# Understanding HII Framework and Related Technologies

Subjects: Computer Science, Hardware & Architecture

Contributor: Siyi Chen, Yu-An Tan, Kefan Qiu, Zheng Zhang, Yuanzhang Li, Quanxin Zhang

The Unified Extensible Firmware Interface (UEFI) is currently the most popular industry standard, defining the software interface between an operating system and its underlying platform firmware. HII (Human Interface Infrastructure) is a part of the UEFI specification, which provides a standard framework for managing user interfaces based on UEFI systems.

Keywords: UEFI firmware ; HII ; UEFI UI ; password policy ; sensitive information

## 1. Introduction

The Unified Extensible Firmware Interface (UEFI) [1] is currently the most popular industry standard, defining the software interface between an operating system and its underlying platform firmware. Many industry-leading technology companies have created the UEFI Forum, which defines the specifications of the interfaces [2] used by operating systems and Platform Initialization (PI) specifications. Now, manufacturers provide UEFI-compatible firmware images as a replacement for the Basic Input/Output System (BIOS). Compared to BIOS, UEFI has a better extensibility, a more flexible configuration, and a more standard boot process. The UEFI firmware [3], commonly named the UEFI BIOS on x86 platforms for historic reasons, is the main component of system firmware. The UEFI operating system runs on top of the UEFI firmware and can usually be viewed as a tandem of a UEFI application, serving as an operating system bootloader and a UEFI-compatible kernel, which is aware of the features brought by the UEFI firmware via the UEFI specification. Alongside UEFI, alternative open-source projects such as coreboot [4], libreboot [5], linuxboot [6], and u-boot [6] offer different approaches and philosophies to firmware initialization, focusing on modularity, security, and the freedom to customize the boot process to specific needs and hardware configurations.

HII (Human Interface Infrastructure) [7][8] is a part of the UEFI specification, which provides a standard framework for managing user interfaces based on UEFI systems. More specifically, HII provides a unified way to create and display user interfaces for BIOS settings, system configurations, boot options, etc. Beyond this, it also offers a mechanism to manage these user interfaces, including how to gather user input, how to change system configurations, and how to display the results of the changes. By providing a standard interface and framework, it simplifies the work of hardware manufacturers and operating system developers in creating user interfaces within the UEFI environment, thereby improving development efficiency and reducing development costs. In order to provide more possibilities in system booting, hardware configuration, and system-level management, the UEFI Management System User Interface (UEFI UI) based on HII came into being. This is an interactive interface that allows users to configure and manage UEFI settings. This user interface can be provided by manufacturers and may vary depending on the manufacturer and device. Typically, users can access this user interface by pressing a specific key (such as F2, F10, or Delete) when the computer starts up. In the UEFI UI, users can adjust various hardware and boot options, such as modifying the boot order, enabling or disabling devices, enabling Secure Boot [9], etc. Therefore, it is a very important tool that allows users to customize and adjust their computer systems.

## 2. Understanding HII Framework and Related Technologies

### 2.1. HII Overview

HII is a set of protocols that allow a UEFI driver to provide the ability to register user interface and configuration content with the platform firmware. Unlike legacy option ROMs, the configuration of drivers and controllers is delayed until a platform management utility chooses to use the services of these protocols. UEFI drivers are not allowed to perform setup-like operations outside the context of these protocols. This means that a driver is not allowed to interact with the user outside the context of this protocol.

**Figure 1** shows a basic platform configuration or “setup” model. The drivers and applications install elements (such as fonts, strings, images, and forms) into the HII Database, which acts as a central repository for the entire platform. The Forms Browser uses these elements to render the user interface on the display devices and receive information from the user via HID devices. When complete, the changes made by the user in the Forms Browser are saved, either to the UEFI global variable storage—(GetVariable() and SetVariable())—or to the variable storage provided by the individual drivers.

