Debug Framework Enhancements User Guide

Final Version
August 7, 2014

Anish Parikh
System Software Engineer, Firmware
anish_parikh@apple.com

Contents

1 Introduction

1.1 What is this about?

The new debug framework for EFI makes on-site or remote debugging a one or two command effort. This is an extensive framework which covers a wide range of platforms.

1.2 Audience

This document is intended for the engineers involved with the development and use of the EFI diagnostic environment.

1.3 Document Objective

This document will provide relevant technical details and concepts introduced in the new framework. It will also provide command usage guidelines that are useful to most users working in EFI.

2 Setting up LLDB

2.1 Clone Repository

Clone the diagssupport repository where all the scripts are located.

git clone git@gitlab.sd.apple.com:blackops/diagssupport.git

2.2 Create ".lldbinit" file

Its a good idea to create a "~/.lldbinit" file so that you dont have to load the scripts each time you start LLDB. You need to have the following information inside the "~/.lldbinit" file to use the framework effectively:

command script import <path-to-file>/efi.py

Note: Here the <path-to-file> means the path where the files are located on your local machine

This file internally loads all the other scripts which you require. This means that now the user does not need to worry about which other files to include! The only caveat here is that the user needs to make sure that all the required script files are in the same directory as the "efi.py" file.

3 Using Framework

3.1 Available Commands

The commands available in the framework are:

console
symbols
loadpei
hob
build_info
config_table

3.2 Console

The console command gives you access to the forensics buffer within the device

Usage: *console -l*

Sample Output:

```
(lldb) console -l
BufferSize: 0x40000
Freespace: 0x3253b
Head: 0x0
Tail: 0xdac5
Data:0x87e2fe018
[Buf_PrintInfo]<0.0s (+0.000000)>Console router buffer allocated @ 0x87E2FE018, size = 262144
 [<no-func>]<14.656600s (+0.000006)>SetPS : Id 0x20E020168, State : 0x2
 [<no-func>]<14.656605s (+0.000005)>SetPS : Id 0x20E020170, State : 0x2
 [<no-func>]<14.656610s (+0.000005)>SetPS : Id 0x20E020178, State : 0x2
 [<no-func>]<14.656782s (+0.000172)>Install Display BackEnd done
 [<no-func>]<14.656934s (+0.000152)>Searching 5 DP Handles...
 [<no-func>]<14.656937s (+0.000003)>  T0x01 S0x03
 [<no-func>]<14.656941s (+0.000004)>  T0x01 S0x04
 [<no-func>]<14.656944s (+0.000003)>    G: 39AF9652-7356-4DA2-BB-C0-18-E8-05-00-99-4B
 [<no-func>]<14.656950s (+0.000006)>      DT:0x19
 [<no-func>]<14.656953s (+0.000003)>  T0x01 S0x04
 [<no-func>]<14.656955s (+0.000002)>    G: 9685762D-1A08-4C1E-A7-86-B9-62-9E-86-DF-69
 [<no-func>]<14.656960s (+0.000005)>  T0x01 S0x04
 [<no-func>]<14.656963s (+0.000003)>    G: 39AF9652-7356-4DA2-BB-C0-18-E8-05-00-99-4B
 [<no-func>]<14.656967s (+0.000004)>      DT:0x00
 [<no-func>]<14.657087s (+0.000120)>Searching 10 DP Handles...
 [<no-func>]<14.657091s (+0.000004)>  T0x01 S0x03
 [<no-func>]<14.657094s (+0.000003)>  T0x01 S0x04
 [<no-func>]<14.657097s (+0.000003)>    G: 39AF9652-7356-4DA2-BB-C0-18-E8-05-00-99-4B
 [<no-func>]<14.657101s (+0.000004)>      DT:0x19
 [<no-func>]<14.657104s (+0.000003)>  T0x01 S0x04
```

3.3 Symbols

The symbols command is used for symbolication. It automatically detects whether you are running a local or stock build and symbolicates appropriately.

