Fonts for Embedded Touch Screen Development

If you are an engineer tasked with getting an embedded touch screen up-and-running and do not have a lot of experience with fonts, foreign languages, or UTF-8 encoding, this article *Fonts for Embedded Touch Screen Development* is worth a read. It gives you some background and tells you how to get the fonts, characters, and encoding you need. It also covers the most commonly asked font-related questions.

Getting Up to Speed

Before you start working with built-in, downloadable, and custom bit-mapped fonts, character sets, and UTF-8 character encoding, we suggest you review some background information.

Built-in, Downloadable, and Custom Fonts

Reach products come with a set of built-in fonts described in the manual for each specific Reach product. If these fonts are not adequate, there is a way to include external fonts in your customer-specific flash image downloaded into the product. You can find fonts ready for download in the Reach Font Library. Often customers request Reach build additional fonts for the following reasons:

1. Custom Design – a graphic designer has specified a look-and-feel using a font that is neither a built-in font nor available for download in the Reach Font Library.

2. Non-English Markets – products sold into non-Western European markets need additional fonts for product localization. Reach built-in fonts use the ISO 8859-1 character set which only covers Western European markets (see a complete list of languages covered).

3. Missing Characters – characters are missing from your standard character set (e.g. the Euro Symbol €).

Which fonts come built-in?

SLCDx Products:

- See SLCDx Reference Manual.

SLCD5 Products:

- Windows Arial
- `Windows Monospac821 BT`

Where can I find additional fonts? The Reach Font Library contains fonts ready for download.

What if I cannot find the font I want built-in, nor in the Reach Font Library? If you do not find your font, request it by contacting Reach Technical Support. See what you will need to specify to get your desired font on page 6.

Character Set

A **character set** is a *defined list of characters* recognized by computer hardware and software (e.g. A-Z, 1-10, and symbols). By default, Reach products use the 256 character extended ASCII character set per ISO 8859-1. To access more than 256 characters, Reach supports the UTF-8 character encoding standard (read more details in *UTF-8 Character Encoding* section below).

Font

A **font** is a *design* for a set of characters, also known as a typeface. A rendered font is the combination of the font itself and other qualities, such as size, pitch, and style (e.g. bold). For example, Arial is a font that defines the shape of each character. Within Arial, however, there are many renderings to choose from -- different sizes, italic, bold, and so on.

Point Size (character height)

The height of characters is measured either in points or in pixels. A point is approximately 1/72 inch. The relationship of points to pixels depends on the pixel density also known as dots per inch or dpi.

Proportional Fonts

A proportional font has character widths that vary depending on the shape of the character. As such, different texts with the same number of characters can have a different total width on the screen. These fonts have a more pleasing appearance to the eye than monospaced fonts.

Monospaced (Fixed-Width) Fonts

A font is said to be monospaced or fixed pitch if every character has the same width. These are useful for displaying numbers since the individual digits stay in the same position no matter which number is displayed.

Built-in Fonts for different SLCD models

Both proportional and monospaced fonts are rendered in different ways depending on your SLCD controller board. For SLCD5 controller boards, consult the "Fonts" section in the "SLCD5 Manual." For SLCD+, SLCD43, and SLCD6 controller boards consult the "Fonts" section in the "SLCDx Software Command Reference." These are available in the Reach Download Center.

Bit-mapped Fonts

Reach products use **bit-mapped fonts**; *every character is represented by an arrangement of dots*. To display a bit-mapped character, displays locate the character's bit-mapped representation stored in memory and display the corresponding dots. Bit-mapped fonts are rendered from True Type or Open Type Windows fonts and then stored in System Independent Format (.SIF) files. One file is created for each font type, size, attribute (italic, bold), and character subset included. Although fonts are copyrighted, the representation of them (e.g. bit-mapped files) cannot be. Once rendered, they are free to use.

Unicode

Unicode is a set of tables that provide a 16-bit index that allows for 65,536 different characters in a given font. Each character is specified by a Unicode code point, which is the table index. See http://en.wikipedia.org/wiki/Unicode.

UTF-8 Character Encoding

To specify a Unicode character, a 16-bit value is needed. Since traditional ASCII is 8 bits, this is a problem. One way to handle this is to use a multi-byte sequence that can be decoded into Unicode characters. The UTF-8 system enables this.

