

presented by



Firmware configuration – Past, Present, and Future

UEFI Fall 2023 Developers Conference & Plugfest

October 9-12, 2023

Presented by

Vincent Zimmer (Intel Corporation)

Gahan Saraiya (Intel Corporation)

Christine Chen (Intel Corporation)

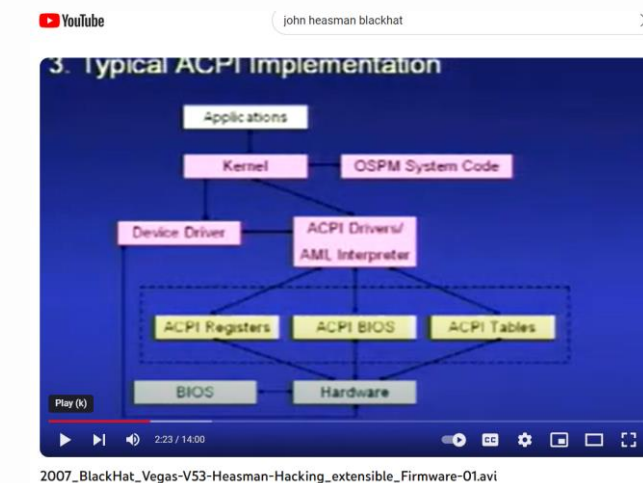
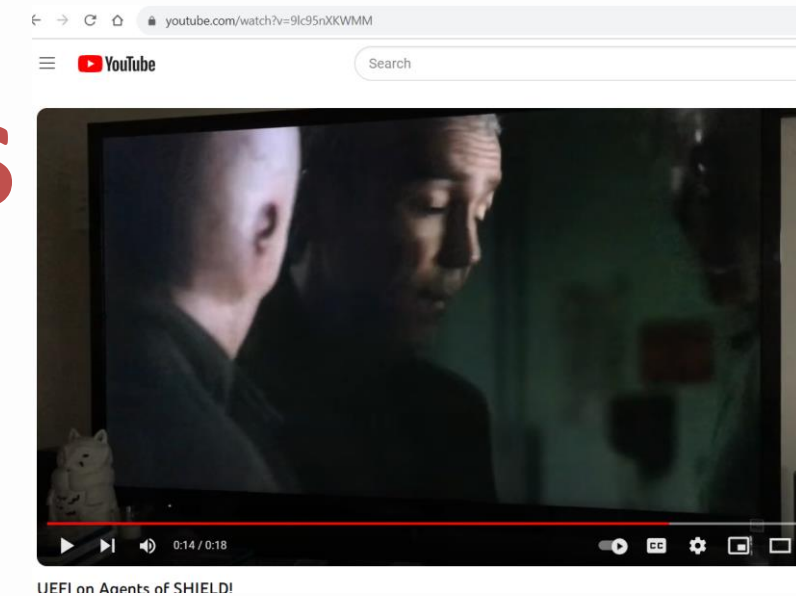
Catch up from last 2 days

- U-E-F-I versus YOU-FEE?
- Agent of SHIELD provides not help

<https://www.youtube.com/watch?v=9lc95nXKWMM> (0:8)

- And if YOU-FEE, then ACK-PEE?

– <https://www.youtube.com/watch?v=A0seUVAC09E> (1:40)





'Speaker' bio's

Vincent Zimmer



Vincent Zimmer is a Senior Principal Engineer with Intel. He joined Intel in 1997 and has been working on EFI / Framework / UEFI / PI along with touching TCG and other stds groups since 1999. He lives in the Seattle area and can be reached via various paths –

*vincent.zimmer@intel.com or 425-881-4874
or <https://twitter.com/vincentzimmer?lang=en>
or <https://www.linkedin.com/in/vzimmer> or*



Gahan Saraiya

Gahan is a skilled Platform Lead at Intel, working within the Software and Advanced Technology Group. With over four years of experience, he has been dedicated to the development and implementation of firmware configurations that support the expansion of automation infrastructure. Gahan's contributions have consistently made a positive and lasting impact on the organization during his tenure.



