UEFI Development Exploration 16 – Using HII to display Chinese characters 1

原创 luobing4365 Posted on 2019-09-15 11:32:31 Read 2.5k Collection 7 Likes 2 copyright
Category Column: UEFI Development Article Tags: UEFI Programming UEFI HII UEFI Chinese character programming Low-level programming HII protocol

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

In my last blog, I used my own method to display Chinese characters. The core idea is to use the implemented dot drawing function to draw the Chinese characters pixel by pixel.

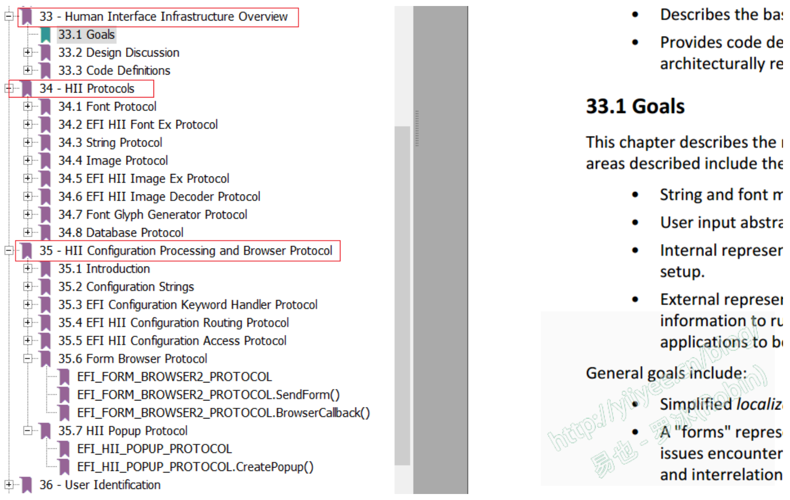
Obviously, this method can only be implemented in graphics mode. If you want to display in character mode (such as UEFI Shell), you can't do it. Although the current commercial Option Rom cannot use character mode, the programmer's nature of seeking the truth has driven me to understand the Human Interface Infrastructure mechanism provided by UEFI itself.

So began my journey of exploration over the past few days.

All explorations start with questions. For HII, the questions I asked are as follows:

1. **How to organize Chinese character libraries (the same applies to other languages, how to store character libraries?);**
2. **Can Chinese characters be displayed in UEFI Shell? How to achieve it?**
3. **How to display Chinese characters in graphic mode?**

The materials I consulted include UEFI Spec, "UEFI Principles and Programming" and "Harnessing the uefi shell", as well as some code I wrote before.


33 - Human Interface Infrastructure Overview
33.1 Goals
33.2 Design Discussion
33.3 Code Definitions
34 - HII Protocols
34.1 Font Protocol
34.2 EFI HII Font Ex Protocol
34.3 String Protocol
34.4 Image Protocol
34.5 EFI HII Image Ex Protocol
34.6 EFI HII Image Decoder Protocol
34.7 Font Glyph Generator Protocol
34.8 Database Protocol
35 - HII Configuration Processing and Browser Protocol
35.1 Introduction
35.2 Configuration Strings
35.3 EFI Configuration Keyword Handler Protocol
35.4 EFI HII Configuration Routing Protocol
35.5 EFI HII Configuration Access Protocol
35.6 Form Browser Protocol
EFI_FORM_BROWSER2_PROTOCOL
EFI_FORM_BROWSER2_PROTOCOL.SendForm()
EFI_FORM_BROWSER2_PROTOCOL.BrowserCallback()
35.7 HII Popup Protocol
EFI_HII_POPUP_PROTOCOL
EFI_HII_POPUP_PROTOCOL.CreatePopup()
36 - User Identification

- Describes the ba
- Provides code de architecturally re

33.1 Goals

This chapter describes the r areas described include the

- String and font m
- User input abstra
- Internal represer setup.
- External represer information to ru applications to b

General goals include:

- Simplified *localiz*
- A "forms" repres issues encounter and interrelation

Figure 1 HII in the UEFI spec2.8

UEFI Spec uses three chapters to introduce HII, and the content is quite rich. It takes a lot of time to really understand it all. I will start with a simpler one: how are strings in various languages stored? This is also the first thing introduced in Chapter 11 of "UEFI Principles and Programming".

