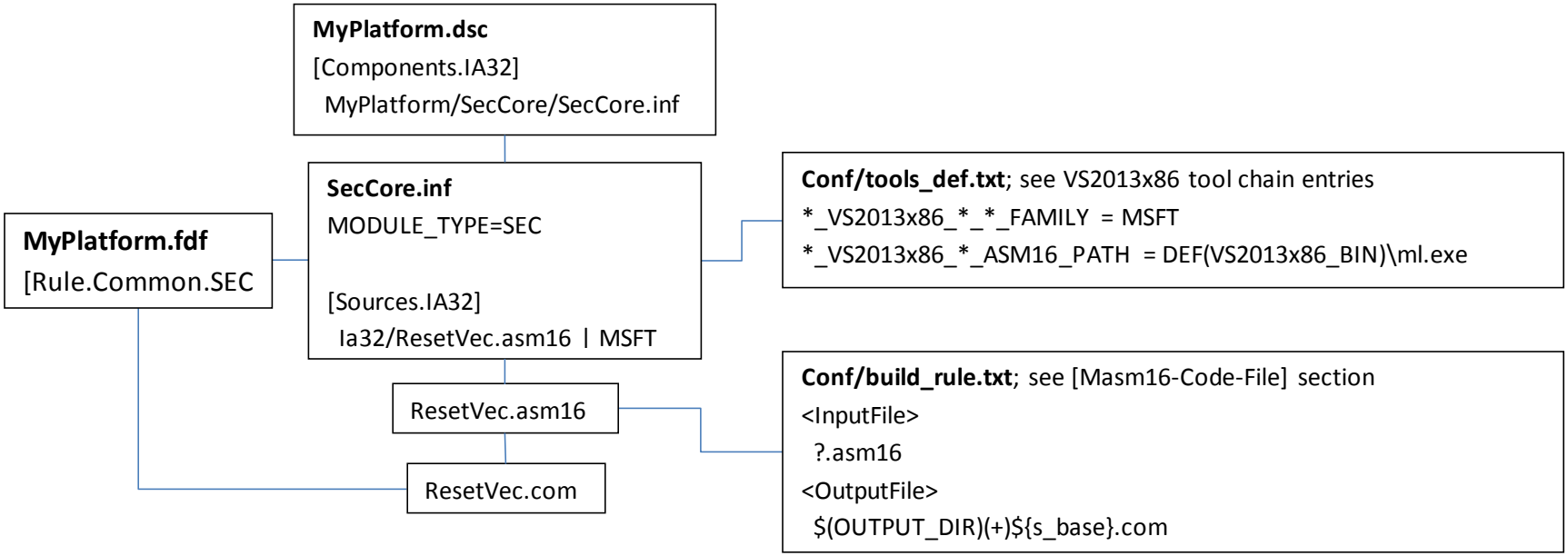
UDK 2014 Build Integration of Reset Vector

There are many stages of the build process that are involved with the creation of the Reset Vector. This document is an example of how one platform uses it with UDK 2014, and is not representative of how all platforms may use it. An example project called *MyPlatformPkg*, Visual Studio 2013 x86 (VS2013x86) tool chain, and Reset Vector code in *MyPlatformPkg/SecCore/SecCore.inf* is used throughout this document. Your platform may be using the Reset Vector code in *UefiCpuPkg/ResetVector/Vtf0/Vtf0.inf*. The concepts discussed in this paper apply to any Reset Vector code.