[Know more about Gahan](#)



Christine Chen

Christine is the software development engineer at Intel, has been working on the Edk2 BaseTools related development and support works since 2020. She has designed and developed BaseTools FMMT python tool, and has been participated in BaseTools incremental Build project, BaseTools C tool python conversion work. She is focus on BIOS configuration enhancement in BaseTools build system currently.



Agenda



- Introduction
- Evolution
- Present Practice
- Next Generation of Firmware Configuration
- Questions



Introduction

Introduction



- Growing requirements of various application demands different firmware flows
- Drilled down to classify with conditional flow



Evolution

Evolution

- Boot Flow Control with Firmware Configuration
- HII Implementation



Boot Flow Control with Firmware Configuration

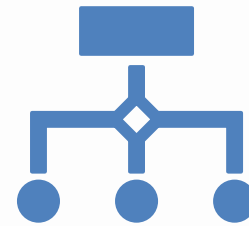


- EFI Variables act as control variable for firmware boot flow process
- Fulfills need of simpler to complex boot flow context, i.e.
 - boot priority ordering
 - security configuration
 - overclocking of device

HII Implementation



HII establishes Infrastructure between external entity to firmware configuration



Control variable (EFI Variable) mapped through various component of HII

vfr/hfr – customized syntactical language to construct user interface

uni – string representation of Identifier mapped in vfr/hfr

Header file *.h – variable structure and GUID

Source file *.c – HII Protocols



Present Practice

Present Practice

- Non-Recommended Practices
- Runtime population of form representation
- Tools and Tech



Present Practice – Non-Recommended Practices



- Action callbacks associated with form representation

Present Practice - Tools and Tech



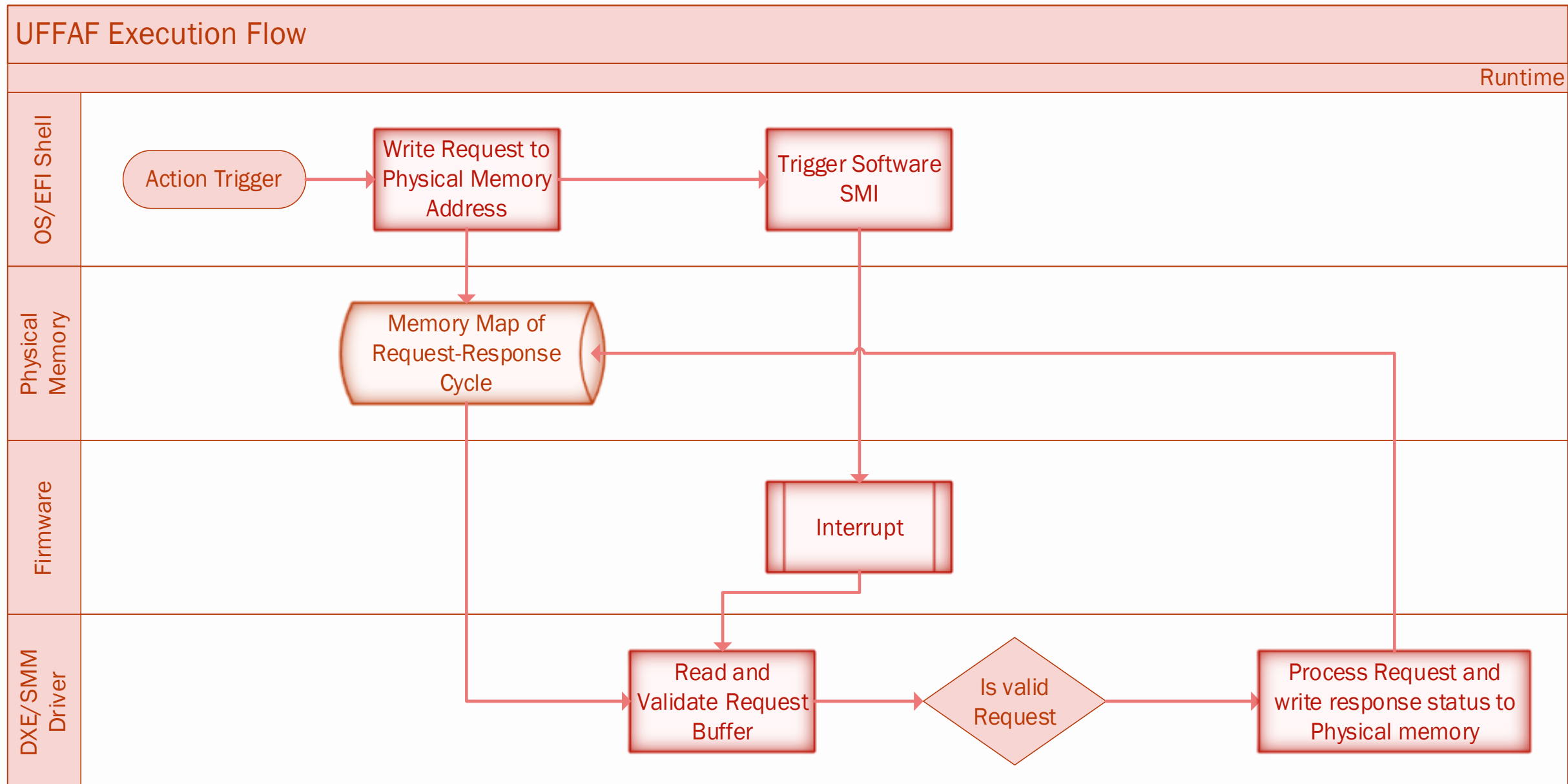
TOOL/TECH	REFERENCE
RedFish	<u>DMTF's Redfish[®]</u>
UFFAF	<u>UEFI Firmware Foundational Automation Framework</u>
USF YAML	<u>yaml boot configuration</u>
FDT	<u>Flattened device tree</u>
FSP UPD	<u>FSP UPD</u>
DFCI	<u>Mu DFCI</u>

