[UEFI Practice] HII FrontPage

原创 jiangwei0512 Posted on 2022-02-08 14:51:01 Read 4k Collection 28 Likes 11

Category Column: UEFI Development Basics Article Tags: hii setup uefi

摘要 This article details the implementation of UEFI FrontPage, including initializing the graphics mode, updating interface elements such as the language menu, third-party driver interface, and creating the continue and reset menus. The article shows how to use EFI_FORM_BROWSER2_PROTOCOL to display and update the configured HII data, and how to build and display interface elements such as radio buttons and dynamic content through code analysis.

The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

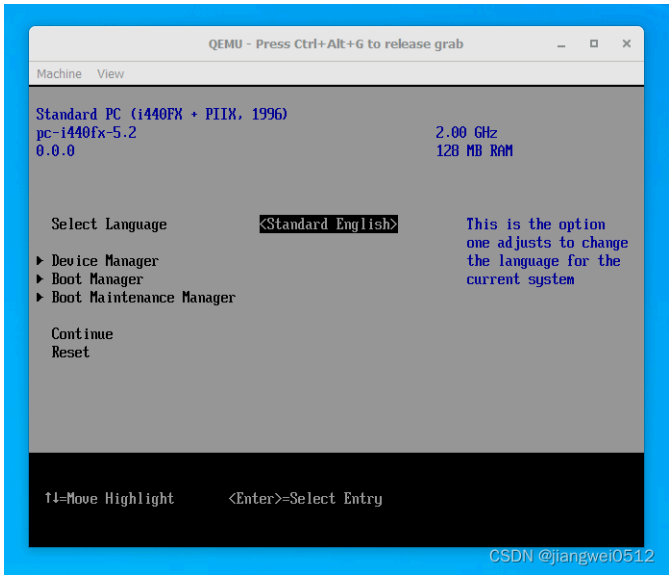
Written in front

UEFI has its own user interface, and its implementation basis is called HII (Human Interface Infrastructure). This article is the first in a series of articles introducing HII implementation. Here we start with the interface in the open source EDK code (called Front Page), introduce its implementation, and further explain the entire HII.

I have written a series of documents related to Setup before, and the content is repeated and supplementary.

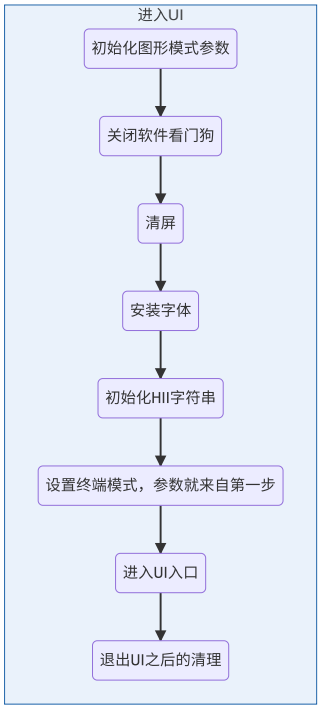
Entry Instructions

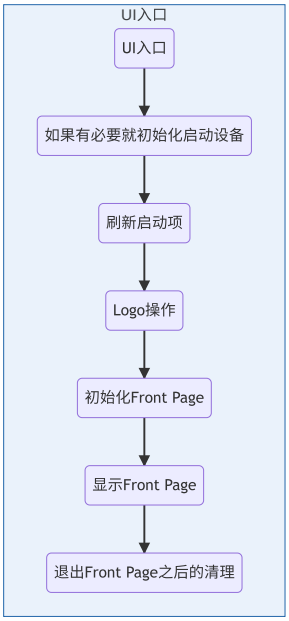
The implementation code of Front Page can be found at https://gitee.com/jiangwei0512/edk2-beni). The results after compilation and execution are as follows:



It corresponds to a startup program UiApp, the module code is MdeModulePkg\Application\UiApp\UiApp.inf, and the entry function of the module is `InitializeUserInterface()` (located in MdeModulePkg\Application\UiApp\FrontPage.c).

