[UEFI Practice] Modules involved in HII

原创 jiangwei0512 Modified on 2022-02-08 14:55:59 Read 3.4k Collection 6 Likes 1

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

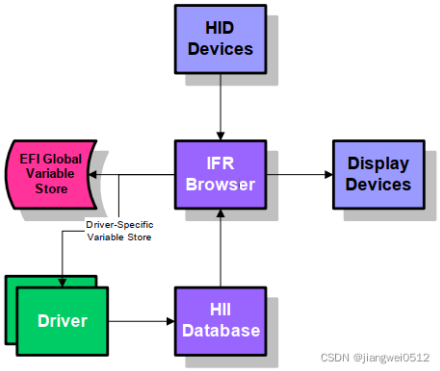
摘要 This article introduces the key modules in the UEFI Human-Machine Interface (HII), including HiiDatabaseDxe.inf, SetupBrowserDxe.inf, and DisplayEngineDxe.inf. These modules are responsible for initializing and managing interface elements, implementing the user interaction engine, and the display engine. By installing and using a series of protocols, such as HiiFont, HiiString, and HiiDatabase, the management and operation of elements such as fonts, strings, and images are realized. At the same time, these modules also involve functions such as the creation of the user interaction interface, event processing, and shortcut …

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

Expand

Modules

The interaction logic of HII is shown in the following figure:



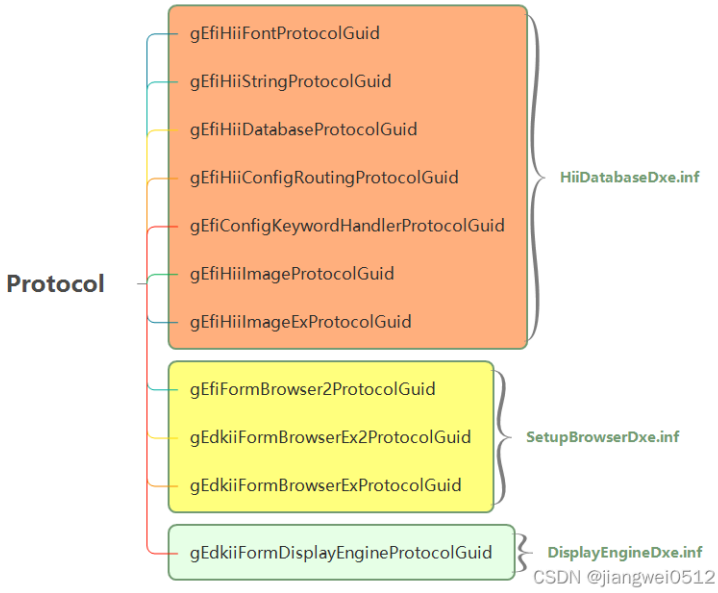
The top part is the input device, such as keyboard, mouse, etc. The right part is the output device, such as monitor, and the left part is the UEFI variable, which is used to store the data involved in HII. These three parts are all UEFI basics and are not the objects introduced here. Here we mainly explain **the IFR Browser** and **HII Database** in the purple part in the middle . They are also important modules that HII needs to pay attention to. The corresponding modules in the code are as follows:

```json
MdeModulePkg/Universal/HiiDatabaseDxe/HiiDatabaseDxe.inf
MdeModulePkg/Universal/SetupBrowserDxe/SetupBrowserDxe.inf
MdeModulePkg/Universal/DisplayEngineDxe/DisplayEngineDxe.inf
```

In addition, other modules (the green part Driver in the above figure) will also provide elements required for the user interaction interface, which is usually to complete the operations required by the module.

Among the three modules above, `HiiDatabaseDxe.inf` the module is used to initialize and install the protocols of the operation interface elements and configure related protocols. These interface elements include fonts, strings, structures, images, etc. `SetupBrowserDxe.inf` This module depends on `HiiDatabaseDxe.inf` , it provides the implementation of the middle part of the above figure, and is actually the engine of the entire user interaction interface, used to implement various necessary operations (in fact, it is not implemented by it, it just calls the operations in the green part Driver). `DisplayEngineDxe.inf` It is `SetupBrowserDxe.inf` used together with, and it depends on the latter.

