UEFI development exploration 60-VFR files and other resource files 1 (adding items to BIOS setup)

原创 luobing4365 Modified on 2023-03-28 16:08:14 Read 5.8k Collection 41 Likes 15

Category Column: UEFI Development Article Tags: uefi uefi bios UEFI bottom-level development VFR formset

(Please keep-> Author: Luo Bingluobing4365's blog_CSDN blog-UEFI development, assembly language exploration, embedded development field blogger)

How to use VFR files is a topic that I am interested in recently. Including VFR files, UNI files and IDF files are all resource files. The use of UNI files has been discussed in the previous Hii example. However, how to use VFR files has not been discussed.

The next few blogs will take modifying BIOS Setup (of course, using the image compiled by OvmfPkg as the BIOS file) as an example to demonstrate how to use VFR files and related resource files.

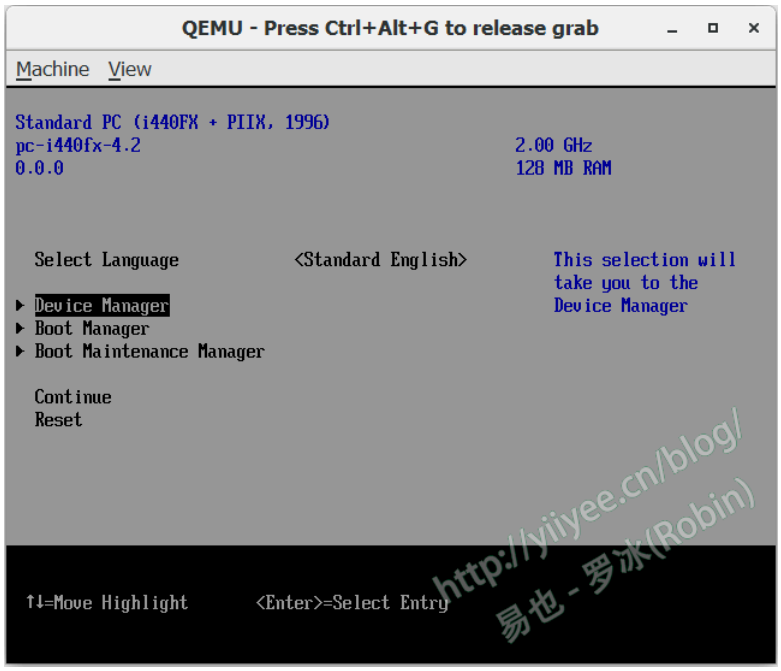

Figure 1 BIOS interface of OVMF image

This example is based on the sample driver MyWizardDriver provided by Intel, and adds strings and forms. The modification steps are as follows.

1 Modify MyWizardDriverNVDataStruc.h

Add the declaration of the GUID (MYWIZARDDRIVER_FORMSET_GUID):

```
#define MYWIZARDDRIVER_FORMSET_GUID { 0x5411db09, 0xe5f7, 0x4158, {0xa5, 0xc5, 0x2d, 0xbe, 0xa4,0x95, 0x34, 0xff} }
```

In the example modifications, the modified parts are marked with "robin add". If it is a multi-line modification, "begin" and "end" comments are added.

2 Modify MyWizardDriver.vfr

Delete all other contents in the VFR file and add a custom form definition as follows:

```
#include “MyWizardDriverNVDataStruc.h”
formset
  guid = MYWIZARDDRIVER_FORMSET_GUID,
  ……
endform;
endformset;
```

The definitions in the VFR file will be described after the program demonstration, and their syntax and how they correspond to the display.

3 Modify MyWizardDriver.uni

Add custom strings. UNI files and string definitions have been described in previous blogs, so I won't explain them in detail here.

```
#string STR_SAMPLE_FORM_SET_TITLE #language en “My Wizard DriverSample Formset”
#string STR_SAMPLE_FORM_SET_HELP #language en “Help for SampleFormset”
……
```

4 Modify MyWizardDriver.h

Add header file:

```
#include <Protocol/HiiConfigRouting.h>
#include <Protocol/FormBrowser2.h>
#include <Protocol/HiiString.h>
#include <Library/DevicePathLib.h>
```