UTF-8 character encoding is gaining traction as the dominant international encoding. It supports almost every language in the world and minimizes the amount of storage space needed for fonts.

UTF-8 provides a way to represent a Unicode Character as a sequence of 8-bit bytes where the number of bytes in the sequence varies based upon where you are in the Unicode Set. For example, the most commonly used 128 Unicode Characters are represented by one byte of UTF-8 from Unicode U+0000 to U+007E. See examples in the table below.

Unicode to UTF-8 Conversion for Example of Commonly Used Characters U - Y

Unicode code point	Character	UTF-8 (hex.)	Name
U+0075	u	75	LATIN SMALL LETTER U
U+0076	v	76	LATIN SMALL LETTER V
U+0077	w	77	LATIN SMALL LETTER W
U+0078	x	78	LATIN SMALL LETTER X
U+0079	y	79	LATIN SMALL LETTER Y

Once above the most common 128 characters, UTF-8 requires two bytes, as shown in the cent sign example below.

Unicode code point	Character	UTF-8 (hex.)	Name
U+00A2	¢	c2 a2	CENT SIGN

Larger character sets (the Chinese, Japanese and Korean (CJK)Unified Ideographs for example) are in the U+4E00 to U+9FFF range, and require three bytes of UTF-8 code for each character.

Unicode code point	Character	UTF-8 (hex.)	Name
U+4E00	一	e4 b8 80	<CJK Ideograph, First>
U+4E01	丁	e4 b8 81	
U+4E02	丂	e4 b8 82	
U+4E03	七	e4 b8 83	
U+4E04	丄	e4 b8 84	

For more information on working with non-English languages on page 12.

Working with Fonts

To use an external font (one that is not built-in), load the fonts into the Reach on-board Flash memory along with bitmaps and macros. A font is contained in a file with the extension .SIF, which contains the binary representation of the rendered font. You need a separate .SIF file for each unique combination of font, size, and attribute (e.g. bold).

In this section, we answer commonly asked font questions.

How do I work with external fonts?

1. Go to the Download Center and download the .SIF files for the fonts you want to use.

2. Create a Font List (.RFL) File: Create a text file with a .RFL file extension. The contents should contain a line for each font in the format:

   ```
   <font_alias> <filename>.sif <CR>
   ```

   The <font_alias> is an alphanumeric string which will be used as an argument to the set font command to select this font. The <filename> is the name of the .SIF file for this font.

   The Reach demo provided with the development kit uses the file "fonts.rfl" to include the rendered font “KaiTi_GB2312_16.sif” and assign it the alias "CH16". It contains one line of text: "CH16 KaiTi_GB2312_16.sif".

3. Load the Font: Use the Windows BMPload program (available in the Reach Download Center) to load external fonts (including Unicode fonts) into Flash memory on the SLCDx board.

4. Use the External Font: To select the loaded font to be used to display text, use the set font command:

   ```
   "f <font_alias>"
   ```

5. If the character set is Unicode, you will need to enable UTF-8 encoding before displaying any characters with Unicode values greater than 255.

How do I enable UTF-8 encoding?

Command: utf8 [on|off]
Example:

```
utf8 on
t "\xe4\xb8\x81"
```

This code displays the Unicode character 0x4e01 whose UTF-8 equivalent is hex E4 B8 81. Note that an embedded host can send the UTF-8 characters as 8-bit bytes, or can use the text escape sequence `\x` before the ASCII Hex value of each byte to be sent, as shown above. The byte sequence for the equivalent of the text command above would be (hex): `74,20,E4,B8,81,0D`

How do I request new external fonts?

If you cannot find the font you need, you can request it by contacting Reach Technical Support. You will need to specify:

1. Font Name: Provide the font name, and the .TTF file if it is not commonly available.

2. Font Size: Provide the height in pixels. Note the height of an individual character is smaller than the overall rendered height because of accents and decenders. See how to match Windows font sizes to SLCDx font sizes on the following page.

How do I match Windows font sizes and SLCDx font sizes?

For accuracy in sizing and locating text on an embedded LCD, rendered fonts need to be specified in pixels rather than points. The conversion of points to pixels depends on the implied dots per inch which can vary between a PC monitor and an embedded display. If you design your graphics on a PC and want to convert to pixels, do the following:

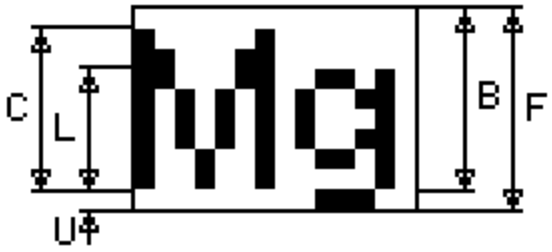
1. Render your font in a program like Paint, Photoshop, or GIMP at whatever point size you want.