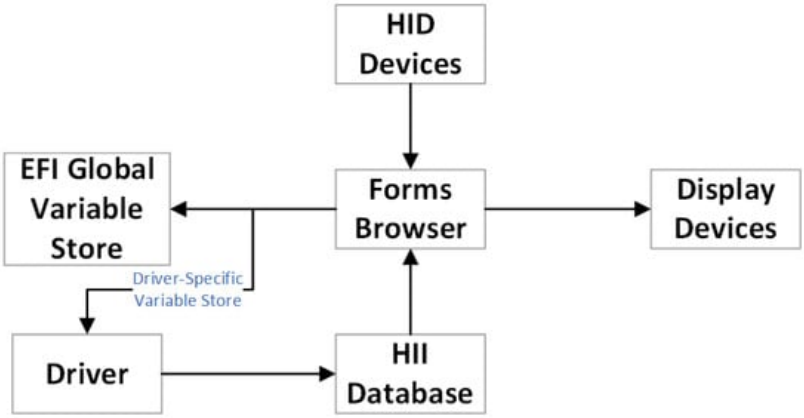


**Figure 1.** UEFI Platform Configuration Overview.

## 2.2. HII Elements

### 2.2.1. HII Databas

The HII database is the resource that serves as the repository of all the form, string, image, and font data for the system. Drivers that contain information destined for the end user will store their data in the HII database. For example, one UEFI module might implement the BIOS Setup program, allowing the user to configure motherboard component settings. Additionally, add-in cards may contain their own UEFI drivers, which, in turn, have their own BIOS Setup-related data. All the UEFI modules that contain BIOS Setup-related data can include their information in the HII database. This architecture is summarized in **Figure 2**.

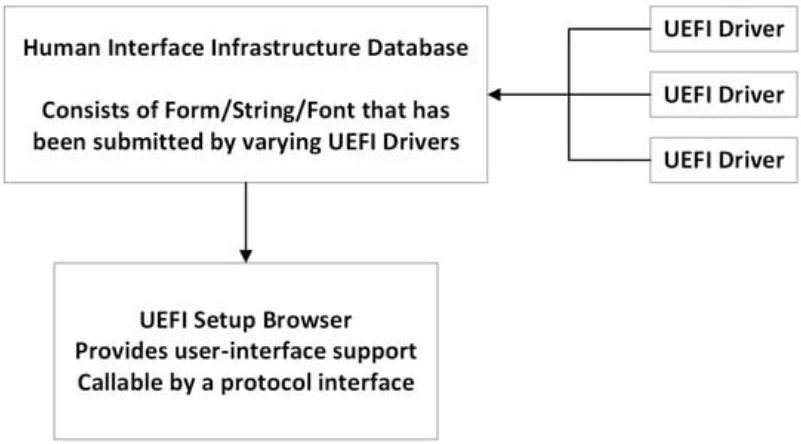


**Figure 2.** HII architecture.

### 2.2.2. Forms

The UEFI specification describes how a UEFI module can present a forms-based interface to a user and a UI element akin to Windows’CFrameWnd or Java’s JFrame. This forms-based interface assumes that each window or screen consists of some window overhead, such as a title and buttons, and a list of user interface controls. For example, these controls could represent individual configuration settings for a UEFI application or driver. Associated with the notion of a form is a Forms Browser—the entity that reads the form data and presents a graphical representation on the display. The Forms Browser provides a forms-based renderer which understands how to read the contents of the forms, interact with the user, and save any resulting values. The Forms Browser uses forms data installed by UEFI modules in the HII database. The Forms Browser organizes the forms so that a user can navigate between the forms, select individual controls, and change the values using a keyboard, touch digitizer, or mouse. When the user has finished making modifications, the Forms Browser saves the values to NVRAM.

### 2.2.3. Strings

Strings in the UEFI environment are defined using the 16-bit UCS-2 character encoding. Strings are another one of the types of resources installed into the HII Database. In order to facilitate localization, programmers reference each string via a unique identifier defined as part of the strings package installed by the UEFI image. Each identifier may have several translations associated with it, e.g., English, French, and Traditional Chinese. When displaying a string, the Forms Browser selects the text to display based on the current platform language setting. The actual text for each language is stored in a separate file, which makes it possible to add and remove language support just by including or excluding language-specific files. Moreover, each string may have font information, including the font family name, font size, and font style, associated with it.

#### 2.2.4. Images

UEFI supports storing images in the HII database. The format of images stored in the HII database was created to conform to the industry standard 1-bit, 4-bit, 8-bit, and 24-bit video memory layouts.

#### 2.2.5. Fonts

UEFI specifies a standard font which is required for all systems that support text display on bit-mapped output devices. The standard font, named “system”, is a fixed pitch font where all characters are either narrow (8 × 19 pixels) or wide (16 × 19 pixels). UEFI also allows for the display of other fonts, both fixed-pitch and variable-pitch. Platform support for fonts beyond the system is optional.

### 2.3. HII Protocol

HII Protocol is a set of protocols and services used for constructing and managing user interfaces. Its primary purpose is to provide a standardized way to describe and manage user interfaces in the UEFI management interface, including everything from BIOS setup menus to full-featured graphical user interfaces.

