UEFI Development Exploration 61-VFR Files and Other Resource Files 2 (Storing Data on NVRAM)

原创 luobing4365 Posted on 2020-07-04 21:15:52 Read 2.9k Collection 15 Likes 4

Category Column: UEFI Development Article Tags: uefi UEFI applicationUEFI principleBIOS BIOS developmentHardware programming UEFI bios uefi driver

(Please keep it-> Author: Luo Bing https://blog.csdn.net/luobing4365)

NVRAM stands for Non-Volatile Ram. Under Legacy BIOS, this is implemented using CMOS, while under UEFI, a part of ROM can be directly allocated to implement it. (The question is, how is CMOS used under UEFI? I will study it again when I have the chance)

Under UEFI, the use of NVRAM is shown in Figure 1.

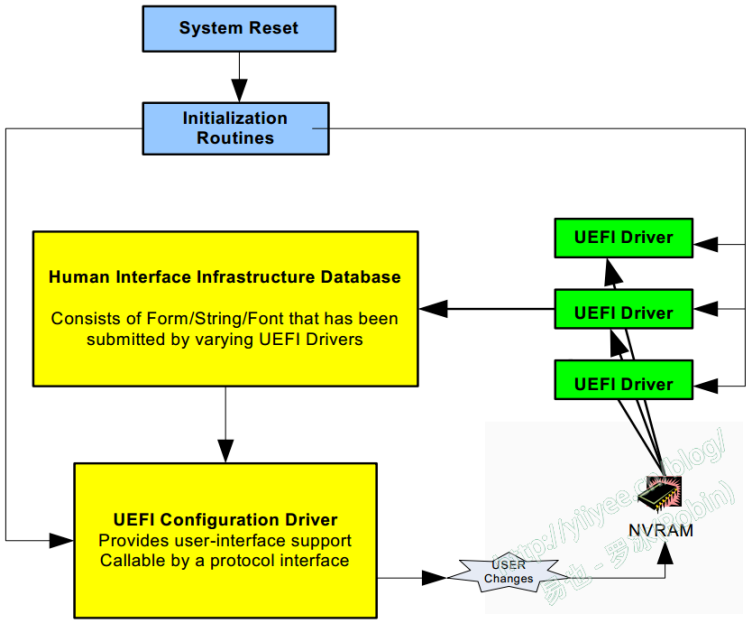

Figure 1 Configuration parameters stored in NVRAM

Based on the code in the previous blog, some modifications are made to store the user's selection in NVRAM. The modification steps are as follows.

1 Update MyWizardDriver.c

Get several HII-related Protocol pointers:

```
EFI_HII_STRING_PROTOCOL *HiiString;
EFI_FORM_BROWSER2_PROTOCOL *FormBrowser2;
EFI_HII_CONFIG_ROUTING_PROTOCOL *HiiConfigRouting;
```

Add the corresponding code (Line228~Line261):

```
// Locate Hii Database protocol
//
Status = gBS->LocateProtocol (&gEfiHiiDatabaseProtocolGuid, NULL, (VOID **) &HiiDatabase);
if (EFI_ERROR (Status)) {    return Status;  }  PrivateData->HiiDatabase = HiiDatabase; … PrivateData->HiiConfigRouting = HiiConfigRouting;
```

In the subsequent code, the code for obtaining the Hii Database protocol is commented out. (Line 289~Line 298 and Line 310)

2 Update HiiConfigAccess.c

Add declarations of external variables (Line14 and Line15):

```
extern EFI_GUID mMyWizardDriverFormSetGuid;
extern CHAR16 mIfrVariableName[];
```

Then transform the three functions, which did not implement specific functions before:

MyWizardDriverHiiConfigAccessExtractConfig(), MyWizardDriverHiiConfigAccessRouteConfig() and MyWizardDriverHiiConfigAccessCallback().

I won't post the specific code here. You can download the code from the address given at the end of the article and view it.