And add a custom data structure . The code is added as shown below:

```
extern EFI_HII_CONFIG_ACCESS_PROTOCOL  gMyWizardDriverHiiConfigAccess;

//robin add 2020-06-26 10:24:18  begin ================================
#define MYWIZARDDRIVER_DEV_SIGNATURE SIGNATURE_32 ('m', 'w','d', 'r')
// Need a Data structure for HII routing and accessing
typedef struct {
  UINT32 Signature;

  EFI_HANDLE Handle;
  MYWIZARDDRIVER_CONFIGURATION Configuration;

  EFI_HANDLE DriverHandle[2];
  EFI_HII_HANDLE HiiHandle[2];
//
// Consumed protocol
//
  EFI_HII_DATABASE_PROTOCOL *HiiDatabase;
  EFI_HII_STRING_PROTOCOL *HiiString;
  EFI_HII_CONFIG_ROUTING_PROTOCOL *HiiConfigRouting;
  EFI_FORM_BROWSER2_PROTOCOL *FormBrowser2;
//
// Produced protocol
//
  EFI_HII_CONFIG_ACCESS_PROTOCOL ConfigAccess;
} MYWIZARDDRIVER_DEV;

#define MYWIZARDDRIVER_DEV_FROM_THIS(a) CR (a,MYWIZARDDRIVER_DEV, Confi
#pragma pack(1)
///
/// HII specific Vendor Device Path definition.
///
typedef struct {
  VENDOR_DEVICE_PATH VendorDevicePath;
  EFI_DEVICE_PATH_PROTOCOL End;
} HII_VENDOR_DEVICE_PATH;

#pragma pack()
//robin add end ================================
//
// Include files with function prototypes
//
#include "DriverBinding.h"
```


Figure 2 MyWizardDriver.h code addition

5 Modify MyWizardDriver.c

Add global variable definition:

```
EFI_GUID   mMyWizardDriverVarGuid = MYWIZARDDRIVER_VAR_GUID;

//robin add 2020-06-26 10:27:15 begin ===============
//HII support
EFI_GUID mMyWizardDriverFormSetGuid = MYWIZARDDRIVER_FORMSET_GUID;
CHAR16 mIfrVariableName[] = L"MWD_IfrNVData";
EFI_HANDLE mDriverHandle[2] = {NULL, NULL};
MYWIZARDDRIVER_DEV *PrivateData = NULL;
// HII support for Device Path
HII_VENDOR_DEVICE_PATH mHiiVendorDevicePath = {
  {
    {
      HARDWARE_DEVICE_PATH,
      HW_VENDOR_DP,
      {
        (UINT8) (sizeof (VENDOR_DEVICE_PATH)),
        (UINT8) ((sizeof (VENDOR_DEVICE_PATH)) >> 8)
      }
    },
    MYWIZARDDRIVER_FORMSET_GUID
  },
  {
    END_DEVICE_PATH_TYPE,
    END_ENTIRE_DEVICE_PATH_SUBTYPE,
    {
      (UINT8) (END_DEVICE_PATH_LENGTH),
      (UINT8) ((END_DEVICE_PATH_LENGTH) >> 8)
    }
  }
};
//robin add edd ========================================

CHAR16     mVariableName[] = L"MWD_NVData";
```


Figure 3 Source file global variable addition

Modify the function MyWizardDriverDriverEntryPoint(). There are many modifications, so please check the source code. The most important code is to add the form to the BIOS Setup through SetVariable().

The code is ready and needs to be compiled. The OVMF image must be compiled at the same time. The code is still in RobinPkg, and the compilation command is as follows:

```
build -p RobinPkg\RobinPkg.dsc -m RobinPkg\Drivers\MyWizardDriver\MyWizardDriver.inf -a X64
```

And compile the OVMF image:

```
build -p OvmfPkg\OvmfPkgX64.dsc -a X64
```

The strange thing is that several OVMF images I compiled have some problems. They are not normal when tested, but can be tested normally with a previously compiled OVMF image. Now I don't know what is the cause. In the link provided at the end of the article, I have given the entire test environment, including OVMF images and batch processing.

Copy the compiled MyWizardDriver.efi to the specified folder hd-contents and start the Qemu simulation environment:

```
qemu-system-x86_64.exe -bios OVMF.fd -hda fat:rw:hda-contents -net none
```

Enter UEFI Shell and execute the following command:

```
Shell> FS0:
FS0:\> load MyWizardDriver.efi
FS0:\> exit
```

Enter BIOS Setup, select Device Manager-My Wizard DriverSample Formset, and you can see the interface shown in Figure 4, indicating that the form is added successfully:

Figure 4 Schematic diagram of adding a form

You can use the spacebar or enter key to modify the options. When modifying options, a yellow string "Configuration changed" will appear in the lower right corner.

