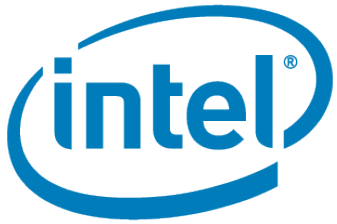

EDK II Build Decoded

Data Center Platform Applications Engineering

WW11 2011

Revision 0.0.2

EDK II Build Decoded

There are many EDK II specs to read to obtain enough knowledge to be able to write EDK II code. It takes even more time to understand how the various files (*DSC/DEC/INF/FDF*, etc) and data (*GUIDs, PCDs*, etc) are declared in the files so that one knows why certain things are done, as opposed to “I do this to make it work, but I don’t know why”.

These slides are supplemental information to the Intel EDK training material. They are intended to explain some of the concepts in a detailed manner so that the developer may obtain a more complete knowledge about EDK II builds, file types, data types, and how they are connected – to “get there quicker” so time can be spent writing and debugging code.

build.exe

- *edksetup.bat* script sets up configuration files *target.txt* and *tools_def.txt* in the **%WORKSPACE%\Conf** directory
- *build.exe* reads build options from *Conf\target.txt*
- Commonly used command line switches:
  - -h for *help*
  - -j to capture build output to specified log file
  - -t to override defined *TOOL_CHAIN_TAG*
  - -a to override defined *TARGET_ARCH*
  - -p to override defined *ACTIVE_PLATFORM*
- Command line switches are used to override parameters in *Conf\target.txt* so that editing it isn't required; one can download project tree and compile based on their needs and specific tools installed
- *TOOL_CHAIN_CONF* defines file which has locations of compiler executables; refer to *Conf\tools_def.txt* and search for "*Supported Tool Chains*" for list of valid values for *TOOL_CHAIN_TAG*
- *Conf\tools_def.txt* documents format of tool chain tags; *TARGET_TOOLCHAIN_ARCH_COMMANDTYPE_ATTRIBUTE*; can be used in package *DSC* to override compiler flags, For Example: different flags for DEBUG and RELEASE builds, DEBUG_*_*_*_* = <some flags> and RELEASE_*_*_*_* = <other flags>
- == is replace, = is append

build.exe (cont.)

- Windows 32-bit default path for all Visual Studio* versions is **C:\Program Files**.
- Windows 64-bit default path for all Visual Studio versions is **C:\Program Files (x86)**.
- 64-bit *OS TOOL_CHAIN_TAG* to use VS2008 is VS2008x86 if using Intel ASL compiler, or VS2008x86xASL if using Microsoft ASL compiler; these are documented in `Conf\tools_def.txt` under "*Supported Tool Chains*"
- Build tool takes a DSC file as input, which describes the package to build.
- Build output is in folder specified in the DSC path defined by *OUTPUT_DIRECTORY; TARGET* + *"_"* + *TOOL_CHAIN_TAG* are appended to this for final output path
  - For Example: **OUTPUT_DIRECTORY=Build/MyPkg, TARGET=DEBUG, TOOL_CHAIN_TAG=VS2008**; output files are created in **Build/MyPkg/DEBUG_VS2008**
  - If *TARGET_ARCH* of IA32 was specified, there will be an IA32 folder with output files
  - If *TARGET_ARCH* of X64 was specified, there will be an X64 folder with output files
  - **FV** folder is where the `FD` file is created; `*.FD` filename will be equal to name specified in *FDF*
- For more info on build options, refer to *EDK II Build spec*
  - *Chapter 5.2* discusses `tools_def.txt` format
  - *Chapter 5.3* discusses `target.txt` format
- *For more info on Macros and Conditional Grammar, refer to EDK II DSC/FDF Specs.*
  - *Chapter 2.2.6* discusses DEFINE keyword for creating variables, and $(VAR) for dereferencing variables created with DEFINE
  - *DSC 2.2.7 and FDF Chapter 2.2.8* discuss conditional grammar

build.exe (cont.)