Present Practice - Tools and Tech

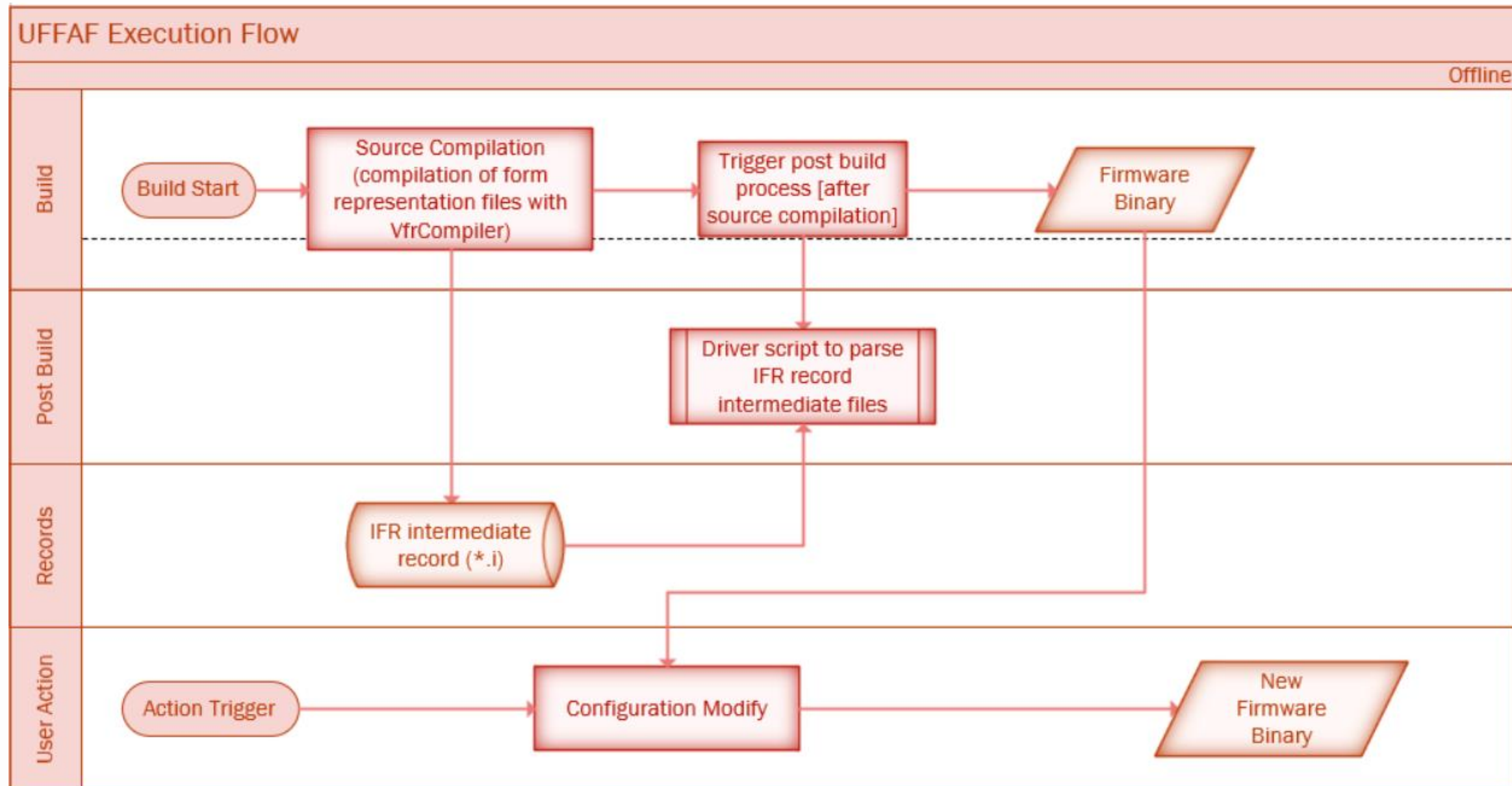


Tool	User Visible Data Format	Firmware Implementation	End-User Programming Language
RedFish	json	Edk2 Advanced Feature Package	Python
UFFAF	Xml,json	Intel Advanced Feature Package	Python
USF YAML	yaml	BaseTools	Python
FDT	dtb	Device tree compiler (DTC)	C
FSP UPD	Bsf, yaml	Configuration editor	Python

UFFAF UEFI Firmware Foundational Automation Framework



UFFAF UEFI Firmware Foundational Automation Framework



YAML and USF



← ↻ 🏠 🔒 https://universalscalablefirmware.github.io/documentation/7_yaml_boot_configuration.html 🔍 📄 ☆ 🟢 ⚙️ | 📄 ☆ 🗑️ 🔒

🏠 Universal Scalable Firmware (USF)

Search docs

Revision History

Notices and Disclaimers

1. Universal Scalable Firmware (USF) Specification

2. Universal Payload

3. Platform Orchestration Layer (POL)

4. Runtime

5. Security

6. Debug

7. YAML Format Boot Configuration

7.1. Introduction

7.2. Target Audience

7.3. Configuration Description (YAML) Explained

7.4. File Layout

7.5. Variable

7.6. Template

7.7. Configs

7.8. Delta (DLT) File Explained

7.9. DLT file rules

7.10. Configuration Process

8. Scalable FSP

9. Bootloader Payloads

🏠 » 7. YAML Format Boot Configuration

7. YAML Format Boot Configuration

7.1. Introduction

This document describes the format of the YAML based boot setting file used to specify features, settings, and tool display information to the Intel Firmware Support Package (FSP) and tools like Config Editor. It further describes the structure and content of YAML format files, which can simplify the configuration of a static binary and during dynamic boot process.

7.2. Target Audience

This document is intended for person creating and using firmware images. It is most likely of interest if a developer needs to create customized feature and expose binary settings for delivery to customers, or that a new SOC device is being enabled.

7.3. Configuration Description (YAML) Explained

The declarations required to build the configuration data blobs and the header files are provided in a configuration description file. This file uses the **YAML** syntax.

YAML (<https://yaml.org/>) is a data serialization language designed to be human-friendly and work well with modern programming languages. A quick syntax reference can be found here - <https://yaml.org/refcard.html>