3 Several important Porotocols

(1) gEfiHiiDatabaseProtocolGuid. The GUID of EFI_HII_DATABASE_PROTOCOL, defined in EdkCompatibilityPkg\Foundation\Efi\Protocol\HiiDatabase\HiiDatabase. Protocol for accessing the HII repository. UEFI Spec 2.8 Section 34.8 describes its usage in detail.

(2) gEfiHiiStringProtocolGuid. The GUID of EFI_HII_STRING_PROTOCOL, defined in EdkCompatibilityPkg\Foundation\Efi\Protocol\HiiString\HiiString.c. Protocol for accessing string resources, section 34.3 of UEFI Spec 2.8 describes its usage, and it has been used in previous blogs.

(3) gEfiFormBrowser2ProtocolGuid. The GUID of EFI_FORM_BROWSER2_PROTOCOL, defined in EdkCompatibilityPkg\Foundation\Efi\Protocol\FormBrowser2\FormBrowser2.c, is the basic engine of the Setup interface. Section 35.6 of UEFI Spec 2.8 describes its usage.

(4) gEfiHiiConfigRoutingProtocolGuid. The GUID of EFI_HII_CONFIG_ROUTING_PROTOCOL, defined in EdkCompatibilityPkg\Foundation\Efi\Protocol\HiiConfigRouting\HiiConfigRouting.c. This is a global protocol for handling Setup interface interactions. Section 35.4 of UEFI Spec 2.8 describes its usage.

Variable is a key/value pair in the UEFI environment (reminds me of a tuple in discrete mathematics), used to store data on the platform, allowing it to be used in the EFI execution environment and the EFI OS Loader, and can also be accessed through the API under the operating system. In most cases, variables persist (do not disappear due to power failure). In this article, the relevant protocols for handling variables in Rutime Services are used, so I took the opportunity to sort them out.

```
(5) typedef EFI_STATUS GetVariable (
      IN CHAR16 *VariableName,   //Variable string name
      IN EFI_GUID *VendorGuid,   //Variable unique GUID
      OUT UINT32 *Attributes OPTIONAL, //Attributes. If the definition is not empty, the relevant properties are returned.
                  //See the relevant macro definition in MdePkg\Include\Uefi\UefiMultiPhase.h
      IN OUT UINTN *DataSize, //Specify the length of the input or output data buffer
      OUT VOID *Data OPTIONAL //Data buffer
);
```

Each vendor will create and maintain its own variables. In order to avoid conflicts with other variables, VendorGuid is used to uniquely identify these variables. Among various attributes, EFI_VARIABLE_AUTHENTICATED_WRITE_ACCESS is quite special. The data interface it returns has a standard definition, which can be viewed in the UEFI standard.

```
(6) typedef EFI_STATUS GetNextVariableName (
      IN OUT UINTN *VariableNameSize, //The byte length of VariableName must be large enough to hold the string (including NULL characters)
      IN OUT CHAR16 *VariableName, //When used as input, the last variable string name returned by this function must be provided;  //When used as output, the variable string is returned
      IN OUT EFI_GUID *VendorGuid //When used as input, the last VendorGuid returned by this function must be provided;
                  //When used as output, the GUID of the current variable is returned );
```

In order to get familiar with the usage of this function, I wrote a ListVariable project example myself, referring to the function in ShellProtocol. For details, please see the link at the end of the article.