If you are running a stock build then it retrieves the symbols for that build from the build server.

Usage: *symbols*

Sample Output:

```
(lldb) symbols
Retrieving symbols from HOB
Loading symbol files to LLDB
File Path Found:/build/archive/shasta_j82_702/***/bootloader/Dwi.macho @ 0x87e3b5000
File Path Found:/build/archive/shasta_j82_702/***/bootloader/Tristar.macho @ 0x87e2c6000
File Path Found:/build/archive/shasta_j82_702/***/bootloader/PMGR.macho @ 0x87e261000
File Path Found:/build/archive/shasta_j82_702/***/bootloader/Gpio.macho @ 0x87e2de000
File Path Found:/build/archive/shasta_j82_702/***/bootloader/DiagBds.macho @ 0x87e2f5000
File Path Found:/build/archive/shasta_j82_702/***/bootloader/ChipId.macho @ 0x87e2ad000
File Path Found:/build/archive/shasta_j82_702/***/bootloader/Timer.macho @ 0x87e2a8000
File Path Found:/build/archive/shasta_j82_702/***/bootloader/Variable.macho @ 0x87e2a4000
File Path Found:/build/archive/shasta_j82_702/***/bootloader/Smbus.macho @ 0x87e243000
File Path Found:/build/archive/shasta_j82_702/***/bootloader/MCA.macho @ 0x87b4f3000
```

To verify if symbols have been loaded you can use the command **image list**

Also to verify that you have the correct source checked out you can use the **build_info** command to compare the tag of the build against your source.

3.3.1. Fallback mechanism

If, for some reason when using the symbols command, you get a message saying that the HOB hasn't been loaded or that you are not connected to the HOB, then there is a very good chance that the crash occured in Pre-EFI. In that case you can use the **loadpei** command which is explained in Section 3.4 as a fallback.

3.3.2. Mapping remote source files

Imagine a situation when you are downloading symbols from the build server, which may be the case when you are running a stock build on a device. If you want to get detailed information when executing a backtrace command, you can use the following command to map the source files on your machine to those which were used to build the stock build on the device:

settings set target.source-map <path-to-remote-source-files> <path-to-source-files-on-local-machine>

Example: settings set target.source-map /Users/build/archive/shasta-intern_j82_9/src/shasta-intern/ /Users/Anish/Documents/shasta-intern

3.3.3. Optimization

If you have already loaded the symbols once on your local machine then the next time you want to load them for the same platform you can use the following command:

Usage: *symbols -p <target-platform>*

This command loads the symbols much faster since it doesn't try to locate them again. Instead it justs loads a pre-defined file which has the symbols from the previous iteration into LLDB!

Sample Output:

```
(lldb) symbols -p J82
Executing commands in '/private/tmp/symbols-J82.lldb'.
(lldb) target modules add /Users/Anish/Documents/shasta-intern/***/AppleConsoleRouter.macho
(lldb) target modules load --file AppleConsoleRouter.macho --slide 0x87dc61000
(lldb) target modules add /Users/Anish/Documents/shasta-intern/***/DxeStatusCode.macho
(lldb) target modules load --file DxeStatusCode.macho --slide 0x87dc17000
(lldb) target modules add /Users/Anish/Documents/shasta-intern/***/ScratchRegister.macho
(lldb) target modules load --file ScratchRegister.macho --slide 0x87da68000
```

3.4 Load Pre-EFI Symbols

To load the Pre-EFI symbols use the loadpei command. This is particularly helpful to debug crashes in Pre-EFI state. If the symbols command is successfully executed prior to running this command then the Pre-EFI symbols would have already been loaded as a part of its execution.

Usage:
loadpei
loadpei -t <target_platform>

Note: Its easy to determine whether the HOB has been initialized by just running **loadpei** initially and if it says that HOB hasn't been initialized then specify the target platform along with the command.