More: VFR File Syntax

Forms are the organizational form of user interaction, and ultimately enter the EDK2 framework in the binary form of IFR (Internal Forms Representation). IFR is generated by VFR files, similar to the relationship between obj files and source files. From the perspective of the platform framework, forms are at the center of interaction:

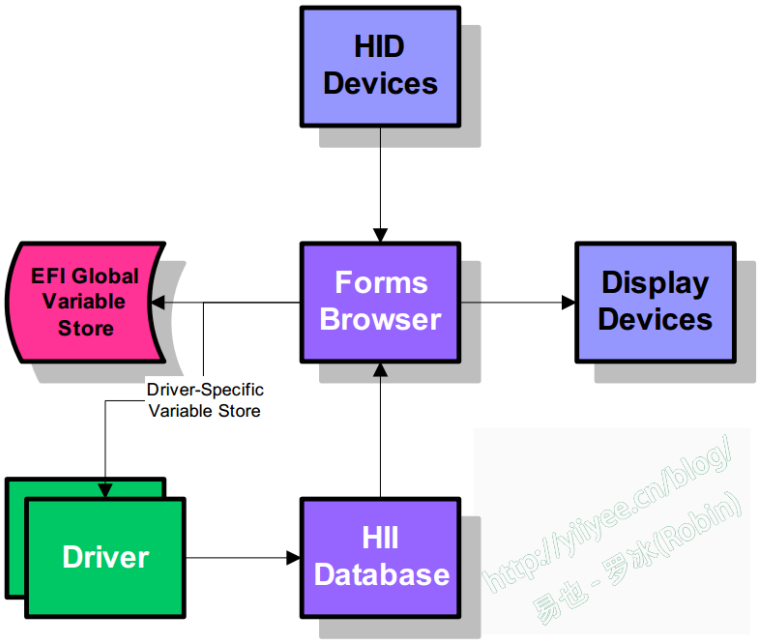

Figure 5 Human-computer interaction structure diagram of the platform architecture

From a programmer's perspective, all resource files are ultimately part of the user UI, and its structure is as follows:

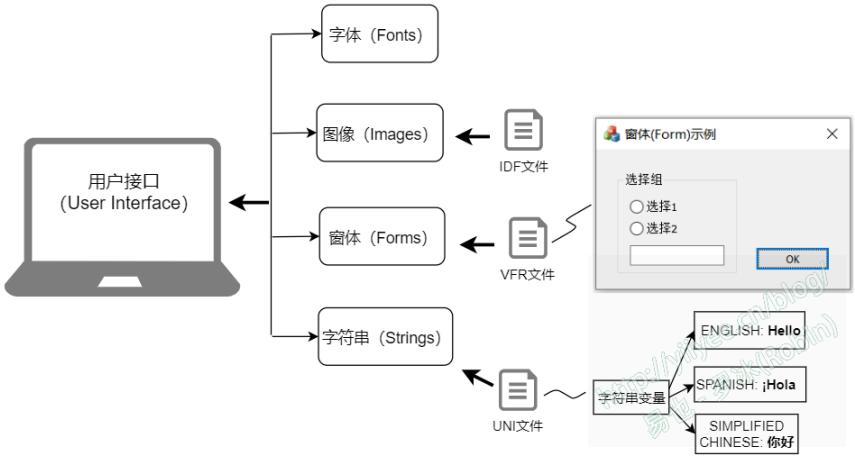

Figure 6 Resource file

VFR files use BNF syntax, which is different from DSC files, which use "#" as a comment mark. There are also two keywords, "#define" and "#include", which are used to define and include header files, similar to C syntax.

Formset is the most important part of a VFR file, and is the most commonly used structure for composing forms. An example is as follows:

```
f ormset
guid = {0xcc5ebb4f, 0xf562, 0x11e7, {0x92, 0x11, 0xf4, 0x8c, 0x50, 0x49, 0xe3, 0xa4}},
title = STRING_TOKEN(STR_SAMPLE_FORM_SET_TITLE),
help = STRING_TOKEN(STR_SAMPLE_FORM_SET_HELP),
  classguid = EFI_HII_PLATFORM_SETUP_FORMSET_GUID
form formid = 1, title = STRING_TOKEN(STR_SAMPLE_FORM1_TITLE);
  ...
endform;
endformset;
```

formset is a keyword used to mark the beginning of the entire form, and appears in pairs with endformset, which marks the end of the form. The meanings of the keywords are:

guid: the GUID value that identifies this formset;
title: the string title that identifies this formset in the interface;
help: the help information of this formset is displayed on the interface;
classguid: the GUID value of the page where this formset is mounted.

The options for this formset are under Device Manager on the initial page, as shown in Figure 7.

