UEFI Development Exploration 17 – Using HII to display Chinese characters 2

原创 luobing4365 Posted on 2019-09-15 12:52:24 Read 1.3k collect Likes 1 copyright

Category Column: UEFI Development Article Tags: UEFI Programming UEFI SimpleFont UEFI Chinese character programming Low-level programming UEFI HII

UEFI Development This column includes this content 503 Subscribe 104 articles Subscribe to our column

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

Keywords for this blog: **SimpleFont.**

After my last programming, I have been thinking about how to display Chinese characters in UEFI Shell. In the previous article, I referred to the examples in the book and basically understood how string resources are organized. When I changed the parameter Language to "zh-Hans" in the TestString function , nothing was displayed.

It seems that there is still a lack of corresponding fonts.

UEFI provides SimpleFont and Font fonts. The former is relatively simple, so I will start with this. The example I refer to is \GUIbasics\font\SimpleFont in "UEFI Principles and Programming". This example did not compile successfully. I did not find out the reason, but still used the old method to transplant its routine into my own program.

1 SimpleFont format

SimpleFont has two font formats, narrow and wide, which are described clearly in the book. The TianoCore simulation environment also uses SimpleFont's font library.

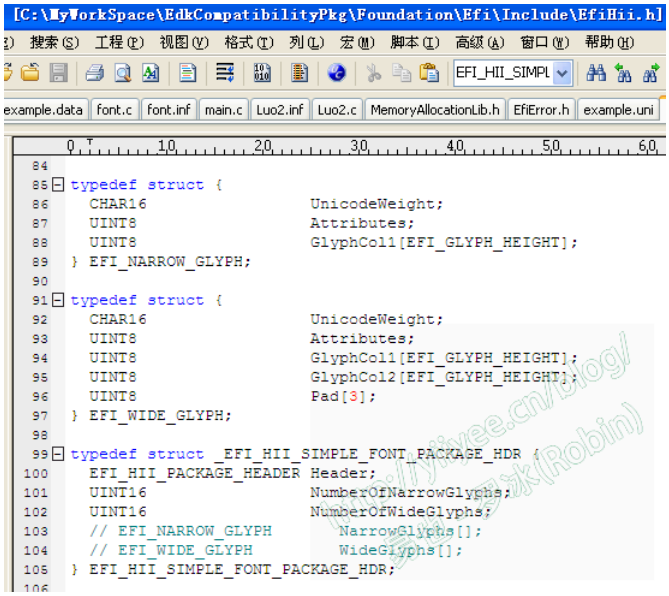

Figure 1 SimpleFont structure, taken from EfiHii.h

The character data in the SimpleFont package is attached to the structure in the form of an array, which can be easily understood by looking at the code:

```
{
    UINT32    packageLen = sizeof (EFI_HII_SIMPLE_FONT_PACKAGE_HDR) + SizeInBytes + 4;
    Package = (UINT8*)AllocateZeroPool (packageLen);

    WriteUnaligned32((UINT32 *) Package,packageLen);
    simpleFont = (EFI_HII_SIMPLE_FONT_PACKAGE_HDR *) (Package + 4);
    simpleFont->Header.Length = (UINT32) (packageLen - 4);
    simpleFont->Header.Type = EFI_HII_PACKAGE_SIMPLE_FONTS;
    simpleFont->NumberOfNarrowGlyphs = 0;
    simpleFont->NumberOfWideGlyphs = (UINT16) (SizeInBytes / sizeof (EFI_WIDE_GLYPH));
}
{
    UINT8 * Location = (UINT8 *) (&simpleFont->NumberOfWideGlyphs + 1);
    CopyMem (Location, WideGlyph, SizeInBytes);
}
```

Figure 2 Initialization of the SimpleFont package (from this blog code Luo2.c Line697–Line710)

2 Generate font files

The generation of font files generally requires writing a small tool to extract them. For example, my previous Chinese character display method requires extracting the required fonts from the font library one by one to form a structure file that meets my requirements.