The general process of initialization is as follows:

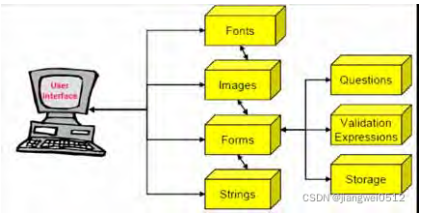




The two most important parts in the above process are "Initialize Front Page" and "Display Front Page", which correspond to two functions `InitializeFrontPage()` and `CallFrontPage()` . They can be viewed together. There are two main things to do: one is to prepare materials, where the materials refer to HII data represented by uni files, vfr files, etc.; the other is to display these materials, which is `EFI_FORM_BROWSER2_PROTOCOL` completed through a Protocol. This Protocol provides two interfaces `SendForm()` for displaying the configured HII, `BrowserCallback()` which will be called by the callback function to obtain and set interface elements.

Interface Elements

There are four kinds of elements that make up an interface in HII, namely strings, forms, fonts, and images, as shown in the following figure (the Questions on the far right are part of the structure, which can be ignored for now and will be introduced later):



Strings are generated by uni files , structures are generated by vfr files , fonts are not introduced for now, and images are not particularly easy to introduce. In this example, the Front Page uses structures, strings, and fonts, but not images.

Element Update

Element updates mainly occur `InitializeFrontPage()` in functions, the corresponding code is:

c AI generated projects 登录复制 run

```c
//
//Updata Front Page banner strings
//
UpdateFrontPageBannerStrings ();

//
// Update front page menus.
//
UpdateFrontPageForm();
```

- The former `UpdateFrontPageBannerStrings()` mainly obtains the value of the string corresponding to **the tag** ① (Token) and sets it to the corresponding **tag** ② . It initializes the static part of the upper half of the Front Page:

Taking the code as an example, it looks like the following:

c AI generated projects 登录复制 run

```c
    NewString = HiiGetString (gFrontPagePrivate.HiiHandle, STRING_TOKEN (STR_FRONT_PAGE_COMPUTER_MODEL), NULL);
    UiCustomizeFrontPageBanner (1, TRUE, &NewString);
    HiiSetString (gFrontPagePrivate.HiiHandle, STRING_TOKEN (STR_FRONT_PAGE_COMPUTER_MODEL), NewString, NULL);
    FreePool (NewString);
```

The tag here `STR_FRONT_PAGE_COMPUTER_MODEL` appears twice. Although the name is the same, it comes from different files, namely the uni file:

json AI generated projects 登录复制

```json
#string STR_FRONT_PAGE_COMPUTER_MODEL  #language en-US  ""
                                       #language fr-FR  ""
```

and from the vfr file:

json AI generated projects 登录复制

```vfr
    banner
      title = STRING_TOKEN(STR_FRONT_PAGE_COMPUTER_MODEL),
      line  1,
      align left;
```

The operation of **mark** ① `HiiGetString()` is to get the string, and the operation of **mark** ② `HiiSetString()` is to set the string. However, the two should be consistent, so there is no need to distinguish them specifically.

- The latter `UpdateFrontPageForm()` updates the other dynamic parts, and the dynamic parts can be seen in the vfr file and can be seen between the two opcodes:

json AI generated projects 登录复制

```vfr
    label LABEL_FRANTPAGE_INFORMATION;
    //
    // This is where we will dynamically add a Action type op-code to show
    // the platform information.
    //
    label LABEL_END;
```

The corresponding code:

c

```c
VOID
UpdateFrontPageForm (
  VOID
  )
{
  VOID                        *StartOpCodeHandle;
  VOID                        *EndOpCodeHandle;
```

```c
  EFI_IFR_GUID_LABEL          *StartGuidLabel;
  EFI_IFR_GUID_LABEL          *EndGuidLabel;

  //
  // Allocate space for creation of UpdateData Buffer
  //
  StartOpCodeHandle = HiiAllocateOpCodeHandle ();
  ASSERT (StartOpCodeHandle != NULL);

  EndOpCodeHandle = HiiAllocateOpCodeHandle ();
  ASSERT (EndOpCodeHandle != NULL);
  //
  // Create Hii Extend Label OpCode as the start opcode
  //
  StartGuidLabel = (EFI_IFR_GUID_LABEL *) HiiCreateGuidOpCode (StartOpCodeHandle, &gEfiIfrTianoGuid, NULL, sizeof (EFI_IFR_GUID_LABEL));
  StartGuidLabel->ExtendOpCode = EFI_IFR_EXTEND_OP_LABEL;
  StartGuidLabel->Number       = LABEL_FRANTPAGE_INFORMATION;  // 对应vfr中的LABEL_FRANTPAGE_INFORMATION
  //
  // Create Hii Extend Label OpCode as the end opcode
  //
  EndGuidLabel = (EFI_IFR_GUID_LABEL *) HiiCreateGuidOpCode (EndOpCodeHandle, &gEfiIfrTianoGuid, NULL, sizeof (EFI_IFR_GUID_LABEL));
  EndGuidLabel->ExtendOpCode = EFI_IFR_EXTEND_OP_LABEL;
  EndGuidLabel->Number       = LABEL_END;   // 对应vfr中的LABEL_END

  //
  //Updata Front Page form
  //
  UiCustomizeFrontPage (
    gFrontPagePrivate.HiiHandle,
    StartOpCodeHandle
    );

  HiiUpdateForm (
    gFrontPagePrivate.HiiHandle,
    &mFrontPageGuid,
    FRONT_PAGE_FORM_ID,
    StartOpCodeHandle,
    EndOpCodeHandle
    );

  HiiFreeOpCodeHandle (StartOpCodeHandle);
  HiiFreeOpCodeHandle (EndOpCodeHandle);
}
```

收起 ^

`StartOpCodeHandle` The two parts (and) here `EndOpCodeHandle` combine to form the rest of the Front Page, and the process is as follows:

1. Create two OpCodeHandles through the function `HiiAllocateOpCodeHandle()` , which correspond to the same structure:

c AI generated projects 登录复制 run

```c
typedef struct {
  UINT8  *Buffer;
  UINTN  BufferSize;
  UINTN  Position;
} HII_LIB_OPCODE_BUFFER;
```

After the structure is created `Buffer` , there is a space of 0x200 bytes; `BufferSize` that is, 0x200; `Position` initialized to 0, it is equivalent to a container for storing other opcodes.

2. Create two OpCodes, which will use the OpCodeHandle created earlier:

c AI generated projects 登录复制 run

```c
StartGuidLabel = (EFI_IFR_GUID_LABEL *) HiiCreateGuidOpCode (StartOpCodeHandle, &gEfiIfrTianoGuid, NULL, sizeof (EFI_IFR_GUID_LABEL));
StartGuidLabel->ExtendOpCode = EFI_IFR_EXTEND_OP_LABEL;
StartGuidLabel->Number       = LABEL_FRANTPAGE_INFORMATION;
```

`HiiCreateGuidOpCode()` The first parameter is OpCodeHandle; the second parameter is a GUID; the third parameter is optional and can be NULL; the fourth parameter is the size of the component element structure. This example creates two `EFI_IFR_GUID_LABEL` structures (`label` corresponding to those in the vfr file), which are also `HiiCreateGuidOpCode()` the return values of , and their structures are as follows:

c AI generated projects 登录复制 run

```c
///
/// Label opcode.
///
typedef struct _EFI_IFR_GUID_LABEL {
  EFI_IFR_OP_HEADER   Header;
  ///
  /// EFI_IFR_TIANO_GUID.
  ///
  EFI_GUID            Guid;
  ///
  /// EFI_IFR_EXTEND_OP_LABEL.
  ///
  UINT8               ExtendOpCode;
  ///
  /// Label Number.
  ///
  UINT16              Number;
} EFI_IFR_GUID_LABEL;
```

收起 ^

The following two lines of code are used to initialize `EFI_IFR_GUID_LABEL` the last two parameters of the structure, which `Number` correspond to the nar
`LABEL_FRANTPAGE_INFORMATION` and `LABEL_END` .

3. To `OpCode` customize , actually, is to create custom elements between `StartOpCodeHandle` and . `EndOpCodeHandle`
4. Update the Front Page structure. The updated part is the custom elements added in the previous code.
5. Release resources.