- The scope of `build.exe` is to build only the BIOS FW image. If there are other components or tasks that need to be built or compiled to complete the platform image, the build process may require Pre and Post build script files. Pre and Post build scripts can determine where various tools are through parsing or using the following variables initialized by running the `edksetup.bat` script:
  - **%WORKSPACE%**
  - **%EDK_TOOLS_PATH%**
- The goal of `build.exe` is to build what is described in the DSC and/or FDF data meta files. Typically, this is a final BIOS image.
- If other firmware binaries are needed and there is required processing of these, such as concatenation, there will need to be script files that determine where these images are located.
- It is currently an issue for Pre and Post build scripts to determine the output path of `build.exe`. A solution is being worked on for `build.exe` to communicate the parameters it uses so Pre and Post build script files can be aware of all `build.exe` parameters.

Visual Studio tips

- Visual Studio* (VS) comes with a tool to generate GUIDs called *GuidGen*, but the installation doesn't create a shortcut to it. For Visual Studio* 2008, create a shortcut to **"C:\Program Files (x86)\Microsoft Visual Studio 9.0\Common7\Tools\guidgen.exe"**.
- Visual Studio* 2003 .NET, Visual Studio* 2005, 2008 and 2010 create system environment variables when installed. If one desires a build process that detects installed versions of VS and uses the latest one found so that editing `Conf\target.txt` isn't required, then the following environment variables can be checked, and the build `-t` flag can be set to indicate the VS version to use.
- VS 2003 .NET environment variable: *VS71COMNTOOLS*
- VS 2005 environment variable: *VS80COMNTOOLS*
- VS 2008 environment variable: *VS90COMNTOOLS*
- VS 2010 environment variable: *VS100COMNTOOLS*

Visual Studio Tips (cont.)

- Batch file to check for VS 2010, 2008, 2005, then 2003 (assumes user Operating System (OS) is *Windows XP* 32-bit*)

  If defined *VS100COMNTOOLS* `( set TOOL_CHAIN_TAG=VS2010; call %VS100COMNTOOLS%\vsvars32.bat; goto build`)

  If defined *VS90COMNTOOLS* `( set TOOL_CHAIN_TAG=VS2008; call %VS90COMNTOOLS%\vsvars32.bat; goto build`)

  If defined *VS80COMNTOOLS* (`set TOOL_CHAIN_TAG=VS2005; call %VS80COMNTOOLS%\vsvars32.bat; goto build`)

  If defined *VS71COMNTOOLS* (`set TOOL_CHAIN_TAG=VS2003; call %VS71COMNTOOLS%\vsvars32.bat; goto build`)

  :error

  Echo ERROR: no Visual Studio version found

  Goto end

  :build

  build `-t` *%TOOL_CHAIN_TAG%* ……..

  :end

FDF Format

- *FDF* stands for *Flash Description File*; `.FD` file stands for *Flash Device*
- The *FDF* file to use is specified in *DSC* file

```
FLASH_DEFINITION = EmulatorPkg/EmulatorPkg.fdf
```

- *FDF* parsing starts with `[FD.xxx]` section; `.xxx` is the name; if no name is specified, then *PLATFORM_NAME* listed in the *DSC* file will be used for the name
  - Section specifies flash size and base address
- PCDs can be set with `SET` command, but not defined; see PCD slides for more information on PCDs
- The `[FD..]` section must define at least one `[FV..]` section. Typically these would be `[FV.Recovery]` for *PEI* and *SEC*, and `[FV.Main]` for BIOS. There are usually other sections for *NVRAM*, *Vital Product Data* (*VPD*), etc.
- See *EDK II FDF Spec Chapter 2.3*

FDF Format (cont.)