2. Setting the image scaling in the program to the appropriate dots per inch (DPI data can be found in the panel specifications, 96 DPI is most common).

3. Enter "Mg" in your font style (in this example Arial), hit return, and then enter "Mg" again.

4. Zoom in and count the number of pixels between the bottom on the first capital M to the bottom of the second g, to give you the actual height (F).

```
F: 10
B: 09
C: 08
L: 06
U: 01
```



3. Font Attributes: Provide attributes if any are needed (e.g. normal, bold, italic) and whether the font should be standard, 2 bit-per-pixel anti-aliased, or 4 bit-per-pixel anti-aliased). Note that anti-aliased fonts can look cleaner at certain font sizes, but take up significantly more memory space. For small high dot density displays like the SLCD43, anti-aliased fonts do not provide a significantly noticeable benefit in most applications.

4. Character/Code Set: A font contains anywhere from 256 to 65,536 characters. When you request a font from Reach, you can specify multiple ranges or individual characters to be included. Here are some things to consider, before selecting your code set direction:

    a. Should I load the whole character set? Arial Unicode MS, for example, has 38,000 characters, which means you end up with a 2MB file just for one font. Since most Reach products have 4MB of storage, half the onboard storage would be consumed for one font. It is better to only encode a certain range, or specific characters, and restrict the number of font sizes and attributes.

    For Asian fonts that render into very large .SIF files, consider specifying a font with a reduced number of characters (glyphs). These tend to be the commonly used ones. Good choices are KaiTi_GB2312 which has 7,580 glyphs and "TSC JSong S TT" with 8,160 glyphs.

    b. Can I just select specific characters for an external font file (.SIF)? Yes, see Option C, "Individual Characters" on the following page.

    c. What is a common code set for non-Asian languages? The code set from U+0000 to U+07FF covers most European languages plus Arabic, Hebrew, and others.

Our technical support team will need to know which way you want to go:

A. Standard ASCII character set per ISO 8859-1 (256 characters)**:** Reach built-in fonts use the ISO 8859-1 character set which covers Western European markets (see a complete list of languages covered). See the ISO 8859-1 character set used by most fonts in the SLCD product manual, or see other ISO sets defined at http://en.wikipedia.org/wiki/ISO_8859 . Note: The ASCII character set is the same as the ISO up to Code 127. The ISO set does not define characters 0-31, or 127-159.

B. Unicode Character Set**:** If you do not see the characters you need in the standard characters sets, or if you are selling into non-Western European markets and need additional fonts for product localization, you will need to select the appropriate block of Unicode (or Unicode subset) characters. See a list of all Unicode characters broken into 30 blocks (e.g. Latin, Cyrillic, and CJK Unified Ideographs). Consult the Unicode to UTF-8 Conversion Table to get the UTF-8 code you need or take your source file (figure out what font and which characters you want), copy the desired characters into Microsoft Notepad, and then use menu option "File->Save As," a .TXT file with Encoding set to UTF-8. You can then open the file in a Hex Editor (such as Neo) to get the resulting UTF-8 code.

Here is an example: Say you wanted a button that said, "Start" in Chinese (Simplified), represented by these characters, "开始." Paste those characters into Microsoft Notepad. Save as a .TXT file with Encoding set to UTF-8. Then open that file with a Hex Editor (such as Neo) and get the resulting code. In this case: `EF BB BF E5 BC 80 E5 A7 8B`. The first three bytes are inserted by Notepad to indicate the file is UTF-8 encoded. The two characters in the file are `E5,BC,80` and `E5,A7,8B`. You can use a tool like http://software.hixie.ch/utilities/cgi/unicode-decoder/utf8-decoder to decode the UTF-8 and provide more details, such as character names:
U+5F00 CJK UNIFIED IDEOGRAPH character (&#x5F00)
U+59CB CJK UNIFIED IDEOGRAPH character (&#x59CB)

C. Individual Characters**:** When you require an external font file (especially using localized characters sets), it is common that an entire character set may be too large for the memory on your display controller. Often even selecting ranges of characters will create .SIF files in the multi-megabyte size range. The solution is to create an external font file (.SIF) with only the characters you need for your project. The fastest way to specify and create specific characters in a external font file (.SIF) is to use a "Pattern File". To create a Pattern File follow these steps.

   a. Under Windows, cut and paste your characters into Microsoft Notepad.

   b. When finished, use the menu option "File->Save As," a .TXT file with Encoding set to "UTF-8".

   c. Send the saved Pattern File to Technical Support with the desired font (name, size, attributes).