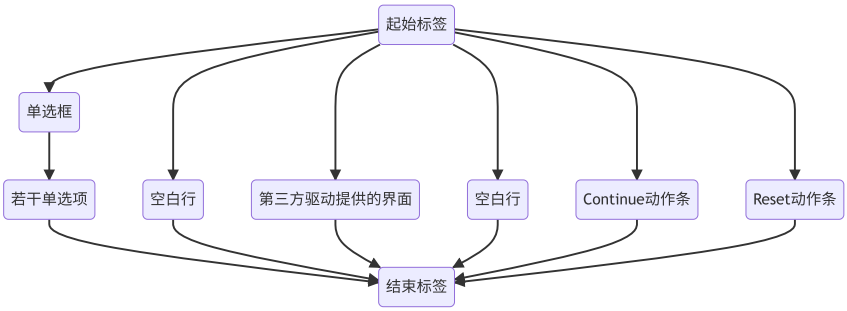
The most important thing here is step 3, which corresponds to the following function:

c AI generated projects 登录复制 run

```c
VOID
UiCustomizeFrontPage (
  IN EFI_HII_HANDLE  HiiHandle,
  IN VOID            *StartOpCodeHandle
  )
{
  //
  // Create "Select Language" menu with Oneof opcode.
  //
  UiCreateLanguageMenu (HiiHandle, StartOpCodeHandle);

  //
  // Create empty line.
  //
  UiCreateEmptyLine(HiiHandle, StartOpCodeHandle);

  //
  // Find third party drivers which need to be shown in the front page.
  //
  UiListThirdPartyDrivers (HiiHandle, &gEfiIfrFrontPageGuid, NULL, StartOpCodeHandle);

  //
  // Create empty line.
  //
  UiCreateEmptyLine(HiiHandle, StartOpCodeHandle);

  //
  // Create "Continue" menu.
  //
  UiCreateContinueMenu(HiiHandle, StartOpCodeHandle);

  //
  // Create reset menu.
  //
  UiCreateResetMenu(HiiHandle, StartOpCodeHandle);
}
```

收起 ^

Basically, the parts shown in the figure have corresponding functions, but the black parts at the bottom do not. They are generated according to specific circumstances, such as being displayed `<Enter>= Select Entry` because of , and they will be displayed as long as the above elements are created. The components displayed by the action have an inclusive relationship, as shown in the following figure: `UiCreateLanguageMenu() ↑↓=Move Highlight`



The following sections describe how to create the dynamic display portion of the diagram.

Create a menu

`UiCreateLanguageMenu()` The specific code:

c AI generated projects 登录复制 run

```c
VOID
UiCreateLanguageMenu (
  IN EFI_HII_HANDLE              HiiHandle,
  IN VOID                        *StartOpCodeHandle
  )
{
  CHAR8                      *LangCode;
  CHAR8                      *Lang;
  UINTN                      LangSize;
  CHAR8                      *CurrentLang;
  UINTN                      OptionCount;
  CHAR16                     *StringBuffer;
  VOID                       *OptionsOpCodeHandle;
  UINTN                      StringSize;
  EFI_STATUS                 Status;
  EFI_HII_STRING_PROTOCOL    *HiiString;

  Lang         = NULL;
  StringBuffer = NULL;

  //
  // Init OpCode Handle and Allocate space for creation of UpdateData Buffer
  //
  OptionsOpCodeHandle = HiiAllocateOpCodeHandle ();
```

```
  ASSERT (OptionsOpCodeHandle != NULL);

  GetEfiGlobalVariable2 (L"PlatformLang", (VOID**)&CurrentLang, NULL);

  //
  // Get Support language list from variable.
  //
  GetEfiGlobalVariable2 (L"PlatformLangCodes", (VOID**)&gLanguageString, NULL);
  if (gLanguageString == NULL) {
    gLanguageString = AllocateCopyPool (
                              AsciiStrSize ((CHAR8 *) PcdGetPtr (PcdUefiVariableDefaultPlatformLangCodes)),
                              (CHAR8 *) PcdGetPtr (PcdUefiVariableDefaultPlatformLangCodes)
                              );
    ASSERT (gLanguageString != NULL);
  }

  if (gLanguageToken == NULL) {
    //
    // Count the language list number.
    //
    LangCode = gLanguageString;
    Lang = AllocatePool (AsciiStrSize (gLanguageString));
    ASSERT (Lang != NULL);

    OptionCount = 0;
    while (*LangCode != 0) {
      GetNextLanguage (&LangCode, Lang);
      OptionCount ++;
    }

    //
    // Allocate extra 1 as the end tag.
    //
    gLanguageToken = AllocateZeroPool ((OptionCount + 1) * sizeof (EFI_STRING_ID));
    ASSERT (gLanguageToken != NULL);

    Status = gBS->LocateProtocol (&gEfiHiiStringProtocolGuid, NULL, (VOID **) &HiiString);
    ASSERT_EFI_ERROR (Status);

    LangCode     = gLanguageString;
    OptionCount  = 0;
    while (*LangCode != 0) {
      GetNextLanguage (&LangCode, Lang);

      StringSize = 0;
      Status = HiiString->GetString (HiiString, Lang, HiiHandle, PRINTABLE_LANGUAGE_NAME_STRING_ID, StringBuffer, &StringSize, NULL);
      if (Status == EFI_BUFFER_TOO_SMALL) {
        StringBuffer = AllocateZeroPool (StringSize);
        ASSERT (StringBuffer != NULL);
        Status = HiiString->GetString (HiiString, Lang, HiiHandle, PRINTABLE_LANGUAGE_NAME_STRING_ID, StringBuffer, &StringSize, NULL);
        ASSERT_EFI_ERROR (Status);
      }

      if (EFI_ERROR (Status)) {
        LangSize = AsciiStrSize (Lang);
        StringBuffer = AllocatePool (LangSize * sizeof (CHAR16));
        ASSERT (StringBuffer != NULL);
        AsciiStrToUnicodeStrS (Lang, StringBuffer, LangSize);
      }

      ASSERT (StringBuffer != NULL);
      gLanguageToken[OptionCount] = HiiSetString (HiiHandle, 0, StringBuffer, NULL);
      FreePool (StringBuffer);

      OptionCount++;
    }
  }

  ASSERT (gLanguageToken != NULL);
  LangCode = gLanguageString;
  OptionCount = 0;
  if (Lang == NULL) {
    Lang = AllocatePool (AsciiStrSize (gLanguageString));
    ASSERT (Lang != NULL);
  }
  while (*LangCode != 0) {
    GetNextLanguage (&LangCode, Lang);

    if (CurrentLang != NULL && AsciiStrCmp (Lang, CurrentLang) == 0) {
      HiiCreateOneOfOptionOpCode (
        OptionsOpCodeHandle,
        gLanguageToken[OptionCount],
        EFI_IFR_OPTION_DEFAULT,
        EFI_IFR_NUMERIC_SIZE_1,
        (UINT8) OptionCount
        );
      gCurrentLanguageIndex = (UINT8) OptionCount;
    } else {
      HiiCreateOneOfOptionOpCode (
        OptionsOpCodeHandle,
        gLanguageToken[OptionCount],
        0,
        EFI_IFR_NUMERIC_SIZE_1,
        (UINT8) OptionCount
        );
    }

    OptionCount++;
  }
```

```c

  if (CurrentLang != NULL) {
    FreePool (CurrentLang);
  }
  FreePool (Lang);

  HiiCreateOneOfOpCode (
    StartOpCodeHandle,
    FRONT_PAGE_KEY_LANGUAGE,
    0,
    0,
    STRING_TOKEN (STR_LANGUAGE_SELECT),
    STRING_TOKEN (STR_LANGUAGE_SELECT_HELP),
    EFI_IFR_FLAG_CALLBACK,
    EFI_IFR_NUMERIC_SIZE_1,
    OptionsOpCodeHandle,
    NULL
    );
}
```

收起 ^

Most of the code here is just the data you need to use when creating options. You don't need to pay special attention to it. There are only two important steps here:

1. Create a new one `HII_LIB_OPCODE_BUFFER` to store the radio button, which `HiiCreateOneOfOptionOpCode()` is placed in the function `HII_LIB_OPCODE_BUFFER`;
2. The radio options are stored `HII_LIB_OPCODE_BUFFER` in a radio button and stored in the upper layer `HII_LIB_OPCODE_BUFFER`.