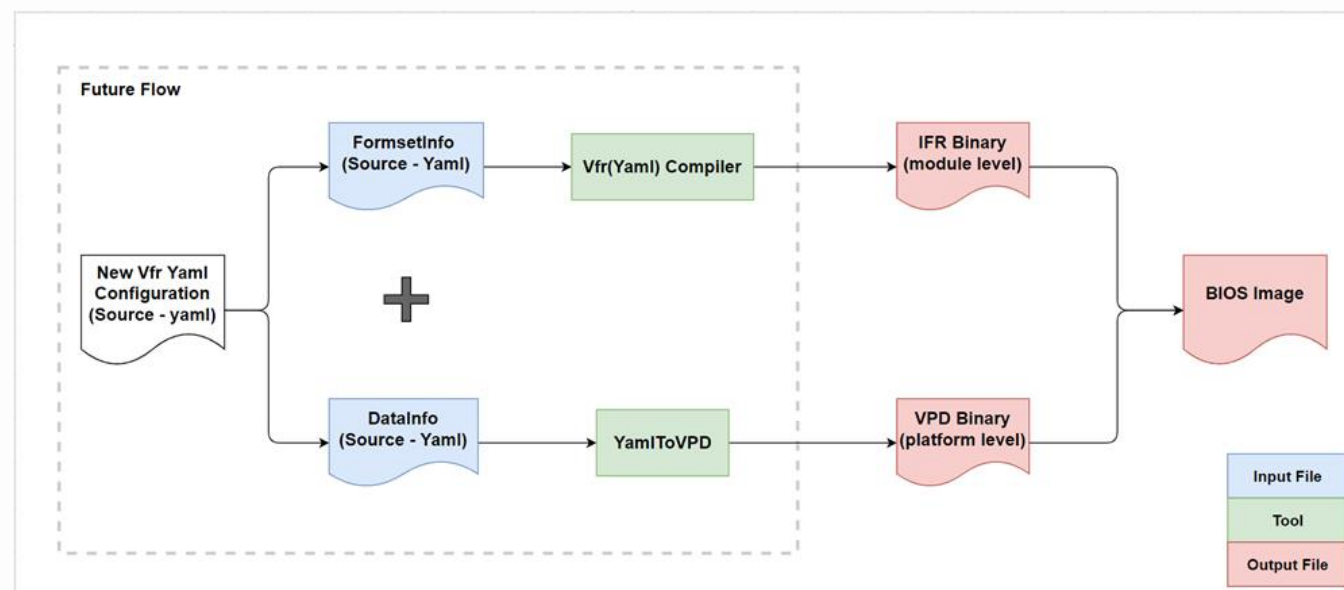
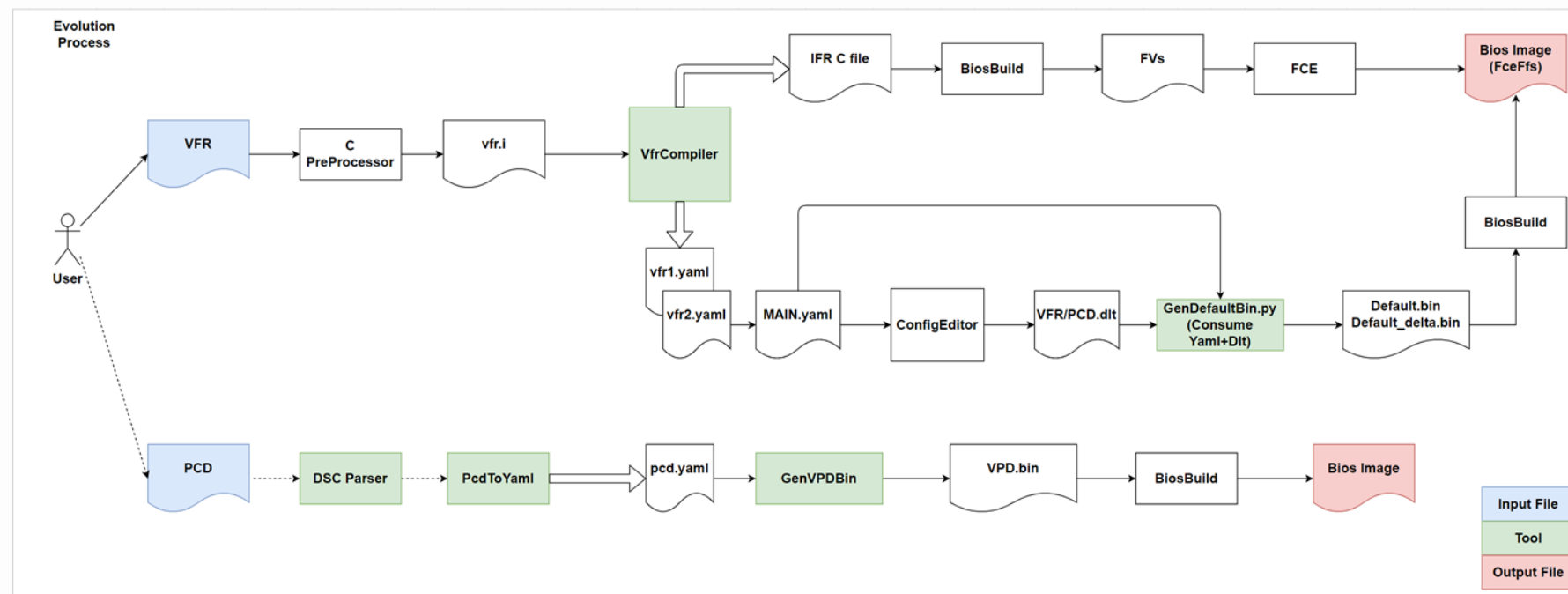
Configuration **YAML** files will be processed by configuration tools like GenCfgData, CfgDataTool, CfgDataStitch in order to generate the configuration header files and binary blobs.