The protocols for module installation are as follows:



HiiDatabaseDxe.inf

This module initializes `HII_DATABASE_PRIVATE_DATA` (corresponding variables `mPrivate`):

```
InitializeListHead (&mPrivate.DatabaseList);
InitializeListHead (&mPrivate.DatabaseNotifyList);
InitializeListHead (&mPrivate.HiiHandleList);
InitializeListHead (&mPrivate.FontInfoList);
```

Then a bunch of protocols were installed:

```
Status = gBS->InstallMultipleProtocolInterfaces (
                &Handle,
                &gEfiHiiFontProtocolGuid,
                &mPrivate.HiiFont,
                &gEfiHiiStringProtocolGuid,
                &mPrivate.HiiString,
                &gEfiHiiDatabaseProtocolGuid,
                &mPrivate.HiiDatabase,
                &gEfiHiiConfigRoutingProtocolGuid,
                &mPrivate.ConfigRouting,
                &gEfiConfigKeywordHandlerProtocolGuid,
                &mPrivate.ConfigKeywordHandler,
                NULL
                );
```

收起 ^

There is another part that is optional:

```
if (FeaturePcdGet (PcdSupportHiiImageProtocol)) {
  Status = gBS->InstallMultipleProtocolInterfaces (
                  &Handle,
                  &gEfiHiiImageProtocolGuid, &mPrivate.HiiImage,
                  &gEfiHiiImageExProtocolGuid, &mPrivate.HiiImageEx,
                  NULL
                  );

}
```

The protocols installed will be introduced later. These protocols are installed into `HII_DATABASE_PRIVATE_DATA` the structure mentioned above:

```
typedef struct _HII_DATABASE_PRIVATE_DATA {
  UINTN                               Signature;
  LIST_ENTRY                          DatabaseList;
  LIST_ENTRY                          DatabaseNotifyList;
  EFI_HII_FONT_PROTOCOL               HiiFont;
  EFI_HII_IMAGE_PROTOCOL              HiiImage;
  EFI_HII_IMAGE_EX_PROTOCOL           HiiImageEx;
  EFI_HII_STRING_PROTOCOL             HiiString;
  EFI_HII_DATABASE_PROTOCOL           HiiDatabase;
  EFI_HII_CONFIG_ROUTING_PROTOCOL     ConfigRouting;
  EFI_CONFIG_KEYWORD_HANDLER_PROTOCOL ConfigKeywordHandler;
  LIST_ENTRY                          HiiHandleList;
  INTN                                HiiHandleCount;
  LIST_ENTRY                          FontInfoList;  // global font info list
  UINTN                               Attribute;     // default system color
  EFI_GUID                            CurrentLayoutGuid;
  EFI_HII_KEYBOARD_LAYOUT             *CurrentLayout;
} HII_DATABASE_PRIVATE_DATA;

```

收起 ^

These protocols are used to operate fonts, strings, images and other elements in the UEFI interactive interface.

SetupBrowserDxe.inf

This module depends on `HiiDatabaseDxe.inf` some protocols installed in , so existing `LocateProtocol()` operations:

```
//
// Locate required Hii relative protocols
//
Status = gBS->LocateProtocol (
                &gEfiHiiDatabaseProtocolGuid,
                NULL,
                (VOID **) &mHiiDatabase
                );
ASSERT_EFI_ERROR (Status);

Status = gBS->LocateProtocol (
                &gEfiHiiConfigRoutingProtocolGuid,
                NULL,
                (VOID **) &mHiiConfigRouting
                );
ASSERT_EFI_ERROR (Status);

Status = gBS->LocateProtocol (
                &gEfiDevicePathFromTextProtocolGuid,
                NULL,
                (VOID **) &mPathFromText
                );
```

收起 ^

Then install the Protocol:

```c
  //
  // Install FormBrowser2 protocol
  //
  mPrivateData.Handle = NULL;
  Status = gBS->InstallProtocolInterface (
                  &mPrivateData.Handle,
                  &gEfiFormBrowser2ProtocolGuid,
                  EFI_NATIVE_INTERFACE,
                  &mPrivateData.FormBrowser2
                  );
  ASSERT_EFI_ERROR (Status);

  Status = gBS->InstallProtocolInterface (
                  &mPrivateData.Handle,
                  &gEdkiiFormBrowserEx2ProtocolGuid,
                  EFI_NATIVE_INTERFACE,
                  &mPrivateData.FormBrowserEx2
                  );
  ASSERT_EFI_ERROR (Status);

  Status = gBS->InstallProtocolInterface (
                  &mPrivateData.Handle,
                  &gEdkiiFormBrowserExProtocolGuid,
                  EFI_NATIVE_INTERFACE,
                  &mPrivateData.FormBrowserEx
                  );
  ASSERT_EFI_ERROR (Status);
```

收起 ^

Then initialize `SETUP_DRIVER_PRIVATE_DATA` (corresponding variables `mPrivateData`):

```c
  InitializeListHead (&mPrivateData.FormBrowserEx2.FormViewHistoryHead);
  InitializeListHead (&mPrivateData.FormBrowserEx2.OverrideQestListHead);
```