Once Reach Technical Support receives your request, Reach will generate a .SIF from your specifications.

We need to use some special characters, how do we make them work?

Here is a 'C' code example of how to get the SLCD to display special characters, like quotes:

```
void demoSpecialChars(void)
{
char buf[120];

// backslash in a string (using backslash as an escape)
sprintf(buf,"t \"%s\"\r","Backslash: \\\\");
sendToSerialPort(buf);
sendToSerialPort("t \"\n\"\r"); // newline, return in string

// backslash as a character (using backslash as an escape)
sprintf(buf,"t \"%s%c%c\"\r","Backslash: ",'\\','\\');
sendToSerialPort(buf);

// percent in a string (using percent as an escape)
sprintf(buf,"t \"Percent: %%\"\r"); // Percent
sendToSerialPort(buf);

// Quote (need 2 to get one for SLCD, and one more to allow " in string)
sprintf(buf,"t \"%s\"\r","Quote: \\\"");
sendToSerialPort(buf);

// Quote as character
sprintf(buf,"t \"%s%c%c\"\r","Quote: ",'\\','"');
sendToSerialPort(buf);
}
```

Our font does not look right. Some characters look bold, and some do not. What could be the cause?

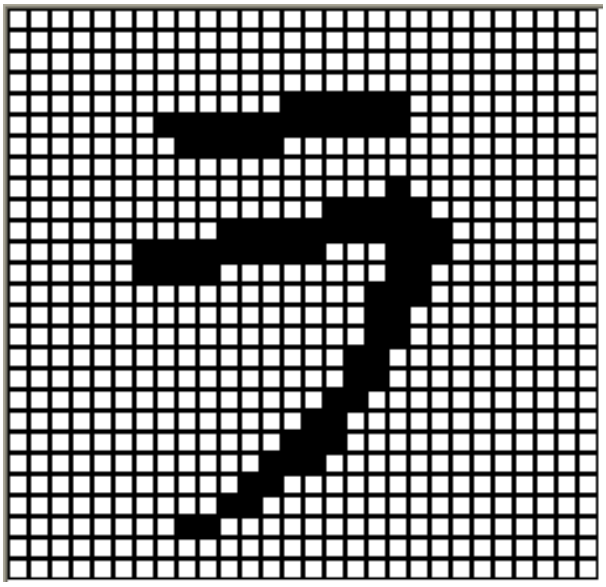
Intermixing Unicode code sets is often the cause of this problem. Below is an example where Katakana (phonetic) and CJK (Unified Ideograph) code set characters are both used in the same text.

```
// English, French, Spanish, Portuguese, Italian, German,
// Japanese, Chinese, Korean
const char* fd1_Speed[LANG_MAX+1] = {
"Speed", "Vites", "Veloc", "Veloc", "Veloci", "Gwkeit",
"\\xE3\\x83\\xA9\\xE3\\x82\\xA4\\xE3\\x83\\xB3\\xE9\\x80\\x9F\\xE5\\xBA\\xA6",
"\\xE9\\x80\\x9F\\xE5\\xBA\\xA6",
\\xEC\\x84\\xA0\\xEC\\x86\\x8D };
```