Build Diagram



Creating the PE32 image file

Start with your Reset Vector code in *MyPlatformPkg/SecCore/SecCore.inf*, which typically looks like this:

```
[Sources.IA32]
  ResetVec.asm16 | INTEL
  ResetVec.asm16 | MSFT
```

MyPlatformPkg/SecCore/SecCore.inf is part of the build because it is included in *MyPlatformPkg/MyPlatformPkg.dsc* by:

```
[Components.IA32]
  MyPlatform/SecCore/SecCore.inf
```

UDK 2014 Build Integration of Reset Vector

When build.exe processes *SecCore.inf* it creates
Build/MyPlatformPkg/DEBUG_VS2013x86/IA32/SecCore/SecCore/makefile
Search for *ResetVec.asm16* in this file. You will find a series of lines that start with

```
$(OUTPUT_DIR)\ResetVec.com :
```

How did the build process know to make *ResetVec.com* from *ResetVec.asm16*? The file *Conf/build_rule.txt* controls how various files are compiled, and it has a section for .asm16 files. Search for .asm16 in the *Conf/build_rule.txt* file and you will find

```
<OutputFile>
  $(OUTPUT_DIR)(+)${s_base}.com
```

The build process substitutes (the value of BASE_NAME in the INF file) *ResetVec* for *s_base* to get *ResetVec.com*.

In the [Sources.IA32] section there is "| INTEL" and "| MSFT". What do these tags control? Refer to *EDK II INF Spec* *Table 3 EDK II [BuildOptions] Variable Descriptions* and examine the FAMILY variable row. The FAMILY variable is cross referenced to the compiler tool chain family specified in *Conf/tools_def.txt* (each Microsoft Visual Studio-based or Intel C++ Compiler for Windows-based tool chain has an item *_*_*_*_FAMILY = INTEL or MSFT) so the path of the assembler can be created in the $(ASM) variable in SecCore/makefile with data from *_*_*_ASM_PATH.

In *MyPlatformPkg/SecCore/SecCore.inf* there are the following statements:

```
MODULE_TYPE = SEC
BASE_NAME = SecCore
```

This tells the build to create the driver SecCore.efi. This file is created in the $(INF_OUTPUT) directory which is *Build/MyPlatformPkg/DEBUG_VS2013x86/IA32/SecCore/SecCore/OUTPUT*. This directory is where you will find ResetVec.com.

Creating the FFS file

How does the build integrate *SecCore.efi* and *ResetVec.com* into your FD file?
If you run **build –v** to get verbose output, you will see that after all files are compiled, *build.exe* calls *GenFds* with an argument of *MyPlatformPkg/MyPlatformPkg.fdf*. At the bottom of the *MyPlatformPkg.fdf* file are [Rules] sections. These rules tell the build how to integrate driver files, which are created from the rules. Since each INF file specified in [FV] sections specify the module type, the output directories for each module are searched for matching patterns specified in a [Rule.<arch>.<ModuleType>] section, and an

FFS file is created containing the UEFI Sections specified by the rule. These FFS files are placed into the Firmware Volumes (FV), which are placed into the Flash Devices (FD).

```
[Rule.Common.SEC]
  FILE SEC = $(NAMED_GUID) RELOCS_STRIPPED {
    TE  TE    Align = 8       $(INF_OUTPUT)/$(MODULE_NAME).efi
    RAW BIN   Align = 16      |.com
  }
```

NOTE: *TE = Terse Executable; see PI Spec Vol 1 Chapter 15.1.*

The build knows to associate this INF with the rules specified in [Rule.Common.SEC] (SEC is the EDK II ModuleType; see *EDK II FDF Spec Chapter 3.9*) because there is a match between the INF MODULE_TYPE and the rule's ModuleType.

Volume Top File (VTF)

SecCore.inf also has this FILE_GUID text:

```
  FILE_GUID = 1BA0062E-C779-4582-8566-336AE8F78F09
```

This Registry Format GUID is known as the VTF GUID, also called EFI_FFS_VOLUME_TOP_FILE_GUID (see *EDK II Build Spec Chapter 2.5.4*). The build system will replace the $(NAMED_GUID) with FILE_GUID value. The first line of the rule after replacement is:

```
  FILE SEC = 1BA0062E-C779-4582-8566-336AE8F78F09 RELOCS_STRIPPED {
```

The build will use the value of the BASE_NAME entry SecCore to replace $(MODULE_NAME). ResetVec.com is associated with the *.com line (SEC rule, |.com file)* because it's the only .com file in the folder.