There is also initialization `EDKII_FORM_DISPLAY_ENGINE_PROTOCOL` (correspondingly `mFormDisplay`, this Protocol will actually be `DisplayEngineDxe.inf` installed in the entire module, so the Callback method is used):

```c
  Status = gBS->LocateProtocol (
                  &gEdkiiFormDisplayEngineProtocolGuid,
                  NULL,
                  (VOID **) &mFormDisplay
                  );

  if (EFI_ERROR (Status)) {
    EfiCreateProtocolNotifyEvent (
      &gEdkiiFormDisplayEngineProtocolGuid,
      TPL_CALLBACK,
      FormDisplayCallback,
      NULL,
      &Registration
      );
  }
```

收起 ^

And `FORM_DISPLAY_ENGINE_FORM` (corresponding `gDisplayFormData`):

```c
/**
  Initialize the Display form structure data.
**/
VOID
InitializeDisplayFormData (
  VOID
  )
{
  EFI_STATUS  Status;

  gDisplayFormData.Signature   = FORM_DISPLAY_ENGINE_FORM_SIGNATURE;
  gDisplayFormData.Version     = FORM_DISPLAY_ENGINE_VERSION_1;
  gDisplayFormData.ImageId     = 0;
  gDisplayFormData.AnimationId = 0;

  InitializeListHead (&gDisplayFormData.StatementListHead);
  InitializeListHead (&gDisplayFormData.StatementListOSF);
  InitializeListHead (&gDisplayFormData.HotKeyListHead);

  Status = gBS->CreateEvent (
        EVT_NOTIFY_WAIT,
        TPL_CALLBACK,
        EfiEventEmptyFunction,
        NULL,
        &mValueChangedEvent
        );
  ASSERT_EFI_ERROR (Status);
}
```

收起 ^

DisplayEngineDxe.inf

The first step is to install a Strings:

```c
  gHiiHandle = HiiAddPackages (
                 &gDisplayEngineGuid,
                 ImageHandle,
                 DisplayEngineStrings,
                 NULL
                 );
  ASSERT (gHiiHandle != NULL);
```

Then install the Protocol:

```c
  //
  // Install Form Display protocol
  //
  Status = gBS->InstallProtocolInterface (
                  &mPrivateData.Handle,
                  &gEdkiiFormDisplayEngineProtocolGuid,
                  EFI_NATIVE_INTERFACE,
                  &mPrivateData.FromDisplayProt
                  );
  ASSERT_EFI_ERROR (Status);
```