This is a rough example. The screen display is not considered. There are too many variables. Only the last screen can be displayed:

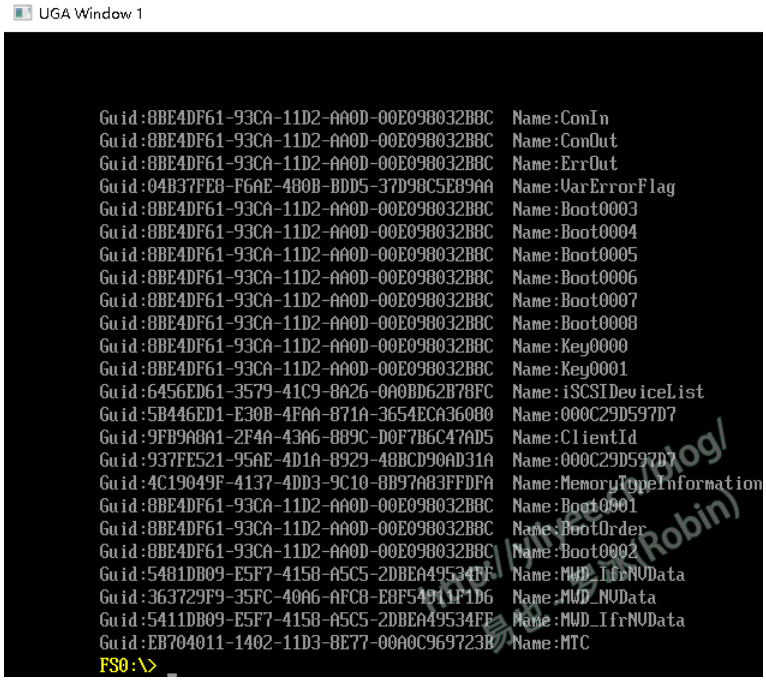

Figure 2 shows Variable

In fact, the Shell command dmpstore can list all variables and their contents, which we will use in the following experiments.

```
(7) typedef EFI_STATUS SetVariable (
      IN CHAR16 *VariableName,  // VariableName's byte length must be large enough to hold the string (including NULL characters)
      IN EFI_GUID *VendorGuid,  //Variable's unique GUID
      IN UINT32 Attributes,     // Attributes, see the description in the GetVariable function
      IN UINTN DataSize,  // Length used as the data buffer. However, when Attributes is the following value, the function is different (see the UEFI specification for details):
                          //EFI_VARIABLE_APPEND_WRITE,
                          //EFI_VARIABLE_AUTHENTICATED_WRITE_ACCESS,
                          //EFI_VARIABLE_ENHANCED_AUTHENTICATED_ACCESS,
                          //EFI_VARIABLE_TIME_BASED_AUTHENTICATED_WRITE_ACCESS
      IN VOID *Data      // Variable's content
);
```

This is a very complex function. The UEFI specification uses 7 pages to introduce it (UEFI Spec 2.8 p235-p241). It is impossible to discuss it in detail here. It is better to learn through examples and understand its usage.

4 Compilation and testing

To demonstrate the use of Variable, I added a few lines of code starting at Line 271 of HiiConfigAccess.c:

```
//robin add 2020-07-01 22:46:51
PrivateData->Configuration.MyWizardDriverStringData[0]=0x31;
PrivateData->Configuration.MyWizardDriverStringData[1]=0x32;
PrivateData->Configuration.MyWizardDriverStringData[2]=0x33;
PrivateData->Configuration.MyWizardDriverStringData[3]=0x34;
PrivateData->Configuration.MyWizardDriverStringData[4]=0x35;
```

Copy the project to the Driver folder of RobinPkg and compile:

```
C:\MyWorkspace>build -a X64 -p RobinPkg\RobinPkg.dsc -m RobinPkg\Drivers\MyWizardDrv01\MyWizardDriver.inf
```

Using the debugging environment in the previous article and the dmpstore tool, the test results are as follows:

Figure 3 Testing process

After loading the driver, enter BIOS Setup, modify the customized options, and then exit. After entering UEFI Shell again, you can check Variable and you can see that the modified part has changed.

Gitee address: https://gitee.com/luobing4365/uefi-explorer
Project code is located in: /FF RobinPkg/RobinPkg/Drivers/ MyWizardDrv01
/FF RobinPkg/RobinPkg/Applications/ListVariable