

Creating a Multi-Lingual User Interface From a Commercial GUI Toolkit

Embedded devices or control systems used by the military and deployed worldwide can be enhanced by multi-lingual graphical user interfaces so that they may be operated using different native languages. Developing a multi-lingual graphical user interface can be difficult, especially when the languages use extensive alphabets, but there are commercial tools available to simplify the tasks.

Ken Maxwell, President and CEO
and
Jim DeLisle, VP and CTO
Swell Software

When working within the seemingly boundless landscape of a graphical user interface, software developers are often loath to contemplate the display of mundane items such as text. However, textual information often accounts for the most significant portion of many complex user interfaces, graphical or otherwise. When the total display of numeric data, system messages and navigation prompts are considered, it quickly becomes apparent that a typical graphical user interface requires the display of a large amount of text.

Historically, a graphical user interface (GUI) toolset is used to allow an embedded systems designer to display text easily and efficiently in various colors, sizes and fonts. Beyond this, full-featured GUIs may also incorporate enhanced functionality such as horizontal and vertical orientation; right, left and center justification within a given screen area; text wrapping; and, ultimately, the ability to display multiple languages at runtime. If an embedded device or control system is intended for use in many locations around the world and by people with many different native languages, it's very possible that a priority requirement will be that its controlling software be able to switch seamlessly between a number of active languages.

For example, consider a global positioning system (GPS)—either with handheld or on any type of vehicle—that requires the ability to display waypoints and visual recognition cues in any language. It's very important that an operator from virtually any country be able to sit down in front of this device and easily determine its operation as well as understand the information the device is relaying. Or consider the possibility of multi-lingual support for radar systems, targeting systems or any type of multifunction display (MFD) such as might be found in aircraft cockpits. Wherever there's an input and/or output device with which an operator interacts, that device can be a good candidate for multi-lingual text support.

Developing a multi-lingual application is no trivial task. Even when working in the desktop world with what may seem by comparison to be nearly unlimited resources, the prospect is daunting to most software developers. When limited memory size, possible lack of a file system and the problems implied by the legacy hardware platforms of many embedded systems are added to the equation, the ability to support multiple languages may at first seem beyond reach. In addition, many software developers are intimidated by such a proposition simply due to a lack of knowledge of foreign language systems and foreign alphabets.

However, an advanced GUI toolkit, such as the Portable Embedded GUI (PEG) from Swell Software, can ease the process of creating ROMable, real-time,

multi-lingual embedded applications. A comprehensive embedded application built with this toolkit typically requires only a couple hundred kilobytes of code space, will run on nearly any type of embedded hardware and may execute stand-alone or with any popular RTOS.

Large Alphabets

When moving to or developing a multi-language system, the first decision has to be to determine the target audience for the application. This decision will mandate the character sets or alphabets that the application will need to support.

Most North American software developers are very well versed in the ASCII character set. This character set does a good job at supporting the English language. With a bit of effort, the character set can be expanded to a full 8 bits, and, assuming that the "0," or NULL character, is used for a string termination, it allows up to 255 total characters. According to the Unicode Standard, Version 2.0, this relatively small character set will allow the support of the English, Danish, Dutch, Farsi, Finnish, German, Icelandic, Irish, Italian, Norwegian, Portuguese, Spanish and Swedish languages.

Unfortunately, even a total of 255 characters may not meet the requirements of an application's target audience. Most notably, the French, Russian, Turkish, Hebrew, Arabic, Japanese, Chinese, and Korean languages aren't completely supported by this limited 255-character

The Softer Side

alphabet. In particular, the Japanese, Chinese and Korean alphabets, which are each derived from ancient Chinese symbols, require approximately 28,000 characters to use the full Chinese-Japanese-Korean (CJK) alphabet. Consequently, these larger character sets require the use of an alternate encoding form.

There are many character-encoding standards in use today, but, by far, the most widely used is the Unicode encoding system. This system has the advantage of having been adopted by the International Standards Organization (ISO) as ISO 10646-1, which is now also supported on the X11 Window System (for Unix and other operating systems) and variations of the Microsoft Windows operating system. Unicode appears to be well on the road to becoming the global standard character encoding system.

Once a decision has been made to use a two-byte character encoding system, notably Unicode, the next item to verify is tool chain support. The Standard C Library supports two-byte character encoding through the use of the "wchar_t" data type. Most C/C++ compilers, both commercial and open source, have built-in support for string manipulation using this data type. If the compiler included in the chosen tool chain for application development doesn't support this feature, then alternatives to the Standard C runtime routines must be devised and implemented.