Sample Output:

```
(lldb) loadpei
Executing commands in '/Users/Anish/Documents/***/load_pei_symbols.lldb'.
(lldb) target modules add /Users/Anish/Documents/***/debug/bootloader/VectorJumpIsland.obj
(lldb) target modules load --file VectorJumpIsland.obj --slide 0x804000000
(lldb) target modules add /Users/Anish/Documents/***/debug/bootloader/SecCore.macho
(lldb) target modules load --file SecCore.macho --slide 0x80436c000
(lldb) target modules add /Users/Anish/Documents/***/debug/bootloader/PreEfi.macho
(lldb) target modules load --file PreEfi.macho --slide 0x8042f8000
```

3.5 Hob

This command prints all the entries in the HOB List.

Usage: *hob -l*

To print just a specific entry in the HOB List

Usage: *hob -t <Name of Entry>*

Sample Output:

```
(lldb) hob -l
Hob Type: 0x1, HobLength: 56
Hob Type: 0x5, HobLength: 24
Hob Type: 0x2, HobLength: 48
Hob Type: 0x2, HobLength: 48
Hob Type: 0x2, HobLength: 48
Hob Type: 0x4, HobLength: 32
Hob Type: 0x6, HobLength: 16
Hob Type: 0x2, HobLength: 48
Hob Type: 0x4, HobLength: 56
Hob Type: 0x4, HobLength: 88
Hob Type: 0x4, HobLength: 88
Hob Type: 0x2, HobLength: 48
Hob Type: 0x4, HobLength: 56
Hob Type: 0x4, HobLength: 32
Hob Type: 0x4, HobLength: 32
Hob Type: 0x4, HobLength: 32
Hob Type: 0x4, HobLength: 32
Hob Type: 0x4, HobLength: 32
Hob Type: 0x4, HobLength: 32
Hob Type: 0x3, HobLength: 48
Hob Type: 0x2, HobLength: 48
Hob Type: 0x2, HobLength: 48
Hob Type: 0x2, HobLength: 48
Hob Type: 0x2, HobLength: 48
Hob Type: 0xb, HobLength: 80
Hob Type: 0x9, HobLength: 56
Hob Type: 0xc, HobLength: 24
Hob Type: 0xa, HobLength: 16
Hob Type: 0xffff, HobLength: 8
EFI_HOB_HANDOFF_INFO_TABLE(0x87fb9f598):
    Header:EFI_HOB_GENERIC_HEADER(0x87fb9f598):
        HobType(UINT16):
            0x87fb9f598: 0x0001
        HobLength(UINT16):
            0x87fb9f59a: 0x0038
        Reserved(UINT32):
            0x87fb9f59c: 0x00000000
    Version(UINT32):
        0x87fb9f5a0: 0x00000009
    BootMode(UINT32):
        0x87fb9f5a4: 0x00000000
    EfiMemoryTop(EFI_PHYSICAL_ADDRESS):
        0x87fb9f5a8: 0x0000000880000000
    EfiMemoryBottom(EFI_PHYSICAL_ADDRESS):
        0x87fb9f5b0: 0x0000000800000000
    EfiFreeMemoryTop(EFI_PHYSICAL_ADDRESS):
        0x87fb9f5b8: 0x000000087fba5000
    EfiFreeMemoryBottom(EFI_PHYSICAL_ADDRESS):
        0x87fb9f5c0: 0x00000008000004f8
    EfiEndOfHobList(EFI_PHYSICAL_ADDRESS):
        0x87fb9f5c8: 0x00000008000004f0
EFI_HOB_FIRMWARE_VOLUME(0x87fb9f5d0):
    Header:EFI_HOB_GENERIC_HEADER(0x87fb9f5d0):
        HobType(UINT16):
```

```
            0x87fb9f5d0: 0x0005
        HobLength(UINT16):
            0x87fb9f5d2: 0x0018
        Reserved(UINT32):
            0x87fb9f5d4: 0x00000000
    BaseAddress(EFI_PHYSICAL_ADDRESS):
        0x87fb9f5d8: 0x0000000804001000
    Length(UINT64):
        0x87fb9f5e0: 0x000000000030f000


(lldb) hob -t EFI_HOB_SYSTEM_TABLE
EFI_HOB_SYSTEM_TABLE(0x87fb75a78):
    Header:EFI_HOB_GENERIC_HEADER(0x87fb75a78):
        HobType(UINT16):
            0x87fb75a78: 0x000a
        HobLength(UINT16):
            0x87fb75a7a: 0x0010
        Reserved(UINT32):
            0x87fb75a7c: 0x00000000
    TableLocation(EFI_PHYSICAL_ADDRESS):
        0x87fb75a80: 0x000000087fb79f18
```