The main platform configuration file is specified in CfgDataDef.yaml. Please note that you may find many YAML files. However, only CfgDataDef.yaml is the primary file used for the platform configuration, and other sub YAML files will be included by the primary YAML file to provide component specific configuration.

An example configuration file in YAML syntax is provided in Figure 15 below.

1 CfgDataDef.yaml X

USF YAML yaml boot configuration



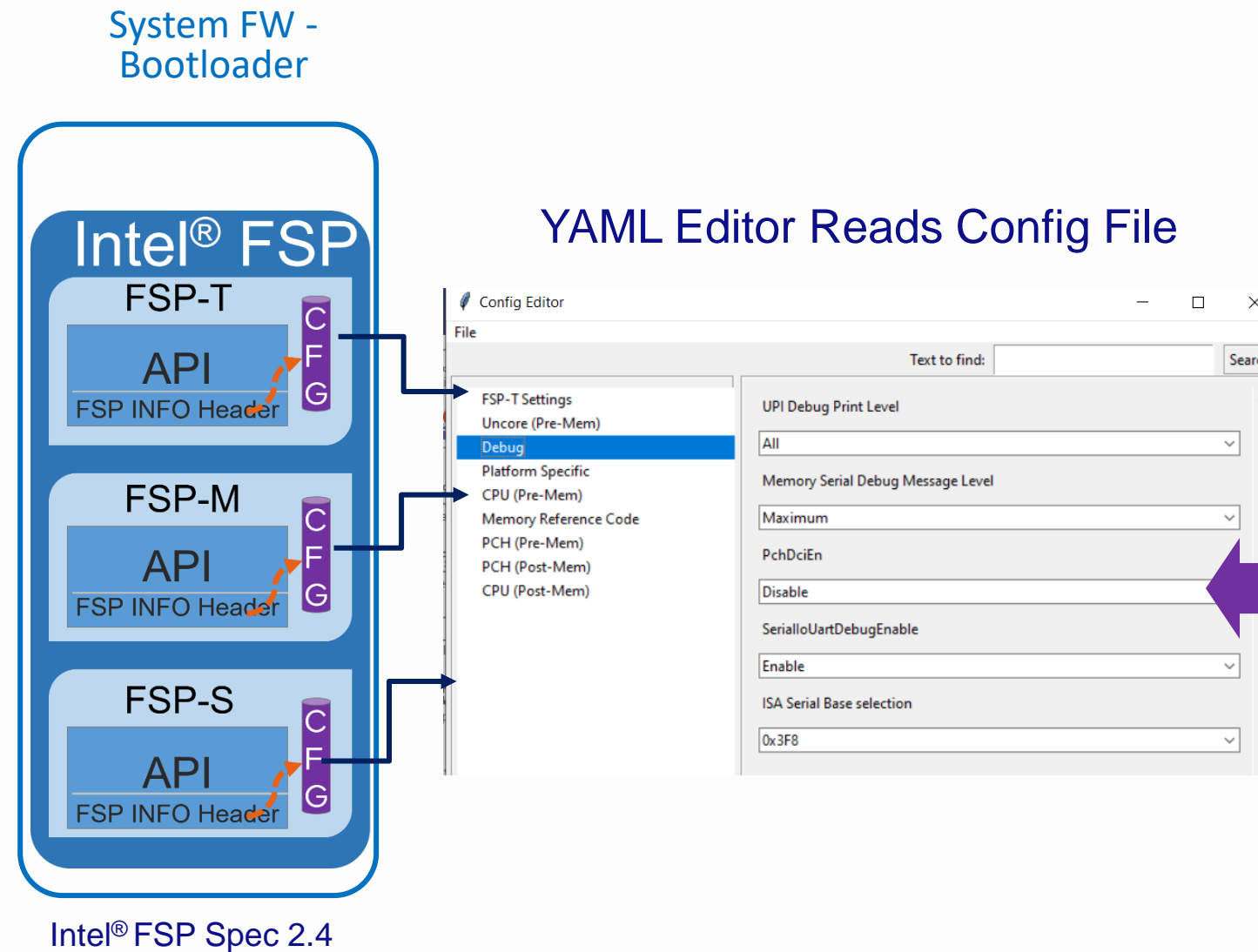
USF YAML focus on the VFR and StructurePCD BIOS Configuration method, expect to unified these source files into YAML type. Based on this expectation, the evolution process in shown as left diagram. For the final flow, all the Bios Configuration related information are from YAML file. [YamlCompiler](#) tool and YamlToVPD tool will be used to generate the binary which will be built into the BIOS Image and be consumed during boot time.

