATION = PCD(gEfiIntelFrameworkModulePkgTokenSpaceGuid.PcdShellFile) {
  SECTION PE32 = EdkShellBinPkg/FullShell/Ia32/Shell_Full.efi
  }
FILE DRIVER = 961578FE-B6B7-44c3-AF35-6BC705CD2B1F {
  SECTION PE32 = FatBinPkg/EnhancedFatDxe/Ia32/Fat.efi
  }
FILE FREEFORM = PCD(gEfiIntelFrameworkModulePkgTokenSpaceGuid.PcdLogoFile) {
  SECTION RAW = MdeModulePkg/Logo/Logo.bmp
  }

######################################################################
#
# Rules are use with the [FV] section's module INF type to define
# how an FFS file is created for a given INF file. The following Rule are
# the default rules for the different module type. User can add the
# customized rules to define the content of the FFS file.
#
######################################################################

######################################################################
# Example of a DXE_DRIVER FFS file with a Checksum encapsulation section
#
#
#
#[Rule.Common.DXE_DRIVER]
#   FILE DRIVER = $(NAMED_GUID) {
#     DXE_DEPEX DXE_DEPEX Optional $(INF_OUTPUT)/$(MODULE_NAME).depex
#     COMPRESS PI_STD {
#       GUIDED {
#         PE32    PE32                    $(INF_OUTPUT)/$(MODULE_NAME).efi
#         UI      STRING="$(MODULE_NAME)" Optional
#         VERSION STRING="$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
#       }
#     }
#   }
#
######################################################################

[Rule.Common.PEI_CORE]
  FILE PEI_CORE = $(NAMED_GUID) {
    PE32    PE32 Align=4K             $(INF_OUTPUT)/$(MODULE_NAME).efi
    UI      STRING = "$(MODULE_NAME)" Optional
    VERSION STRING = "$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
  }

[Rule.Common.PEIM]
  FILE PEIM = $(NAMED_GUID){
    PEI_DEPEX PEI_DEPEX                Optional $(INF_OUTPUT)/$(MODULE_NAME).depex
    PE32      PE32 Align=4K            $(INF_OUTPUT)/$(MODULE_NAME).efi
    UI        STRING = "$(MODULE_NAME)" Optional
    VERSION   STRING = "$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
  }

[Rule.Common.DXE_CORE]
  FILE DXE_CORE = $(NAMED_GUID) {
    COMPRESS PI_STD {
      PE32    PE32                      $(INF_OUTPUT)/$(MODULE_NAME).efi
      UI      STRING = "$(MODULE_NAME)" Optional
      VERSION STRING = "$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
    }
```

```
  }

[Rule.Common.UEFI_DRIVER]
  FILE DRIVER = $(NAMED_GUID) {
    DXE_DEPEX DXE_DEPEX Optional $(INF_OUTPUT)/$(MODULE_NAME).depex
    COMPRESS PI_STD {
      GUIDED {
        PE32    PE32                      $(INF_OUTPUT)/$(MODULE_NAME).efi
        UI      STRING = "$(MODULE_NAME)" Optional
        VERSION STRING = "$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
      }
    }
  }

[Rule.Common.UEFI_DRIVER.TIANOCOMPRESSED]
  FILE DRIVER = $(NAMED_GUID) {
    DXE_DEPEX DXE_DEPEX Optional $(INF_OUTPUT)/$(MODULE_NAME).depex
    GUIDED A31280AD-481E-41B6-95E8-127F4C984779 PROCESSING_REQUIRED = TRUE {
      PE32    PE32                     $(INF_OUTPUT)/$(MODULE_NAME).efi
      UI      STRING = "$(MODULE_NAME)" Optional
      VERSION STRING = "$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
    }
  }

[Rule.Common.DXE_DRIVER]
  FILE DRIVER = $(NAMED_GUID) {
    DXE_DEPEX   DXE_DEPEX Optional        $(INF_OUTPUT)/$(MODULE_NAME).depex
    COMPRESS PI_STD {
      GUIDED {
        PE32    PE32                      $(INF_OUTPUT)/$(MODULE_NAME).efi
        UI      STRING = "$(MODULE_NAME)" Optional
        VERSION STRING = "$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
      }
    }
  }

[Rule.Common.DXE_RUNTIME_DRIVER]
  FILE DRIVER = $(NAMED_GUID) {
    DXE_DEPEX   DXE_DEPEX Optional        $(INF_OUTPUT)/$(MODULE_NAME).depex
    COMPRESS PI_STD {
      GUIDED {
        PE32    PE32                      $(INF_OUTPUT)/$(MODULE_NAME).efi
        UI      STRING = "$(MODULE_NAME)" Optional
        VERSION STRING = "$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
      }
    }
  }

[Rule.Common.UEFI_APPLICATION]
  FILE APPLICATION = $(NAMED_GUID) {
    COMPRESS PI_STD {
      GUIDED {
        PE32    PE32                      $(INF_OUTPUT)/$(MODULE_NAME).efi
        UI      STRING = "$(MODULE_NAME)" Optional
        VERSION STRING = "$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
      }
    }
  }
```

APPENDIX B COMMON ERROR MESSAGES

Standard build tools must throw error messages, halting the build. Warning messages may be emitted from build tools, unless a quiet flag has been set on the command-line.

B.1 [FD] Syntax Errors:

Missing required Token statements - report line number and file name.

Size of the FV (%s) is larger than the Region Size 0x%X specified.

The named FV (%s) is not listed in the FDF file.

Size of the DATA is larger than the region size specified.

Size of the FILE (%s) is larger than the region size specified.

The region at Offset 0x%X cannot fit into Block array with BlockSize %X.

Region: %s is not in the FD address scope.

B.2 [FV] Syntax Errors:

Missing required Token statements - report line number and file name.

Incorrect alignment in the FV.

B.3 [CAPSULE] Syntax Errors:

No capsule specific errors, as all capsule errors are captured under other parser errors.

B.4 [Rule] Syntax Errors:

Unable to find rule for INF %s.

Module built under different architectures, unable to determine which module to use.

APPENDIX C REPORTS

The following reports could be generated by either usage enhancement tools or the build tools. Refer to the EDK II Build Specification for the description of reports generated by the EDK II build system.