- *FDF spec Chapter 2.3.4* discusses the FD Region Layout
- Format is `Offset|Size ? [RegionType]`
- For Example:

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

- Specifying `RegionType` is optional; if not specified, it implies this region should not be touched. If specified, the `RegionType` must be `FV`, `DATA`, or `FILE`.
- The region is type FV and details are specified in the `[FV.FvMain]` section, such as attributes and files that populate the region.
- PCDs are set to Offset and Size values in same format
  - gTokenSpaceGuid.PcdFlashFvMainBaseAddress is assigned Offset value 0x000000
  - gTokenSpaceGuid.PcdFlashFvMainSize is assigned Size values 0x0C0000
  - This is a shortcut to using SET command for each PCD, and less work because the value is changed only at Offset|Value, instead of Offset|Value and 2 SET commands

FDF Format (cont.)

- The following example sets microcode base and size in the FD:

```
0x002A0000|0x00040000
gPlatformTokenSpaceGuid.PcdFlashNvStorageMicrocodeBase|gPlatformTokenSpaceGuid.PcdFlashNvStorageMicrocodeSize
FV = MICROCODE_FV
```

- The `[FV.MICROCODE_FV]` section populates region with file

```
FILE RAW = 197DB236-F856-4924-90F8-CDF12FB875F3 {
$(OUTPUT_DIRECTORY)/$(TARGET)_$(TOOL_CHAIN_TAG)/X64/Microcode.bin}
```

- `SECTION GUIDED` allows for another tool to process the section.
  - See *EDK II Build spec Chapter 2.6.11.1*
  - `Conf\tools_def.txt` is checked for a GUID that matches the `SECTION GUIDED`
  - If match found, the tool for the matching GUID is executed on the data
  - There must be an associated library in firmware that is specific to the tool
  - For Example: LZMA compression
    - Tool GUID: EE4E5898-3914-4259-9D6E-DC7BD79403CF
    - Tool used by the build: *LzmaCompress*
    - Library class: *LzmaCustomDecompressLib*
    - Library Instances:

    EmbeddedPkg/Library/LzmaHobCustomDecompressLib/LzmaHobCustomDecompressLib.inf
    IntelFrameworkModulePkg/Library/LzmaCustomDecompressLib/LzmaArchCustomDecompressLib.inf

DSC Format

- Platform Description file.
- Platform driver and application *INFs* must be listed in `[Components]` section.
- List libraries that your application or driver links to in the `[LibraryClasses]` section by specifying a library class name + "|" + the library instance *INF* file name.
  - If an *INF* has a `LIBRARY_CLASS` line, then the *module* (described by the INF) produces a library. The first word after the = is the class name of the Library. Also on the line, following the "|" character, is a list of the module types the library instance will support.
    - Different library instances may provide the same functionality for a library class, using different code.
    - Additionally they may be coded to support only specific module types. For example, there may be one instance of a library that supports only `DXE_DRIVER` modules, while another instance of a library may only support PEIMs.
  - The *DSC* binds the library instance to the library class used during build.
  - The build system will automatically bind library instances by module type, so some library classes may have multiple library instances, with each instance supporting different module types.
    - Where there is a conflict, as in two library instances supporting exactly the same module types, then the last library instance listed will be link to the module.
- A UEFI application must have an *INF* and must also be listed in the `[Components]` section.
  - It should list library class names it must link to in a `[LibraryClasses]` section.
  - The library class names (to library instance bindings) in the DSC must match the library class name to library instances specified in the INF `[LibraryClasses]` section.

DSC Format (cont.)

- In the following example in the DSC, two different library instances provide the implementation of the library APIs for the *PcdLib* library class.
  - `[LibraryClasses]`

    `PcdLib|MdePkg/Library/DxePcdLib/DxePcdLib.inf`

    `PcdLib|MdePkg/Library/BasePcdLibNull/BasePcdLibNull.inf`
  - The first instance does not support PEIM or SEC module types, and supports `DXE_CORE`, `DXE_DRIVER`, `DXE_RUNTIME_DRIVER`, `DXE_SAL_DRIVER`, `DXE_SMM_DRIVER`, `SMM_CORE`, `UEFI_APPLICATION` and `UEFI_DRIVER` module types. The library may be linked to the modules, however additional overrides can be specified
  - The second instance will be linked against modules of type PEIM or SEC
- The globally defined library instances, that satisfies a module's requirement for a Library Class, may be overridden by instances specified in either an architectural section, i.e., `[LibraryClasses.IA32]` or module type section, as in `[LibraryClasses.Common.DXE_DRIVER]`.
- They may also be overridden for an individual module listed in the `[Components]` section.