I am going to transplant the routines in the book into my program. The template program I use is 04 ReadKey, which was introduced in the previous blog.

1 Organization of string resources

String resources are organized in *.uni files, and use identifiers such as #langdef and #string to define fields. Take the example.uni provided in this blog (from chap 11 of UEFI Principles and Programming) as an example:

#langdef en-US “English
” #langdef zh-Hans “Simplified Chinese” … #string STR_LANGUAGE_SELECT #language en-US
“ Select Language”
#language zh
-Hans “Select Language”
…

Among them, **#langdef** is used to declare the language supported by this string resource file. The first parameter is the character identifier; the second parameter is the displayable name string of the language.

I am very curious. Taking the display of Chinese as an example, although the name string in the definition is "Simplified Chinese", what will happen if I want to print it under the shell?

I guess nothing will be displayed. This is back to the question of how to display Chinese characters in UEFI Shell. Without providing a font library, nothing can be displayed.

#string is used to define a string. The first parameter is a string identifier, which is equivalent to a variable definition; the second parameter is a string in the language.

The character package combination generated after compiling example.uni will be defined in the generated AutoGen.c . If you study this file carefully, you will understand a lot.

It should be noted that the strings defined in example.uni are not generated as exampleStrings in this example after compilation, but as Luo2Strings. I guess the compiler adds "Strings" to the BASE_NAME in *.inf (BASE_NAME in Luo.inf is Luo2), but I haven't found any evidence to prove it, so I guess this is the case.

2 Strings in AutoGen.c

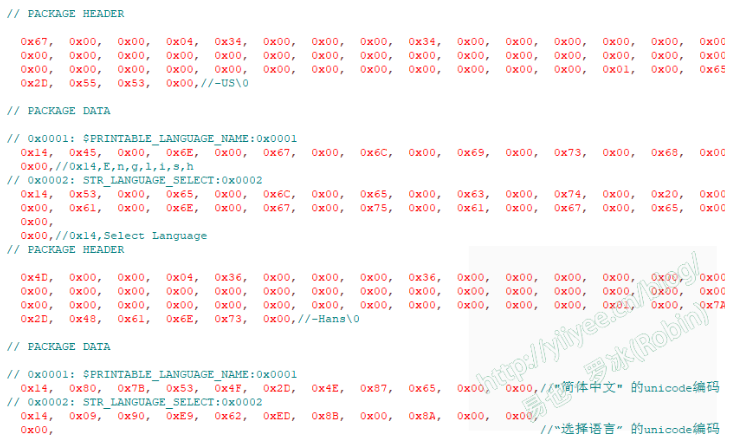
The string after *.uni compilation can be found in AutoGen.c.

At first, I thought that Chinese strings were still stored in the form of national standard codes. The national standard codes of Chinese characters can be easily found, and many editors provide binary viewing functions, which can be found directly.

I read the string part of AutoGen.c carefully. The first 4 bytes of Pack are the length and the rest is the string, but I can't match the meaning of the Chinese characters one by one.

Suddenly I realized that strings should be stored in Unicode. The code defines strings in the form of L"xx", which already explains this. Why didn't I think of it earlier?

I found the UCS-2 character table and checked them one by one, and sure enough, they were correct.