YAML Config Tool for Intel® FSP UPD



YAML UPD Editor Features:

- Read FSP binary information
- Allow patching any BIOS/IFWI image containing FSP UPDs
- Read YAML config format while Boot Setting File (BSF) backward compatible
- Bit format FSP support instead of bytes
- Modifying BSF parameters and export loadable delta files
- FSP 1.x and 2.x format backward compatible
- Search function



<https://github.com/tianocore/edk2/blob/master/IntelFsp2Pkg/Tools/UserManuals/ConfigEditorUserManual.md>



Next Generation of Firmware Configuration

Next Generation of Firmware Configuration



- Uniting Configuration syntax usage to YAML configuration
- Industry wide standard language syntax with advantageous integration across tech-stack



QA

Other events

Flattened devicetree (FDT)

- devicetree.org

See more on FDT at
<https://2023ocpglobal.fnvirtual.app/a/schedule/>



SJCC - Concourse Level
By Sam Berenji

Why Quantum and
level
h. D
me
level
By Jordan Johnson, Sea

Extending PCIe Co
SJCC - Concourse Level
By Sam Kocsis, Chris C

Universal Payload
SJCC - Concourse Level
By Vincent Zimmer, Lea

Standardizing RAS
SJCC - Concourse Level
By Rama Bhimanadhu

Increase Uptime t
SJCC - Concourse Level
By Will Street

Wed, October 18, 12:30pm - 12:50pm | SJCC - Concourse Level - 210CG

Universal Payload for Optimized Firmware Handoff in Server Platforms

- Open System Firmware (OSF)

With the collaborative effort to redefine the standards in the firmware industry, the Universal Payload (UPL) drives key firmware communities and industry partners towards a unified firmware handoff interface.

The UPL allows for decoupling of the platform initialization logic from the more platform-independent but technology specific boot technology, including re-use of the upstream EDKII as the UEFI Payload and upstream Linux kernel and userlands like u-root as the LinuxBoot payload, respectively. This re-use should allow for ease of platform integration, deployment, and servicing as the boot technology concerns are separate from the mainboard & SOC-specific platform details. This should also offer new opportunities for platform designs as the potential platform design targets for payload creators increases. This should also pave a new way towards a common industrial standard lowering the firmware deployment costs and maintenance on server platforms in the long run.

Speakers

Vincent Zimmer
Senior Principal Engineer - Intel

Lean Sheng Tan
Firmware Lead - 9elements

Thanks for attending the UEFI Fall 2023
Developers Conference & Plugfest



For more information on UEFI Forum and UEFI
Specifications, visit <http://www.uefi.org>

presented by