Create a blank line

The blank line here corresponds to a `EFI_IFR_SUBTITLE_OP` (Subtitle statement), and the corresponding structure is:

c AI generated projects 登录复制 run

```c
typedef struct _EFI_IFR_SUBTITLE {
  EFI_IFR_OP_HEADER        Header;
  EFI_IFR_STATEMENT_HEADER Statement;
  UINT8                    Flags;
} EFI_IFR_SUBTITLE;
```

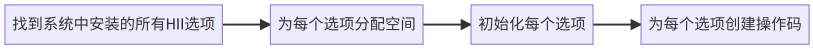
But without actual data, it becomes a blank line. Note that the parameters when creating it are all 0:

c AI generated projects 登录复制 run

```c
VOID
UiCreateEmptyLine (
  IN EFI_HII_HANDLE              HiiHandle,
  IN VOID                        *StartOpCodeHandle
  )
{
  HiiCreateSubTitleOpCode (StartOpCodeHandle, STRING_TOKEN (STR_NULL_STRING), 0, 0, 0);
}
```

Creating an interface for third-party drivers

Execution process:



The final operation is also to create an opcode:

c AI generated projects 登录复制 run

```c
    HiiCreateGotoExOpCode (
      StartOpCodeHandle,
      0,
      gHiiDriverList[Index].PromptId,
      gHiiDriverList[Index].HelpId,
      0,
      (EFI_QUESTION_ID) (Index + FRONT_PAGE_KEY_DRIVER),
      0,
      &gHiiDriverList[Index].FormSetGuid,
      gHiiDriverList[Index].DevicePathId
    );
```

收起 ^

The corresponding opcodes may be `EFI_IFR_REF_OP`, `EFI_IFR_REF2_OP`, `EFI_IFR_REF3_OP` and `EFI_IFR_REF4_OP`, depending on the passed in parameter values.

Creating a Continue/Reset Menu

Both correspond to `EFI_IFR_ACTION_OP` opcodes, and the corresponding implementations are:

c AI generated projects 登录复制 run

```c
/**
  Create continue menu in the front page.

  @param[in]    HiiHandle           The hii handle for the Uiapp driver.
  @param[in]    StartOpCodeHandle   The opcode handle to save the new opcode.

 **/
VOID
UiCreateContinueMenu (
  IN EFI_HII_HANDLE              HiiHandle,
  IN VOID                        *StartOpCodeHandle
  )
{
  HiiCreateActionOpCode (
    StartOpCodeHandle,

```

```
    FRONT_PAGE_KEY_CONTINUE,
    STRING_TOKEN (STR_CONTINUE_PROMPT),
    STRING_TOKEN (STR_CONTINUE_PROMPT),
    EFI_IFR_FLAG_CALLBACK,
    0
    );
}

/**
  Create Reset menu in the front page.

  @param[in]    HiiHandle           The hii handle for the Uiapp driver.
  @param[in]    StartOpCodeHandle   The opcode handle to save the new opcode.

**/
VOID
UiCreateResetMenu (
  IN EFI_HII_HANDLE              HiiHandle,
  IN VOID                        *StartOpCodeHandle
  )
{
  HiiCreateActionOpCode (
    StartOpCodeHandle,
    FRONT_PAGE_KEY_RESET,
    STRING_TOKEN (STR_RESET_STRING),
    STRING_TOKEN (STR_RESET_STRING),
    EFI_IFR_FLAG_CALLBACK,
    0
    );
}
```

收起 ^

As you can see from the code, the only difference is in the display and `QuestionId`. The latter judges its value in the callback function and performs different operations. The code is in `UiSupportLibCallbackHandler()`. Part of the code is as follows:

c AI generated projects 登录复制 run

```c
    switch (QuestionId) {
    case FRONT_PAGE_KEY_CONTINUE:
      //
      // This is the continue - clear the screen and return an error to get out of FrontPage loop
      //
      *ActionRequest = EFI_BROWSER_ACTION_REQUEST_EXIT;
      break;

    case FRONT_PAGE_KEY_LANGUAGE:
      *Status = LanguageChangeHandler(Value);
      break;

    case FRONT_PAGE_KEY_RESET:
      //
      // Reset
      //
      gRT->ResetSystem (EfiResetCold, EFI_SUCCESS, 0, NULL);
      *Status = EFI_UNSUPPORTED;

    default:
      break;
    }
```

收起 ^