DSC Format (cont.)

- PCD values can be overridden.
  - Section names are `PcdsFixedAtBuild,PcdsFeatureFlag,PcdsPatchableInModule,PcdsDynamic, PcdsDynamicEx`
  - See *DSC spec Chapter 2.8*
  - `[PcdsFixedAtBuild]` # if you want to override *DEC* value
  - `  gMyGuid.foo|0xBEEF1234`
- Section names not followed by `.<arch>` is the same as `.common`
  - `[PcdsFixedAtBuild]` and `[PcdsFixedAtBuild.common]` are the same
- If you experience build errors and you think there should be none, delete your **BUILD** folder and run it again. There may be a bug in `build.exe` which isn't picking up changes you made in source files between BUILD invocations.
  - You may also delete the directory %WORKSPACE%\Conf\.cache which is used to check for changes to the meta-data files.

DSC Format (cont.)

- How do you know which PCD type to use?
  - **PcdsFeatureFlag**: PCD is BOOLEAN value and is set at build time in the *DSC* file and cannot be modified at runtime. It behaves like a `CONST` global variable in the module's *PE/COFF* image. These are also used in the module's code, and allows compilers to optimize out content that is not needed.
  - **PcdsFixedAtBuild**: PCD is set at build time in the *DSC* file and cannot be modified at runtime. It behaves like a `CONST` global variable in the module's *PE/COFF* image.
  - **PcdsPatchableInModule**: PCD power on default value is set at build time in the *DSC* files. The PCD value can be modified at runtime within the scope of a single module. It behaves like a normal global variable in the module's *PE/COFF* image. The PE32 (.efi) image file can have the value changed by external tools without rebuilding the module.
  - **PcdsDynamicDefault**: PCD power on default value is set at build time in the *DSC* file. The *PCD* value is `R/W` volatile and globally scoped to the entire platform, so if one module does a *Set*, another module will see the new value on the next Get.
  - **PcdsDynamic**: A synonym for `PcdsDynamicDefault`.
  - **PcdsDynamicExDefault**: PCD `power on default` value is set at build time in the *DSC* file. The PCD value is `R/W` volatile and globally scoped to the entire platform, so if one module does a Set, another module will see the new value on the next Get. The Ex PCD types are accessed with the GUID + TokenNumber, so the code generated is slightly larger than the non Ex styles. The Ex styles are required when modules are provided as binaries.
  - **PcdsDynamicEx**: A synonym for `PcdsDynamicExDefault`.

DSC Format (cont.)

- How do you know which PCD type to use?
  - **PcdsDynamicVpd** : `PCD power on default` value is set at build time in the *DSC* file. The PCD value is `RO` and globally scoped to the entire platform. This allows the PCD value to be patched in a single location in the FLASH image before burning the image into the FLASH device. Good for things like system *GUID, MAC address, product names*, etc.
  - **PcdsDynamicExVpd**: `PCD power on default` value is set at build time in the DSC file. The PCD value is `RO` and globally scoped to the entire platform. This allows the PCD value to be patched in a single location in the FLASH image before burning the image to the FLASH device. Good for things like system *GUID, MAC address, product names*, etc. The Ex PCD types are accessed with the *GUID* + *TokenNumber*, so the code generated is slightly larger than the non Ex styles. The Ex styles are required when modules are provided as binaries.
  - **PcdsDynamicHii**: PCD default value is set at build time in the *DSC* file. The PCD value is `R/W` and non-volatile and globally scoped to the entire platform. These PCDs are stored internally into *EFI Variables*, so values set during one boot are preserved for the next boot.
  - **PcdsDynamicExHii**: PCD default value is set at build time in the *DSC* file. The PCD value is `R/W` and non-volatile and globally scoped to the entire platform. These PCDs are stored internally into *EFI Variables*, so values set during one boot are preserved for the next boot. The Ex PCD types are accessed with the *GUID* + *TokenNumber*, so the code generated is slightly larger than the non Ex styles. The Ex styles are required when modules are provided as binaries.