3.6 Build Information

This commmand prints out all the information about the build which is running on the device currently.

Usage: *build_info*

Sample Output:

```
(lldb) build_info
PlatformID: J82
BuildType: BuildEng
BuildTrain: Shasta
DiagsBuildBranch: master
DiagsBuildID: 11D2130
BuildNumber: 702
```

3.7 Configuration Table

This command prints out the configuration table. It shows you the number of tables and their respective GUID information.

Usage: *config_table*

Sample Output:

```
(lldb) config_table
EFI_SYSTEM_TABLE(0x87fba3f18):
    Hdr:EFI_TABLE_HEADER(0x87fba3f18):
```

```
            Signature(UINT64):
                0x87fba3f18: 0x5453595320494249
            Revision(UINT32):
                0x87fba3f20: 0x0001000a
            HeaderSize(UINT32):
                0x87fba3f24: 0x00000078
            CRC32(UINT32):
                0x87fba3f28: 0xa9ea8f10
            Reserved(UINT32):
                0x87fba3f2c: 0x00000000
        FirmwareVendor(CHAR16*):
            0x87fba3f30: 0x0000000000000000
        FirmwareRevision(UINT32):
            0x87fba3f38: 0x00000000
        ConsoleInHandle(EFI_HANDLE):
            0x87fba3f40: 0x0000000000000000
        ConIn(EFI_SIMPLE_TEXT_INPUT_PROTOCOL*):
            0x87fba3f48: 0x000000087e112000
        ConsoleOutHandle(EFI_HANDLE):
            0x87fba3f50: 0x0000000000000000
        ConOut(EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL*):
            0x87fba3f58: 0x000000087e112030
        StandardErrorHandle(EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL):
            0x87fba3f60: 0x0000000000000000
        StdErr(EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL*):
            0x87fba3f68: 0x0000000000000000
        RuntimeServices(EFI_RUNTIME_SERVICES*):
            0x87fba3f70: 0x000000087fba2d98
        BootServices(EFI_BOOT_SERVICES*):
            0x87fba3f78: 0x000000087fbb2000
        NumberOfTableEntries(UINTN):
            0x87fba3f80: 0x0000000000000004
        ConfigurationTable(EFI_CONFIGURATION_TABLE*):
            0x87fba3f88: 0x000000087fb9ee18
    NumberOfTableEntries = 4
    ConfigurationTable = 87fb9ee18
    05AD34BA-6F02-4214-952E-4DA0398E2BB9  VendorTable = 0x87fbb2170
    7739F24C-93D7-11D4-9A3A-0090273FC14D  VendorTable = 0x87fb9f598
    4C19049F-4137-4DD3-9C10-8B97A83FFDFA  VendorTable = 0x87fbb2938
    49152E77-1ADA-4764-B7A2-7AFEFED95E8B  VendorTable = 0x87fbb58d8
```

4 Next Steps

The ultimate goal of this project is to integrate this framework with the panic.apple.com infrastructure. This will allow us to provide real-time coverage for build issues occurring in the factory. This process is already in motion and should be completed soon!

To enable this there is one more command **triage** which will run a bunch of these scripts and collect the output and file a Radar with the information. This way we can get a better idea of the issue which we are facing by studying the relevant Radar.