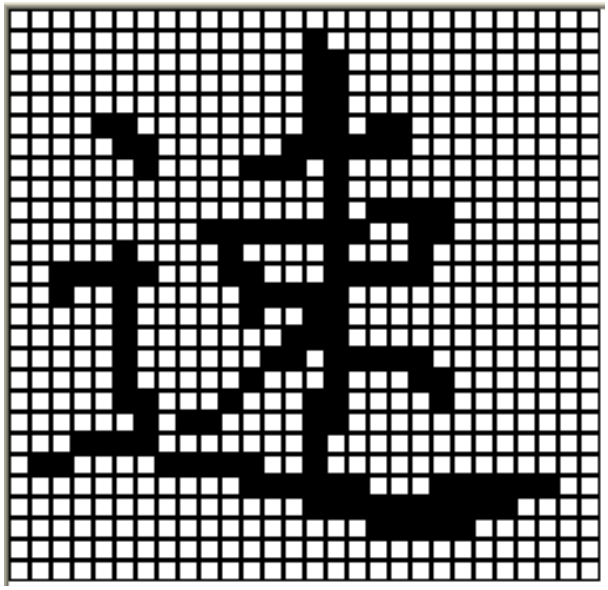
The Japanese UTF-8 code represents:
U+30E9 KATAKANA LETTER RA character (0x30E9;)
U+30A4 KATAKANA LETTER I character (0x30A4;)
U+30F3 KATAKANA LETTER N character (0x30F3;)
U+901F CJK UNIFIED IDEOGRAPH character (0x901F)
U+5EA6 CJK UNIFIED IDEOGRAPH character (0x5EA6)

Ideograph characters with much detail will look "thinner" than those with less detail:

Rendered U+30E9 (KATAKANA):



Rendered U+901F (CJK):



Note: There is a useful UTF-8 decoder tool at http://software.hixie.ch/utilities/cgi/unicode-decoder/utf8-decoder.

How do I deal with multiple languages?

In most cases, you will need an external font. Follow instructions on page 6.

Here is a code sample used by a customer that supports nine different languages.
This customer uses C programming in the microcontroller that is talking to our device; below is a list of different language strings used to represent the English word "Length".

```
// English, French, Spanish, Portuguese, Italian, German,
Japanese, Chinese, Korean
const char* fd1_Length[LANG_MAX+1] = {
"Length", "Long", "Long", "Compr", "Lunghe", "Länge",
"\\xE6\\x9D\\xA1\\xE9\\x95\\xB7",
"\\xE9\\x95\\xBF\\xE5\\xBA\\xA6",
"\\xEA\\xB8\\xB8\\xEC\\x9D\\xB4"
};
```

Results by Language

English: Length
French: Long
Spanish: Long
Portuguese: Compr
Italian: Lunghe
German: Länge
Japanese: 条長
Chinese:长度
Korean: 길이

Note: Windows does not always ship with the Asian Character Set Installed. Find instructions to enable the Asian Characters Set at http://newton.uor.edu/Departments%26Programs/AsianStudiesDept/Language/asianlanguageinstallation_XP.html.

How do I handle fonts in India with multiple languages?

You will need to follow the steps outlined for “How do I request new external fonts?” on page 6.

Per Wikipedia: ISCII uses 8-bit code which is an extension of the 7-bit ASCII code containing the basic alphabet required for the 10 Indian scripts which have originated from the Brahmi script. There are 15 officially recognized languages in India. Apart from Perso-Arabic scripts, all the other 10 scripts used for Indian languages have evolved from the ancient Brahmi script and have a common phonetic structure, making a common character set possible. The ISCII Code table is a superset of all the characters required in the Brahmi based Indian scripts. For convenience, the alphabet of the official script Devnagari is used as the standard. If you can provide a font with ISCII characters, we can provide downloadable fonts to access as 8-bit characters.

How do I embed foreign language text in a macro?

The SLCD controller family uses UTF-8 encoding for non-ASCII characters. To incorporate UTF-8 characters into a macro file, you need to use an editor that supports UTF-8 encoding. Windows Notepad can do this. First, make sure you have a Unicode font installed (Arial Unicode MS comes with Microsoft Office) on your desktop so you can see the Unicode characters. Then start Notepad and set Format->Font to a Unicode font. Then select File->Save (or Save As...) and at the bottom of the pop-up box you will see Encoding: ANSII. Click on the drop-down box and select UTF-8. Then save to set the encoding. Then, copy the following into the editor:

```
#define test
m test2 "Нажмите здесь"
#end

#define test2
// display text argument somewhere
t "`0`" 10 20
#end
```

If you see the Russian text, then save the file as "testMacro.txt" (the .txt extension is required), use BMPload to install this macro file and Unicode font to the SLCDx. Exit the BMPload program and use a terminal emulator (like TeraTerm, RealTerm, or HyperTerm) to send the SLCD commands to select the font, and run the macro. You should see the Russian for "Press here" (or "Click") on the screen.