收起 ^

Then initialize a bunch of strings:

```c
InitializeDisplayStrings();

/**
  Initialize the HII String Token to the correct values.
**/
VOID
InitializeDisplayStrings (
  VOID
  )
{
  gReconnectConfirmChanges = GetToken (STRING_TOKEN (RECONNECT_CONFIRM_CHANGES), gHiiHandle);
  mUnknownString        = GetToken (STRING_TOKEN (UNKNOWN_STRING), gHiiHandle);
  gSaveFailed           = GetToken (STRING_TOKEN (SAVE_FAILED), gHiiHandle);
  gNoSubmitIfFailed     = GetToken (STRING_TOKEN (NO_SUBMIT_IF_CHECK_FAILED), gHiiHandle);
  gReconnectFail        = GetToken (STRING_TOKEN (RECONNECT_FAILED), gHiiHandle);
  gReconnectRequired    = GetToken (STRING_TOKEN (RECONNECT_REQUIRED), gHiiHandle);
  gChangesOpt           = GetToken (STRING_TOKEN (RECONNECT_CHANGES_OPTIONS), gHiiHandle);
  gSaveProcess          = GetToken (STRING_TOKEN (DISCARD_OR_JUMP), gHiiHandle);
  gSaveNoSubmitProcess  = GetToken (STRING_TOKEN (DISCARD_OR_CHECK), gHiiHandle);
  gDiscardChange        = GetToken (STRING_TOKEN (DISCARD_OR_JUMP_DISCARD), gHiiHandle);
twen gJumpToFormSet     = GetToken (STRING_TOKEN (DISCARD_OR_JUMP_JUMP), gHiiHandle);
twen gCheckError        = GetToken (STRING_TOKEN (DISCARD_OR_CHECK_CHECK), gHiiHandle);
twen gPromptForData     = GetToken (STRING_TOKEN (PROMPT_FOR_DATA), gHiiHandle);
twen gPromptForPassword = GetToken (STRING_TOKEN (PROMPT_FOR_PASSWORD), gHiiHandle);
  gPromptForNewPassword = GetToken (STRING_TOKEN (PROMPT_FOR_NEW_PASSWORD), gHiiHandle);
  gConfirmPassword      = GetToken (STRING_TOKEN (CONFIRM_PASSWORD), gHiiHandle);
  gConfirmError         = GetToken (STRING_TOKEN (CONFIRM_ERROR), gHiiHandle);
  gPassowordInvalid     = GetToken (STRING_TOKEN (PASSWORD_INVALID), gHiiHandle);
  gPressEnter           = GetToken (STRING_TOKEN (PRESS_ENTER), gHiiHandle);
  gEmptyString          = GetToken (STRING_TOKEN (EMPTY_STRING), gHiiHandle);
  gMiniString           = GetToken (STRING_TOKEN (MINI_STRING), gHiiHandle);
  gOptionMismatch       = GetToken (STRING_TOKEN (OPTION_MISMATCH), gHiiHandle);
  gFormSuppress         = GetToken (STRING_TOKEN (FORM_SUPPRESSED), gHiiHandle);
  gProtocolNotFound     = GetToken (STRING_TOKEN (PROTOCOL_NOT_FOUND), gHiiHandle);
  gFormNotFound         = GetToken (STRING_TOKEN (STATUS_BROWSER_FORM_NOT_FOUND), gHiiHandle);
  gNoSubmitIf           = GetToken (STRING_TOKEN (STATUS_BROWSER_NO_SUBMIT_IF), gHiiHandle);
  gBrowserError         = GetToken (STRING_TOKEN (STATUS_BROWSER_ERROR), gHiiHandle);
  gConfirmDefaultMsg    = GetToken (STRING_TOKEN (CONFIRM_DEFAULT_MESSAGE), gHiiHandle);
  gConfirmDiscardMsg    = GetToken (STRING_TOKEN (CONFIRM_DISCARD_MESSAGE), gHiiHandle);
  gConfirmSubmitMsg     = GetToken (STRING_TOKEN (CONFIRM_SUBMIT_MESSAGE), gHiiHandle);
  gConfirmResetMsg      = GetToken (STRING_TOKEN (CONFIRM_RESET_MESSAGE), gHiiHandle);
  gConfirmExitMsg       = GetToken (STRING_TOKEN (CONFIRM_EXIT_MESSAGE), gHiiHandle);
  gConfirmDefaultMsg2nd = GetToken (STRING_TOKEN (CONFIRM_DEFAULT_MESSAGE_2ND), gHiiHandle);
  gConfirmSubmitMsg2nd  = GetToken (STRING_TOKEN (CONFIRM_SUBMIT_MESSAGE_2ND), gHiiHandle);
  gConfirmResetMsg2nd   = GetToken (STRING_TOKEN (CONFIRM_RESET_MESSAGE_2ND), gHiiHandle);
  gConfirmExitMsg2nd    = GetToken (STRING_TOKEN (CONFIRM_EXIT_MESSAGE_2ND), gHiiHandle);
  gConfirmOpt           = GetToken (STRING_TOKEN (CONFIRM_OPTION), gHiiHandle);
  gConfirmOptYes        = GetToken (STRING_TOKEN (CONFIRM_OPTION_YES), gHiiHandle);
  gConfirmOptNo         = GetToken (STRING_TOKEN (CONFIRM_OPTION_NO), gHiiHandle);
  gConfirmMsgConnect    = GetToken (STRING_TOKEN (CONFIRM_OPTION_CONNECT), gHiiHandle);
  gConfirmMsgEnd        = GetToken (STRING_TOKEN (CONFIRM_OPTION_END), gHiiHandle);
  gPasswordUnsupported  = GetToken (STRING_TOKEN (PASSWORD_NOT_SUPPORTED ), gHiiHandle);
}
```

收起 ^

Then initialize several variables:

```c
  ZeroMem (&gHighligthMenuInfo, sizeof (gHighligthMenuInfo));
  ZeroMem (&gOldFormEntry, sizeof (gOldFormEntry));
```

Finally, register a few shortcut keys:

```c
//
// Use BrowserEx2 protocol to register HotKey.
//
Status = gBS->LocateProtocol (&gEdkiiFormBrowserEx2ProtocolGuid, NULL, (VOID **) &FormBrowserEx2);
if (!EFI_ERROR (Status)) {
  //
  // Register the default HotKey F9 and F10 again.
  //
  HotKey.UnicodeChar = CHAR_NULL;
  HotKey.ScanCode    = SCAN_F10;
  NewString          = HiiGetString (gHiiHandle, STRING_TOKEN (FUNCTION_TEN_STRING), NULL);
  ASSERT (NewString != NULL);
  FormBrowserEx2->RegisterHotKey (&HotKey, BROWSER_ACTION_SUBMIT, 0, NewString);

  HotKey.ScanCode    = SCAN_F9;
  NewString          = HiiGetString (gHiiHandle, STRING_TOKEN (FUNCTION_NINE_STRING), NULL);
  ASSERT (NewString != NULL);
  FormBrowserEx2->RegisterHotKey (&HotKey, BROWSER_ACTION_DEFAULT, EFI_HII_DEFAULT_CLASS_STANDARD, NewString);
}
```

收起 ^