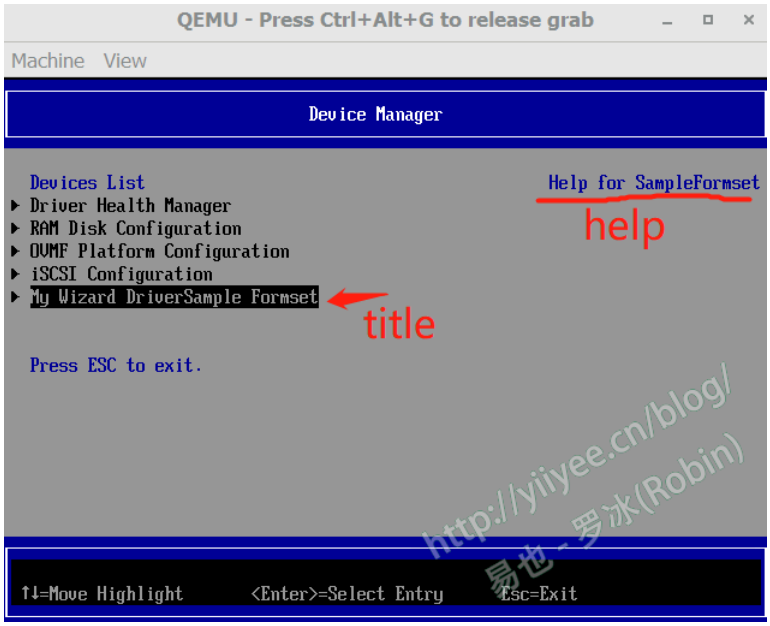

Figure 7 Formset defined in VFR file

In the formset collection, you can use "form" and "endform" to define a complete page, such as the example given in this blog (RobinPkg\Drivers\MyWizardDriver\MyWizardDriver.vfr):

```
form formid = 1, title = STRING_TOKEN(STR_SAMPLE_FORM1_TITLE); // “My Wizard Driver”
  subtitle text = STRING_TOKEN(STR_SUBTITLE_TEXT); //”My Wizard DriverConfiguration”
  subtitle text = STRING_TOKEN(STR_SUBTITLE_TEXT2); //”Device XYZ Configuration”
  checkbox varid =MWD_IfrNVData.MyWizardDriverChooseToEnable,
```

```
    prompt =STRING_TOKEN(STR_CHECK_BOX_PROMPT), //"Enable My XYZ Device"
    help = STRING_TOKEN(STR_CHECK_BOX_HELP), //"This is the help message ...
    flags = CHECKBOX_DEFAULT ,
    key = 0,
    default = 1,
  endcheckbox;
endform;
```

The interface looks like this:

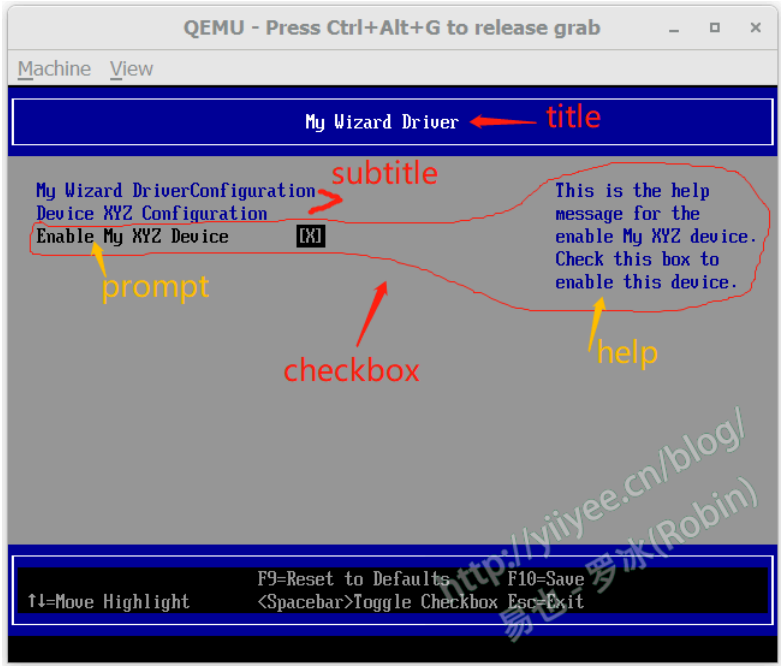

Figure 8 Form defined in VFR file

The content of VFR files is quite extensive, especially its usage, which requires careful study of the Hii part of UEFI spec to get a general understanding. In my daily development, I basically don't use VFR files, and these relatively new knowledge can really make me indulge in it.

In the future, I will also study the usage of VFR files, UNI files and IDF files from time to time, especially the localized display part that I am most concerned about.

In addition, since the name of Baidu Cloud, which originally provided the code, is not very elegant (my daughter pressed it randomly at the time and I didn't pay attention...), I will gradually transfer the code to gitee.

Code for this article:
Gitee address: https://gitee.com/luobing4365/uefi-explorer
Project code is located in: /FF RobinPkg/RobinPkg/Drivers/ MyWizardDriver
Other tools and test environment: /60 VFR file and other Res