After replacement, the entire rule for processing *MyPlatformPkg/SecCore/SecCore.inf* is expanded by the build system to be:

```
  FILE SEC = 1BA0062E-C779-4582-8566-336AE8F78F09 RELOCS_STRIPPED {
    TE  TE    Align = 8       Build/MyPlatformPkg/DEBUG_VS2013x86/IA32/SecCore/SecCore/OUTPUT/SecCore.efi
    RAW BIN   Align = 16      Build/MyPlatformPkg/DEBUG_VS2013x86/IA32/SecCore/SecCore/OUTPUT/ResetVec.com
  }
```

GenFds modifies the PE32 header to make it a TE header, then GenFds puts the SecCore.efi file in a TE section at the bottom of the FFS file and the ResetVec.com file at the top of the FFS file. The generated FFS file is named by the registry format FILE_GUID, and is located in a subdirectory under the FV directory in the build output directory:
`Build/MyPlatformPkg/DEBUG_VS2013x86/FV/Ffs/1BA0062E-C779-4582-8566-336AE8F78F09SecCore/1BA0062E-C779-4582-8566-336AE8F78F09.ffs`

The FFS file for SEC uses EFI_FFS_VOLUME_TOP_FILE_GUID and is defined in
BaseTools/Source/C/Include/Guid/PiFirmwareFileSystem.h.
The GUID is the same value as in *MyPlatformPkg/SecCore/SecCore.inf* but is in DEFINE_GUID format.

NOTE: *Searching your source tree for the VTF GUID in either format won't yield all matches because the Registry and DEFINE_GUID formats are both used due to the programming languages and build constructs used.*

The C Name for the VTF GUID is defined as gEfiFirmwareVolumeTopFileGuid in the MdePkg/MdePkg.dec file using the C Format GUID structure. Search for the C name in the code files rather than the GUID value.
GenFds processes the VTF GUID (see *BaseTools/Source/Python/Eot/FvImage.py*) and relocates the image to the top of the FV.

Placing ResetVec FFS file in the FV and FD

The FDF file specifies the FV layout, so the FV that contains the SEC must be the FV in the top region of the FD that includes the Reset Vector. Your FDF file should have something like

```
[FD.SecPei]
  BaseAddress = $(FLASH_REGION_FV_SECPEI_NORMAL_BASE) # 4G - 256K
  Size        = $(FLASH_REGION_FV_SECPEI_NORMAL_SIZE) # 256K
```

This tells the build to create an FD file by taking the tag to the right of "FD.", called the FdUiName (see *EDK II FDF Spec Chapter 2.3*), and appends ".FD" to form the filename SECPEI.FD. This file is created in *Build/MyPlatformPkg/DEBUG_VS2013x86/FV* directory by the build tool *BaseTools/Source/Python/GenFds/FdfParser.py*. GenFds knows what FV file to put in [FD.SecPei] because there should be a line FV = FVSECPEI.

GenFds searches for a section in the FDF called [FV.FVSECPEI] to know how to create the FV that will go in this FD. [FV.FVSECPEI] will have a series of INF xxx statements; one of them should be

```
INF MyPlatformPkg/SecCore/SecCore.inf
```

You should now understand how the build processes the DSC to build your SEC INF, which is specified by MODULE_TYPE = SEC and FILE_GUID = 1BA0062E-C779-4582-8566-336AE8F78F09 (the VTF GUID), and populate the top of memory with the Reset Vector as specified by the FDF statements of [Rule.Common.SEC], [FD.SECPEI], and [FV.FVSECPEI].

Reference Documents

UDK 2014
EDK II INF Spec
EDK II FDF Spec
EDK II Build Spec
PI Spec Vol 1