The book provides a method to extract font data using JavaScript , which is very convenient. You can read createdata.html in the code folder of this blog (**special statement: this file is from "UEFI Principles and Programming", the copyright belongs to the original author**), render the characters into

canvas, analyze the canvas, and generate data in EFI_WIDE_GLYPH format. The processing flow is:

1. *Create a canvas and obtain the canvas context;*
2. *Process each character of Unicode16 encoding from 0x4E00 to 0x9FA5 in turn;*
3. *Write characters to the context area;*
4. *Take the 16×19 bitmap from the context area;*
5. *Convert the bitmap to EFI_WIDE_GLYPH format.*

I am not familiar with JavaScript. From the information I found online, canvas is only supported by Html5. I succeeded in running it in Chrome 74.0.3729.169 (official version) on Windows 10. I failed in IE on Windows XP before. It seems that the version is too old and does not support Html5.

The generated font file is more than 4M, including all Chinese characters. In actual use, it is unlikely that so many Chinese characters will be needed. For example, in the development of Oprom, the ROM space is only a few dozen K, and it is impossible to fit them in anyway.

The best way is to write a program to analyze the *.uni file, see which Chinese characters are needed, then extract the corresponding fonts, and generate the code files required for the project.

However, the code is currently running under UEFI Shell, so program space is not a problem and there is no need to do this. You can write it when you have time.

3Register font files

After the preparation is completed, the last step is to register the font file in the program. Similar to registering a string in the previous blog, use HiiAddPackages() to register the font package in the HII database.

This step is relatively simple, and for me it was just a matter of porting CreatesimpleFontPkg() from the book into my code.

There are several places that need to be modified:

1. Add the header file <Library/MemoryAllocationLib.h>, in which AllocateZeroPool() and FreePool() are used;
2. The return -1 in the function is changed to return EFI_NOT_FOUND. The function return value is EFI_STATUS, which should be UINT type. The author used -1 to return. I don't know why. Return -1 cannot be compiled in my program.

4 Compile and test

Change the Language parameter in TestString() to "zh-Hans" to test the Chinese character display under UEFI Shell. Compile and run:

```
_STATUS TestString(EFI_HANDLE HiiHandle )

FI_STATUS Status = 0;
/CHAR8* BestLanguage = "en-US";
HAR8* BestLanguage = "zh-Hans";
```

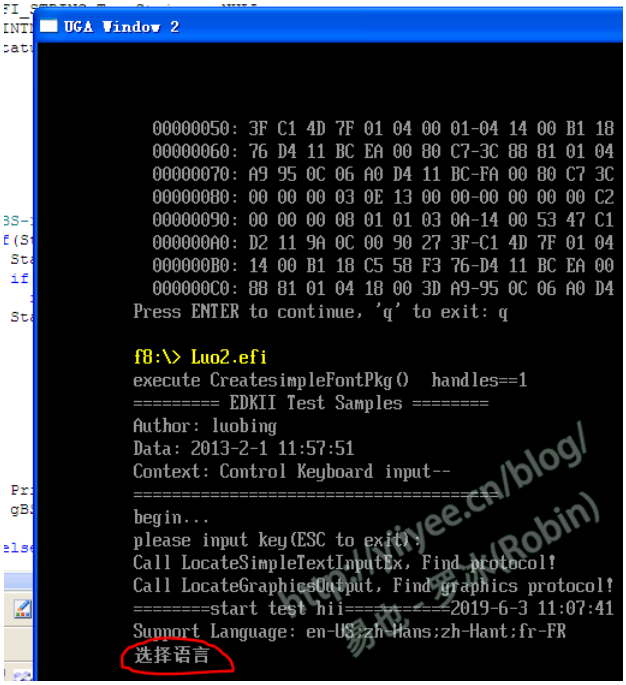

Figure 3 The running results of the program in the TianoCore simulation environment

Success! Thumbs up!

In other words, the operating logic previously guessed is correct. After the font file is registered, the string is encoded with Unicode and the font is searched in the Hii database. Once it is found, it can be displayed.

A new question arises: Is there a way to display colored text in UEFI Shell? (The drive letter shown in the picture is bright yellow)

Gitee address: https://gitee.com/luobing4365/uefi-explorer

The project code is located under: /10 HiiShellPrint-Chinese.