DEC Format

- PCDs are declared in DEC files, and can be set anywhere else
  - See *DEC spec Chapter 3.10* for PCD grammar
- `TokenSpaceGuidCName` must be a *GUID* declared under `[Guids]` section
- For Example: The *DEC* file has

```
[Guids]
gPlatform1TokenSpaceGuid = { 0x07dfa0d2, 0x2ac5, 0x4cab, { 0xac, 0x14,
0x30, 0x5c, 0x62, 0x48, 0x87, 0xe4 }}
gPlatform2TokenSpaceGuid = { 0x17dfa0d2, 0x2ac5, 0x4cab, { 0xac, 0x14,
0x30, 0x5c, 0x62, 0x48, 0x87, 0xe4 }}
[PcdsFixedAtBuild]
gPlatform1TokenSpaceGuid.Pcd1|0xBEEF0000|UINT32|0x00000000
gPlatform1TokenSpaceGuid.Pcd2|0x0000BEEF|UINT32|0x00000001
gPlatform2TokenSpaceGuid.Pcd3|0x12340000|UINT32|0x00000000
gPlatform2TokenSpaceGuid.Pcd4|0x00001234|UINT32|0x00000001
```

- `gPlatform1TokenSpaceGuid.Pcd1 and gPlatform1TokenSpaceGuid.Pcd2` belong to the same GUID token space, or “the same namespace”
- In addition to the name, a token number (unique to the token space) is also declared, and allows code to get or set a PCD value using the token guid and token number

DEC Format (cont.)

- When getting 32-bit PCD value, `PcdGet32()` is passed the *PcdName*. If 2 PCDs have same name but code calls `PcdGet32()` for only 1 of them, build will be successful. If code calls `PcdGet32()` for both of them, build will fail. This is because `AutoGen.h` and `AutoGen.c` (created by `build.exe` and referenced by compiler) contains #defines of *Guid_PcdName* for every PCD that is used in code; thus there will be a macro redefinition error because same macro will be created twice.
  - Example of successful build: `MyGuid1.foo and MyGuid2.foo are defined, only MyGuid1.foo is declared in INF`
  - Example of failed build: `MyGuid1.foo and MyGuid2.foo are defined, MyGuid1.foo and MyGuid2.foo are declared in INF`
- PCD types are described in *DSC spec Chapter 2.2.8*

PCD Format

- PCDs are declared in *DEC* files, and can be set anywhere else
  - PcdName grammar is `<TokenSpaceGuidCName> “.” <PcdName>`, where both items are defined as a valid *C variable* name
  - Other PCD parameters are *InitialValue* | *DataType* | *TokenNum*
  - `TokenSpaceGuidCName` must be a GUID declared under `[Guids]` section
  - `TokenNum` must be unique within a GUID because some *PCD* Protocol functions to get PCD data require a GUID and *TokenNum* parameters; the internal PCD database indexes the PCDs by the GUID and *TokenNum* to set and get values.
- See *DEC spec Chapter 3.9* for PCD grammar
- See *PI spec 1.2 Volume 3 Chapter 8*
- See *EDK II INF spec Chapter 3.8*

PCD Format (cont.)