For just this reason, PEG provides a replacement string library to ensure that the software will run on any target including those compilers that have limited or no support for two-byte characters. However, using the PEG string library is optional and only required when the chosen toolset provides no support for two-byte character encoding.

String Tables

To write an application that can determine, at runtime, which set of strings to use for the active language generally means that the strings themselves are isolated from the code, off to the side in nice neat tables. In C/C++, these "string tables" are nothing more than arrays of pointers to the strings. One array is required for each supported lan-

guage. At runtime, each object that displays text must obtain the string pointer associated with the object via an array index. The need to create and manage these string tables is so prevalent in multi-lingual embedded applications that PEG provides a complete utility, aptly named the String Table Editor, for creating and maintaining string tables.

As shown in Figure 1, each column in the string table corresponds to one language, and the first table column contains a list of user-defined string identification (ID) values. These string ID values are used to generate a simple enumerated list in C; that is, these ID values become the array indices previously described.

Using the String Table Editor is very simple. The application developer first configures a project file, specifies how many languages the application will support and assigns a name, or alias, to each language. This, in effect, determines how many columns will be required in the string table.

Next, the strings for the first, or "reference," language are created. The reference language can be any language. However, generally the reference language will be English, since the strings for the reference language become the default strings used by the application when no translation is provided for the active language. This occurs for proper

names, like California, and for many technical terms and trade names for which no translation is required.

Once the reference language strings are entered into the string table, the next step is to get translations of each string for each of the supported languages. Most often, application engineers won't be fluent in every language supported by the application. As a result, it's most likely that translators will be called on to support the activity. Usually copies of the PEG project file will be sent out to translators—who may be college professors, foreign associates, engineers for hire in the target countries—or to whomever else can properly translate each of the reference language strings.

For many languages with large alphabets, the process of entering the string data itself can be a challenge. In Japan, for example, formal input methods exist for generating traditional Kanji characters from the simpler phonetic Katakana or Hiragana alphabets. Many countries use special keyboards containing characters not found on a typical U.S. ASCII keyboard. In order to allow editing strings in any language, the PEG String Table Editor displays a complete list of characters contained in the chosen font and allows the characters to be selected simply by clicking on them with a mouse. Alternatively, text may be cut and pasted

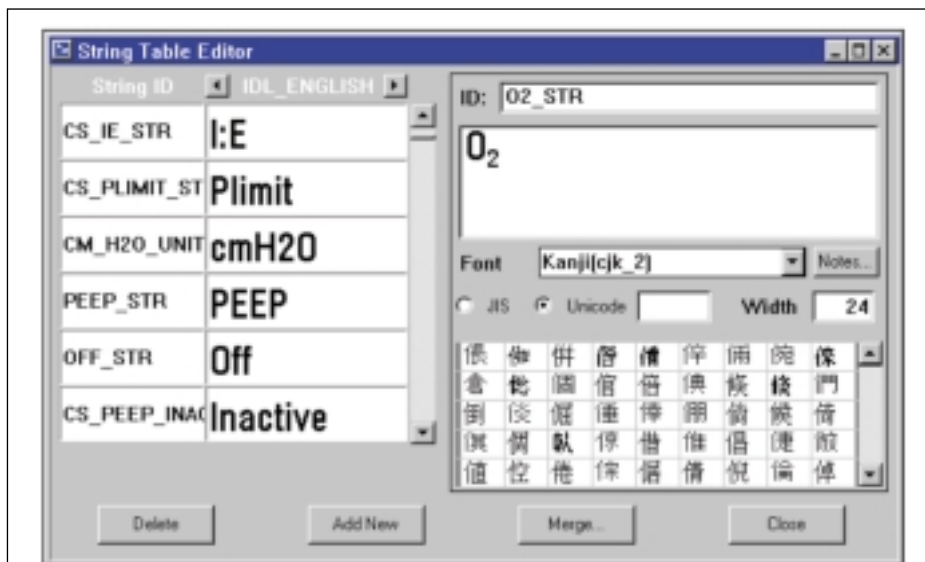


Figure 1

An example of a string table of the Portable Embedded GUI (PEG) from Swell Software.

from other applications when running the utility under Microsoft Windows.

