UEFI Development Exploration 51 – Printing Function under UEFI

原创 luobing4365 Posted on 2020-04-01 15:30:56 Read 5.3k Collection 23 Likes 8 copyright

Category Column: UEFI Development Article Tags: UEFI Programming Low-level programming UEFI Printing Low-level application development

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

The 50 blogs I originally planned have finally been completed. Unfortunately, the current length cannot cover all the content I originally envisioned. Therefore, the development and exploration series will continue to be written. Since the original goal of 50 blogs has been achieved, a new target number will not be set. I will only continue to explore the various aspects that I am interested in.

In daily development, we always use various print functions . The format of the print function under UEFI is a bit strange, which is different from the print function under Windows and Linux. In addition, it involves the support of Ascii characters and Unicode characters. It is very annoying to make some small mistakes in use.

Therefore, in order to facilitate subsequent development, I plan to sort out the relevant printing functions for reference.

1OutputString()

The most basic print output function, other Print functions are built based on this function. Its function prototype is:

```
typedef
EFI_STATUS
(EFIAPI *EFI_TEXT_STRING) (
IN EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL * This , // Pointer to //EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL instance IN CHAR16
* String // Null- terminated string );
```

This function writes a string to the output device and displays it at the current cursor position. It is the most basic output mechanism.

Here is an example:

gST->ConOut->OutputString(gST->ConOut,L"Hello, UEFI World!\n\r");

This is the core function of the Simple Text Output Protocol. With other functions of this protocol, such as SetAttribute, you can achieve colored backgrounds and colored fonts. In the previous blog, I have shown that you can make a unique command-line Sell program.

2Formatting Output

PrintLib provides support functions for formatted output, which supports all Unicode and ASCII strings. In the dsc file of the Package, the compilation of this library is generally provided. As follows:

PrintLib|MdePkg/ Library /BasePrintLib/BasePrintLib.inf

The formatting method of UEFI is different from the ANSI C standard, which can easily cause conceptual confusion when used.

The formatting syntax is as follows:

%[flags][width][.precision]type

[flags]

- – Left-aligned flag, if not set, it is right-aligned
- Space Add leading spaces to numeric type characters, valid for types X, x and d
- + Sign prefix, showing the positive and negative of numbers, valid for types X, x and d, when used with spaces, the spaces are ignored
- 0 Supplement the left side of the number with leading 0, often used with the subsequent width (width),

valid for types X, x and d;

- , Represent numbers with thousands separators, only valid for type d, when used with flag 0, 0 is ignored;
- L, l Print the specified number as UINT64 type, valid for types X, x and d. If this flag is not specified,

it will be printed as int type;

Note: flags other than the above will be ignored

[width]

- * The width is given by the number in the parameter list. For example, Print(L"%0*d",5,a) means that the variable

a is represented by a width of 5, and the missing width is padded by leading 0s;

- Number This decimal number gives the width to be represented;

Note: If the value of width is not given, it is specified as 0 by default.

[.precision]

- * The width is given by the number in the parameter list;
- number The precision to be represented is given by this decimal number;

Note: If this field is not given, it is specified as 0 by default.

type

- % prints a percent sign
- c prints Unicode characters. This type can also be used for ASCII characters, as long as bits 8…15 are 0;
- x considers the parameter to be printed as an unsigned decimal number and prints it in hexadecimal form. This is different from the ANSI C standard;
- X considers the parameter to be printed as an unsigned decimal number and prints it in hexadecimal form, while filling it with leading 0s. This is different from the ANSI C standard;
- d considers the parameter to be printed as a signed decimal number and prints it in decimal form;
- p The parameter to be printed is a pointer (Void *), and the pointer address is printed in unsigned hexadecimal form;
- a The parameter is a pointer to an ASCII string, which is different from the ANSI C standard;
- S, s The parameters are pointers to Unicode strings, which are different from the ANSI C standard;
- g The parameter is a pointer to a GUID structure, which is used to print GUIDs. Different from the ANSI standard;
- The t parameter is a pointer to the EFI_TIME structure, which is printed in the form of mm/dd/yyyy hh:mm, and the missing digits are filled with 0. Different from the ANSI C standard;
- The r parameter is the RETURN_STATUS value, and the meaning represented by this value is converted into a string and printed out. Different from the ANSI C standard.