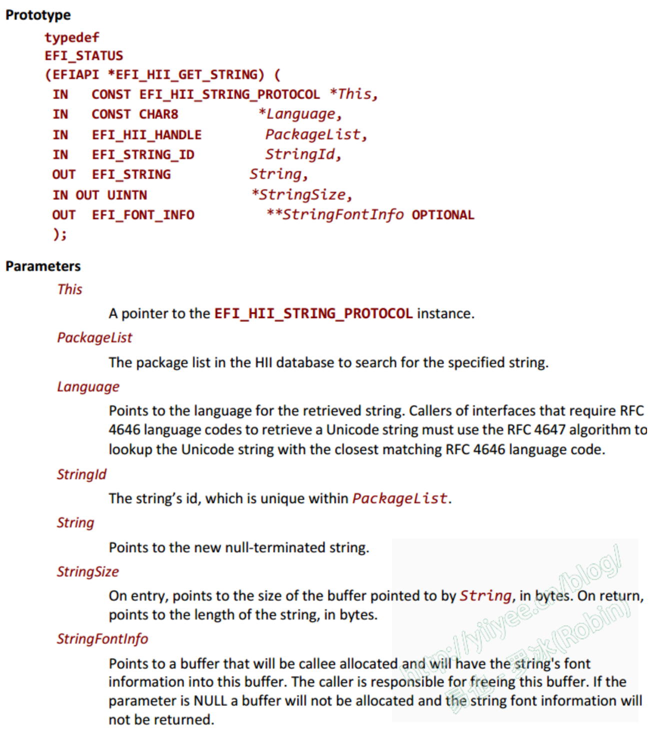
```
// PACKAGE HEADER

  0x67,  0x00,  0x00,  0x04,  0x34,  0x00,  0x00,  0x00,  0x34,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00
  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00
  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x01,  0x00,  0x65
  0x2D,  0x55,  0x53,  0x00,//-US\0

// PACKAGE DATA

// 0x0001: $PRINTABLE_LANGUAGE_NAME:0x0001
  0x14,  0x45,  0x00,  0x6E,  0x00,  0x67,  0x00,  0x6C,  0x00,  0x69,  0x00,  0x73,  0x00,  0x68,  0x00
  0x00,//0x14,E,n,g,l,i,s,h
// 0x0002: STR_LANGUAGE_SELECT:0x0002
  0x14,  0x53,  0x00,  0x65,  0x00,  0x6C,  0x00,  0x65,  0x00,  0x63,  0x00,  0x74,  0x00,  0x20,  0x00
  0x00,  0x61,  0x00,  0x6E,  0x00,  0x67,  0x00,  0x75,  0x00,  0x61,  0x00,  0x67,  0x00,  0x65,  0x00
  0x00,
  0x00,//0x14,Select Language
// PACKAGE HEADER

  0x4D,  0x00,  0x00,  0x04,  0x36,  0x00,  0x00,  0x00,  0x36,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00
  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00
  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x00,  0x01,  0x00,  0x7A
  0x2D,  0x48,  0x61,  0x6E,  0x73,  0x00,//-Hans\0

// PACKAGE DATA

// 0x0001: $PRINTABLE_LANGUAGE_NAME:0x0001
  0x14,  0x80,  0x7B,  0x53,  0x4F,  0x2D,  0x4E,  0x87,  0x65,  0x00,  0x00,//"简体中文" 的unicode编码
// 0x0002: STR_LANGUAGE_SELECT:0x0002
  0x14,  0x09,  0x90,  0xE9,  0x62,  0xED,  0x8B,  0x00,  0x8A,  0x00,  0x00,
  0x00,                                                              //"选择语言" 的unicode编码
```

Figure 2 Organization of strings in AutoGen.c

3Managing string resources

UEFI provides EFI_HII_STRING_PROTOCOL to manage string resources. For specific usage, please refer to the reference book mentioned above, which is very detailed.

I mainly used GetString to get the string and print it out.



Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_HII_GET_STRING) (
 IN   CONST EFI_HII_STRING_PROTOCOL *This,
 IN   CONST CHAR8          *Language,
 IN   EFI_HII_HANDLE        PackageList,
 IN   EFI_STRING_ID         StringId,
 OUT  EFI_STRING           String,
 IN OUT UINTN              *StringSize,
 OUT  EFI_FONT_INFO         **StringFontInfo OPTIONAL
 );
```

Parameters

This

A pointer to the **EFI_HII_STRING_PROTOCOL** instance.

PackageList

The package list in the HII database to search for the specified string.

Language

Points to the language for the retrieved string. Callers of interfaces that require RFC 4646 language codes to retrieve a Unicode string must use the RFC 4647 algorithm to lookup the Unicode string with the closest matching RFC 4646 language code.

StringId

The string's id, which is unique within *PackageList*.

String

Points to the new null-terminated string.

StringSize

On entry, points to the size of the buffer pointed to by *String*, in bytes. On return, points to the length of the string, in bytes.

StringFontInfo

Points to a buffer that will be callee allocated and will have the string's font information into this buffer. The caller is responsible for freeing this buffer. If the parameter is NULL a buffer will not be allocated and the string font information will not be returned.

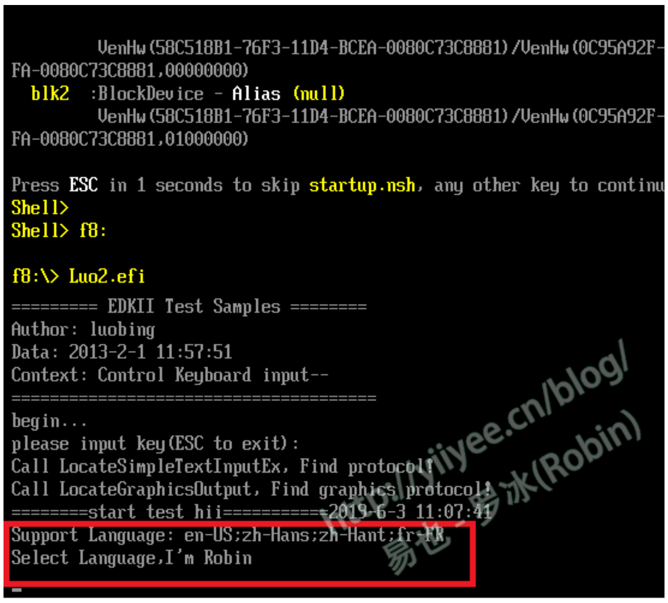
Figure 3 GetString function description

There are two main parameters to focus on: Language, which specifies the language type to be displayed; and String, which is equivalent to the name of a string variable.

In the examples provided in this blog, we mainly focus on TestString and TestLanguage, and use HiiAddPackages to add strings to the database. Other functions have nothing to do with Hii.

4. Compile test

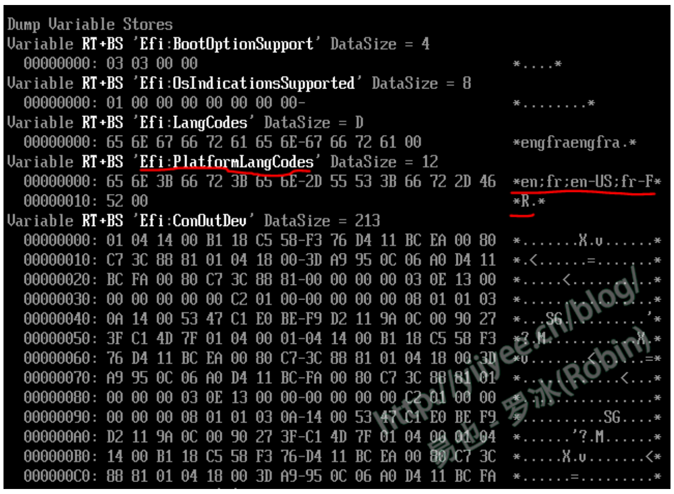
After compiling the program, the effect achieved is as follows:



```
        VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A92F-
FA-0080C73C8881,00000000)
  blk2  :BlockDevice - Alias (null)
        VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A92F-
FA-0080C73C8881,01000000)

Press ESC in 1 seconds to skip startup.nsh, any other key to continu
Shell>
Shell> f8:

f8:\> Luo2.efi
======== EDKII Test Samples ========
Author: luobing
Data: 2013-2-1 11:57:51
Context: Control Keyboard input--
=====================================
begin...
please input key(ESC to exit):
Call LocateSimpleTextInputEx, Find protocol!
Call LocateGraphicsOutput, Find graphics protocol!
=======start test hii==========2019-6-3 11:07:41
Support Language: en-US;zh-Hans;zh-Hant;fr-FR
Select Language,I'm Robin
_
```

Figure 4 Running the program in the TianoCore simulation environment

It is worth noting that using the Shell command ***dmpstore -b*** , the information obtained is as follows:



```
Dump Variable Stores
Variable RT+BS 'Efi:BootOptionSupport' DataSize = 4
  00000000: 03 03 00 00                                      *....*
Variable RT+BS 'Efi:OsIndicationsSupported' DataSize = 8
  00000000: 01 00 00 00 00 00 00 00-                         *........*
Variable RT+BS 'Efi:LangCodes' DataSize = D
  00000000: 65 6E 67 66 72 61 65 6E-67 66 72 61 00           *engfraengfra.*
Variable RT+BS 'Efi:PlatformLangCodes' DataSize = 12
  00000000: 65 6E 3B 66 72 3B 65 6E-2D 55 53 3B 66 72 2D 46  *en;fr;en-US;fr-F*
  00000010: 52 00                                            *R.*
Variable RT+BS 'Efi:ConOutDev' DataSize = 213
  00000000: 01 04 14 00 B1 18 C5 58-F3 76 D4 11 BC EA 00 80  *.......X.v......*
  00000010: C7 3C 88 81 01 04 18 00-3D A9 95 0C 06 A0 D4 11  *.<......=.......*
  00000020: BC FA 00 80 C7 3C 88 81-00 00 00 00 03 0E 13 00  *.....<..........*
  00000030: 00 00 00 00 00 C2 01 00-00 00 00 00 08 01 01 03  *................*
  00000040: 0A 14 00 53 47 C1 E0 BE-F9 D2 11 9A 0C 00 90 27  *...SG..........'*
  00000050: 3F C1 4D 7F 01 04 00 01-04 14 00 B1 18 C5 58 F3  *?.M...........X.*
  00000060: 76 D4 11 BC EA 00 80 C7-3C 88 81 01 04 18 00 3D  *v.......<......=*
  00000070: A9 95 0C 06 A0 D4 11 BC-FA 00 80 C7 3C 88 81 01  *............<...*
  00000080: 00 00 00 03 0E 13 00 00-00 00 00 00 C2 01 00 00  *................*
  00000090: 00 00 00 08 01 01 03 0A-14 00 53 47 C1 E0 BE F9  *..........SG....*
  000000A0: D2 11 9A 0C 00 90 27 3F-C1 4D 7F 01 04 00 01 04  *......'?.M......*
  000000B0: 14 00 B1 18 C5 58 F3 76-D4 11 BC EA 00 80 C7 3C  *.....X.v.......<*
  000000C0: 88 81 01 04 18 00 3D A9-95 0C 06 A0 D4 11 BC FA  *......=.........*
```

Figure 5 Languages supported in the TianoCore simulation environment

The global variable L "PlatformLangCodes" stores all the languages supported by the system, only English and French.

It seems that the language supported in the program is irrelevant to the system variables. The program is self-consistent and can use the language supported by the system itself. If you want to display a language that the system does not support, you have to implement it yourself.

Gitee address: https://gitee.com/luobing4365/uefi-explorer
The project code is located under: /09 HiiShellPrint.