There are a few important features built into the String Table Editor to facilitate the translation process. First, each string can have a short note or comment associated with it. This can be useful for explaining how the string is used because the context can be very important when translating. For example, one might enter a note for the string “Run” that explains that this button activates the machine and isn’t a dire warning to clear out of the area physically.

Another consideration can be the size, in terms of pixel width, associated with each string. This isn’t much of a concern for a multi-line edit control but can be an important factor in determining the best translation, for example, for a menu command. The String Table Editor dynamically calculates the width, in pixels, of each string using the selected font. This allows the translator to determine whether the translated string will fit within the graphical element used to display the text.

Finally, once translated strings arrive back from the translators, they must be merged in with the existing strings in the local project’s string table. The String Table Editor provides a merge facility specifically for this purpose. The merge facility collects the strings from a second project and merges these strings back into the primary project file.

When the string table is complete and all translations are merged, the String Table Editor includes an export

facility that exports the string table data into two standard C source files, ready to be compiled and used in the application. The names and locations of these generated source files are specified during the project configuration. One source file contains the actual string data, properly formatted as two bytes per character with terminating nulls, and the pointer arrays previously described. The second source file is a header file containing the string ID enumerations and language names. All modules in the project will need to include this header file in order to use the string table for string lookup.

At this point, the application designer may also choose to distill a font used for any given language. As discussed earlier, some languages require tens of thousands of characters to make up the complete alphabet. Chances are, even in very complex applications, a vast majority of those characters are never used in the application. The GUI provides the means to allow the application designer the flexibility to specify a range or a certain font to keep with the application. Therefore, only a predetermined subset is compiled in with the application and loaded at runtime.

Application designers also have the option of letting PEG do this range setting for them. It can be instructed to create a composite font from any given font and to include in the composite font only those characters that are actually in use in the application’s string tables. This technique has the potential to save an order of magnitude in precious runtime memory space.

Putting It All Together

At runtime, each GUI object that displays text runs a simple lookup macro using the associated string ID and the active language to determine the actual text for display. Switching the active language changes the array of string pointers used to find each string and the program can run effortlessly in any supported language.

Figure 2 is a side-by-side comparison of the same object running while displaying Japanese and English characters. Notice how the PEG object names in the text on the window don’t change, having reverted to the default language, and are displayed in the appropriate font.

From a practical standpoint, this technology is important to any manufacturer that has considered opening up its products to foreign markets, but it can be especially important today for the military hardware market where multiple strategic global alliances are in place. This includes not only those markets where the alphabet is specifically Unicode but also markets, such as Germany, Italy and Sweden, whose alphabets fit into the ASCII character range. Developing systems and embedded devices with built-in multi-lingual support provide “legs across the pond” and demonstrate international awareness.

There are many other details related to multi-lingual embedded systems—such as producing the necessary fonts, other localization issues and text orientation (right to left or top to bottom instead of left to right)—that have been intentionally omitted from this discussion. However, the challenges involved in working with a GUI for an embedded system such as would be found in military applications should be clear. The development process of creating these multi-lingual embedded applications can be greatly simplified and automated by the use of readily available tools such as PEG. ■■

Swell Software
Port Huron, MI.
(810) 982-5955.
[www.swellsoftware.com].

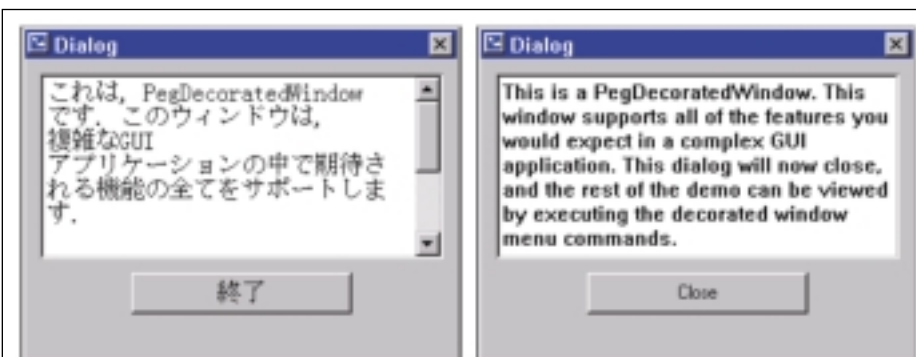


Figure 2

A side-by-side comparison of the same object running while displaying Japanese and English characters. The object names in the text on the window have reverted to the default language, and they are displayed in the appropriate font.