Through the analysis of these protocols, researchers propose a method to analyze the UEFI password policy produced by AMI vendors. To ascertain the storage location of the password policy in UEFI, protocols related to keyboard input are first needed to be aware of, such as the EFI SIMPLE TEXT INPUT PROTOCOL. Due to the customizability of UEFI, many manufacturers redefine the keyboard input protocol based on this protocol. For instance, the AMI firmware manufacturer defines a keyboard input protocol named AMI EFIKEYCODE PROTOCOL as follows:

```
struct AMI_EFIKEYCODE_PROTOCOL{
    AMI_RESET_EX Reset;
    AMI_READ_EFI_KEY ReadEfikey;
    EFI_EVENT WaitForKeyEx;
    EFI_SET_STATE SetState;
    EFI_REGISTER_KEYSTROKE_NOTIFY RegisterKeyNotify;
    EFI_UNREGISTER_KEYSTROKE_NOTIFY UnregisterKeyNotify;
}
```

Based on the GUID of AMI EFIKEYCODE PROTOCOL as follows:

```
#define AMI_EFIKEYCODE_PROTOCOL_GUID{
    0x0ADFB62D, 0xFF74, 0x484C,
        {0x89, 0x44, 0xF8, 0x5C, 0x4B, 0xEA, 0x87, 0xA8}
}
```

In HII, there is a protocol known as the HII Configuration Processing and Browser Protocol, which encompasses two other protocols: EFI FORM BROWSER2 PROTOCOL and EFI HII CONFIG ROUTING PROTOCOL. EFI FORM BROWSER2 PROTOCOL is used for displaying pages, and HII Drivers can retrieve information from current HII Driver configuration options that users have modified but not yet stored via its BrowserCallback function. EFI HII CONFIG ROUTING PROTOCOL is a global protocol that handles interactions on the Setup interface.

### 2.4. Reverse Engineering of VFR

VFR (Visual Forms Representation) is a high-level descriptive language used to define the layout and behavior of the UEFI settings’ interface. It is closer to source code, providing developers with a convenient way to design and describe forms, controls, and related actions. When VFR is compiled, it is transformed into IFR. IFR is the binary representation of VFR, stored in the UEFI firmware and parsed at runtime to display the corresponding user interface. Since it is in a binary

format, directly reading or editing it is challenging; thus, researchers resort to VFR reverse engineering techniques to convert IFR back to VFR for easier comprehension. The reverse process from IFR to VFR can be employed to view or modify the UEFI firmware settings interface. The basic steps to transition from IFR to VFR are shown in **Figure 3**. Using tools like UEFITool, one can extract the IFR binary segment from the UEFI firmware image. As IFR is the compiled form of VFR, you need a decompiler to convert IFR back to VFR. Such tools might not be as common as forward compilers, but the community offers some, such as the Universal IFR Extractor. Once the VFR representation of the IFR is obtained, one can start analyzing it to understand the user interface layout, options, and behavior of the UEFI firmware.

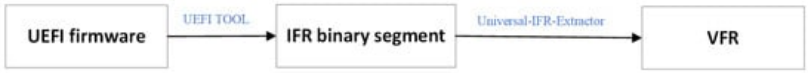


**Figure 3.** VFR Reverse Analysis Steps.

## References


1. UEFI Specification. Available online: https://uefi.org/specifications (accessed on 8 October 2023).
2. Zimmer, V.; Rothman, M.; Marisetty, S. Beyond BIOS: Developing with the Unified Extensible Firmware Interface; Walter de Gruyter GmbH & Co KG: Berlin, Germany, 2017.
3. Jeong, D.; Lee, S. Forensic signature for tracking storage devices: Analysis of UEFI firmware image, disk signature and windows artifacts. Digit. Investig. 2019, 29, 21–27.
4. Butterworth, J.; Kallenberg, C.; Kovah, X.; Herzog, A. Bios chronomancy: Fixing the core root of trust for measurement. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, 4–8 November 2013; pp. 25–36.
5. Sutherland, J.A. On Improving Cybersecurity Through Memory Isolation Using Systems Management Mode. Ph.D. Thesis, Abertay University, Dundee, Scotland, 2019.
6. Banik, S.; Zimmer, V. Understanding the BIOS and Minimalistic Design. In System Firmware: An Essential Guide to Open Source and Embedded Solutions; Apress: Berkeley, CA, USA, 2022; pp. 145–211.
7. Skalsky, N.; Kirch, T.; Rickey, A.; Rothman, M.A. UEFI and the OEM and IHV Community. Intel Technol. J. 2011, 15, 40–67.
8. Leara, W.D. Language Applications for UEFI BIOS. Ph.D. Thesis, The University of Texas at Austin, Austin, TX, USA, 2014.
9. Wilkins, R.; Richardson, B. UEFI secure boot in modern computer security solutions. In Proceedings of the UEFI Forum, Nancy, France, 26 September 2013; pp. 1–10.