The following is a partial conversion comparison: (For the meaning of specific values, please refer to \BaseTools\Source\C\Include\Common\UefiBaseTypes.h in the EDKII code)

RETURN_SUCCESS: "Success"
RETURN_LOAD_ERROR: "Load Error"
RETURN_INVALID_PARAMETER: "Invalid Parameter"
RETURN_UNSUPPORTED: "Unsupported"

Example description:

Although the '+' of flags is valid for both type x and X, I personally think it has no effect because both x and X treat parameters as unsigned.

```
INTN a=-234;
Print("a=%+d\n",a);
Print("a=0x%+x\n",a);
```

The output is:

```
a=-234
a=0xFFFFFF16
```

The more commonly used, but also easily mistaken types are 'a', 'S', and 's'. The meaning is completely different from that in ANSI, and is mainly used to distinguish ANSI strings from Unicode strings in UEFI.

```
CHAR8 * str="Hello, UEFI World!\n";
Print(L"%a",str);
```

Output:

```
Hello, UEFI World!
```

When programming, you need to be careful when using the two encodings. For example, in the above example, you would habitually use "%s" to print, but of course nothing will be printed because str is not a valid Unicode string.

3 Commonly used printing related functions

UINTN EFIAPI Print(IN CONST CHAR* Format, …)

Parameters:

Format: Unicode string terminated by Null
…: Variable parameter list given according to formatting rules

Return:

Output variables to ConOut according to format

Function description:

Output variables through ConOut according to the format string given in Format. If the size of the output variable is greater than PcdUefiLibMaxPrintBufferSize, it will be truncated according to this length and output to ConOut.

UINTN EFIAPI PrintXY (IN UINTN PointX,
IN UINTN PointY,
IN EFI_GRAPHICS_OUTPUT_BLT_PIXEL * ForeGround,
IN EFI_GRAPHICS_OUTPUT_BLT_PIXEL * BackGround,
IN CONST CHAR16 * Format,
…
)

Parameters:

PointX: X coordinate of the position of the displayed string
PointY: Y coordinate of the position of the displayed string
ForeGround: Font color of the string display, can be set to NULL, directly use the background color of the ConOut device
BackGround: Font background color of the string display, can be set to NULL, directly use the background color of the ConOut device
Format: Unicode format string ending with NULL.

…: Variable parameter list given according to formatting rules

Return:

Output variables according to format

Function description:

According to the format string given in Format, in graphics mode, output variables through ConOutputHandle, and the output position is (PointX, PointY). If the output position exceeds the edge of the display, the displayed image will be truncated. Can be used with HII to achieve display.

UINTN EFIAPI AsciiPrint (IN CONST CHAR8 * Format, …)

Parameters:

Format: ASCII string terminated by Null

…: Variable parameter list given according to formatting rules

Return:

Output variables to ConOut according to format

Function description:

Output variables through ConOut according to the format string given in Format. If the size of the output variable is greater than PcdUefiLibMaxPrintBufferSize, it will be truncated according to this length and output to ConOut.

Other functions:

UINTN EFIAPI **StrLen** (IN CONST CHAR16 * String) ;
UINTN EFIAPI **AsciiStrLen** (IN CONST CHAR8 * String) ; CHAR16* EFIAPI **StrCpy** (OUT CHAR16 * Destination, IN CONST CHAR16 * Source); CHAR8* EFIAPI **AsciiStrCpy** (OUT CHAR8 * Destination, IN CONST CHAR8 * Source); INTN EFIAPI **StrCmp** (IN CONST CHAR16 * FirstString, IN CONST CHAR16 * SecondString); INTN EFIAPI **AsciiStrCmp** (IN CONST CHAR8 * FirstString, IN CONST CHAR8 * SecondString); CHAR8* EFIAPI **UnicodeStrToAsciiStr** (IN CONST CHAR16 * Source, OUT CHAR8 * Destination); CHAR16* EFIAPI **AsciiStrToUnicodeStr** (IN CONST CHAR8 * Source, OUT CHAR16 * Destination);