- Steps required to add PCD access to a *UEFI* application:

1. Declare GUID and PCD in application *DEC* file (`MyTest.dec`)

```
[Guids]
  gMyGuid = { 0x07dfa0d2, 0x2ac5, 0x4cab, { 0xac, 0x14, 0x30, 0x5c, 0x62, 0x48,
0x87, 0xe4 }}
[PcdsFixedAtBuild]
  gMyGuid.foo|0xBEEF0000|UINT32|0x00000000
```

2. Declare PCD library, GUID, and PCD in the application *INF* file (`MyTest.inf`)

```
[Packages]
  MdePkg/MdePkg.dec
  MyTest/MyTest.dec
[LibraryClasses]
  PcdLib
[Guids]
  gMyGuid
[FixedPcd]
  gMyGuid.foo
```

3. Declare PCD library in the application *DSC* file (`MyTest.dsc`)

```
[LibraryClasses]
  PcdLib|MdePkg/Library/BasePcdLibNull/BasePcdLibNull.inf
[Components]
  MyTestPkg/MyTest/MyTest.inf
[PcdsFixedAtBuild] # if you want to override DEC value
  gMyGuid.foo|0xBEEF1234
```

PCD Format (cont.)

4. Declare GUID extern in application *H* file (`MyTest.h`)

```
extern EFI_GUID gMyGuid;
```

5. Include PCD library and application H file in application *C* file (`MyTest.c`)

```
#include <Library/PcdLib.h>
#include “MyTest.h”
Print(L"gMyGuid.foo = 0x%08X\n", PcdGet32(foo));
```

INF Format

- *MODULE_TYPE* values for `[Defines]` section described in *INF spec Appendix G*
- Source files must be listed in `[Sources]` section.
  - `.C` (code), strings (`.UNI`), etc.
- Packages used must be listed in `[Packages]` section.
  - `.DEC` files
- Libraries used must be listed in `[LibraryClasses]` section.
  - Names should match the name specified in each library's *LIBRARY_CLASS*, but ultimately it is up to the platform DSC to decide the library instance to use when specifying a library INF file.
- *Guids* used must be listed in `[Guids]` section.
- *PCDs* used must be listed in a typed `PCD` section:
  - `FixedPcd, FeaturePcd, PatchPcd, Pcd, PcdEx`
  - See *INF spec Chapter 3.8*
- A module developer is responsible for creating the *INF*, and knows how their module source code accesses the PCDs via PcdLib (`MdePkg/Include/Library/PcdLib.h`).

INF Format (cont.)

- How do you know which PCD type to use?
  - If the C sources for a module access a PCD using `FeaturePcdGet (TokenName)`, the PCD must be listed in `[FeaturePcd]`
  - If the C sources for a module access a PCD using `FixedPcdGetXX (TokenName)`, the PCD must be listed in `[FixedPcd]`
  - If the C sources for a module access a PCD using `PatchPcdGetXX (TokenName)`, the PCD must be listed in `[PatchPcd]`
  - If the C sources for a module access a PCD using `PatchPcdSetXX (TokenName)`, the PCD must be listed in `[PatchPcd]`
  - If the C sources for a module access a PCD using `PcdGetXX (TokenName)`, the PCD must be listed in `[Pcd]`
  - If the C sources for a module access a PCD using `PcdSetXX (TokenName)`, the PCD must be listed in `[Pcd]`
  - If the C sources for a module access a PCD using `PcdGetExXX (TokenName)`, the PCD must be listed in `[PcdEx]`
  - If the C sources for a module access a PCD using `PcdSetExXX (TokenName)`, the PCD must be listed in `[PcdEx]`

Legal Notice

THIS INFORMATION COTNAINED IN THIS DOCUMENT, INCLUDING ANY TEST RESULTS ARE PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT OR BY THE SALE OF INTEL PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel products are not intended for use in medical, life saving, or life sustaining applications.

- Intel retains the right to make changes to its specifications at any time, without notice.
- Recipients of this information remain solely responsible for the design, sale and functionality of their products, including any liability arising from product infringement or product warranty.
- Intel may make changes to specifications, product roadmaps and product descriptions at any time, without notice.
- Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- *Other names and brands may be claimed as the property of others.