Internationalization in GTK+

Owen Taylor
Red Hat Software

otaylor@redhat.com

GTK+ provides the basic tools for writing applications that can work in the user's native language and writing system. The current implementation uses the facilities of the operating system and the X libraries and supports the major languages of Europe and East Asia. Future plans include suppport for the Unicode standard and languages written in a right-to-left direction.

Introduction

Writing software that works for people in all countries, all languages, and all scripts is a challenge. There are a number of issues that must be addressed. First, provisions must be made to allow the user to input text in their native language. This can be a simple matter of changing the keyboard mapping. (As it is usually is for European languages) or it can be a highly complex process involving dictionary lookups. The latter situation often comes up for the languages of East Asia, where the user's phonetic input must be converted into the correct ideograph from a set of thousands of characters. (These languages are often known as the CJK languages, for China, Japan, and Korea.)

Output is another issue that must be dealt with. When displaying CJK text, the primary difficulty is that the fonts needed contain a very large set of characters, so instead of the 8 bits that are sufficient for displaying Roman text, 16 bits are needed per character to index the font. When displaying mixed Roman and CJK text, it is often necessary to use multiple fonts when displaying a single string. A different set of problems comes up for the languages of the Middle East, which are written predominately in a right-to-left direction instead of a left-to-right direction. For these languages, it is necessary to reorder the characters before displaying them on the screen.

Terminology

There are a number of terms that are commonly used when discussing writing software to be used in international settings. First, the terms *internationalization* and *localization* refer to the process of making software support a range of languages, and to the process of adapting the messages and conventions of a program to those of a particular locale, respectively. These terms are often abbreviated i18n and l10n respectively, after the number of letters between the first and last letters of the word.

The *locale* is the set of settings for the user's country and/or language. It is usually specified by a string like `en_UK`. The first two letters identify the language (English) the second two the country (the United Kingdom). Included in the locale is information about things like the currency for the country and how numbers are formatted, but, more importantly, it describes the characters used for the language. The *character set* is the set of characters used to display the language. When storing characters in memory or on disk, a given character set may be stored in different ways - the way it is stored is termed the *encoding*.

Handling international text is complicated by the fact that the encoding (especially for languages with large character sets, like the Asian languages) may be somewhat different than that used for English or European text - each character does not fit into a single byte. (Since there are more than 256 characters in the character set).

There are two basic strategies for dealing with such characters. In a *multi-byte* encoding, each character is represented as a variable number of bytes. As an example of such an encoding, in the commonly used EUC encoding, bytes less than 128 are simply ASCII characters, while bytes bytes greater than 128 are taken in pairs to represent extended portions of the character set. Since multi-byte encodings are usually backwards compatible with ASCII they are convenient to handle for programs that just want to use strings opaquely. However, because each character is a different number of bytes, it is difficult if a program needs to look at the bytes of the string one-by-one. In *wide-character* encodings, every character is the same width. (For instance, each character is two bytes.) Wide character strings are generally easier to manipulate, but have poor backwards compatibility.

Internationalization in GTK+

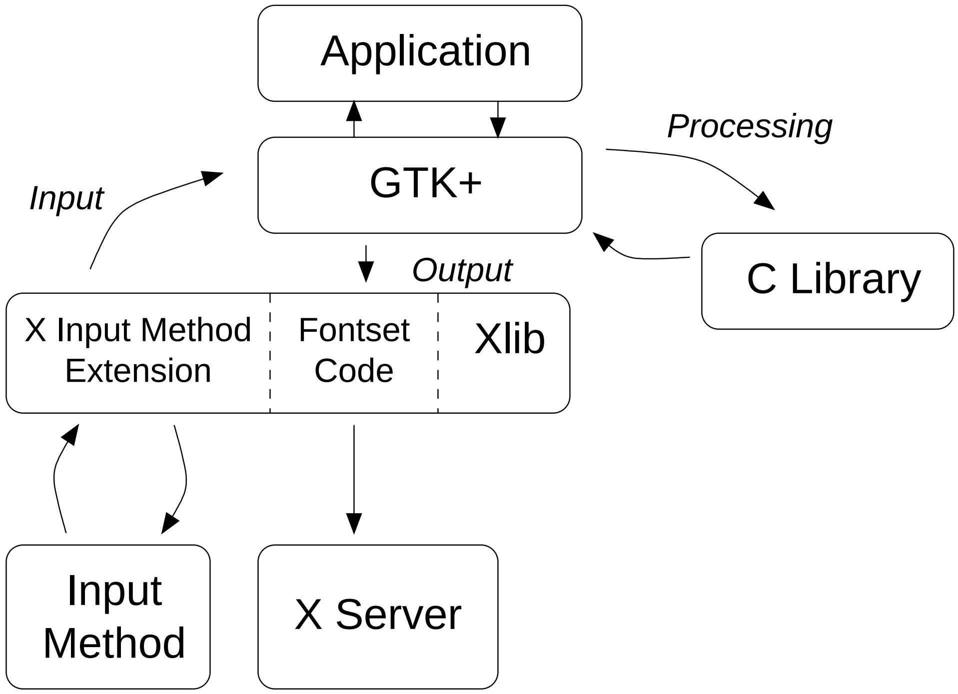

Figure 1. Current architecture of internationalization in GTK+.

Manipulating Text

The most basic task that GTK+ (and applications using GTK+) have to handle when dealing with international text is manipulating strings. The strings in the GTK+ interfaces are handled in the multi-byte encoding for the locale. This allows good compatibility with existing applications that aren't explicitely enabled for multi-byte support. Internally, GTK+ converts these strings to wide-character strings for easier manipulation, and the conversion routines are also available for applications that need these facilties.

Input

Internationalized input in GTK+ is done using the X Input Method extension (XIM). The X Input Method extension is an interface between X Window System applications and *input methods*. An input method handles translating keystrokes into characters. The X libraries include a simple built in input method that does compose-key handling for European languages. The more complicated input method handling for Asian languages is typically done by an external program.

You can see the basic architecture of XIM in Figure 1. GTK+ forwards the keystrokes it receives to the input method via Xlib, and when a complete input string is received, it is displayed to the user. From the point of view of an application which is just using the standard GTK+ Text and Entry widgets, this is all done transparently behind the scenes, and the application only sees the final strings.

The X Input Method extension provides a complete set of facilities, and input methods are available for Chinese, Japanese and Korean. However, the programming interface that XIM provides has a number of disadvantages. For one thing, strings are communicated using the native encoding for the current locale and input methods are also selected by looking at the current locale. This makes it difficult to use XIM to do input in multiple languages at once. Also, the functioning of XIM is very closely tied to the event handling and text string models of X. An application that renders strings itself (for instance, an illustration program) will have a hard time using XIM.

Output

GTK+ handles output of strings in different scripts using *font sets*. A font set is a list of X fonts for different character sets. When drawing, for example, mixed Roman and CJK text, then the two different fonts needed are extracted with multiple character sets.

The font to use for a particular widget is generally determined in GTK+ using resource configuration (RC) files. There is a system-wide file that the system admininstrator can set up, and additionally, each user can override the settings by creating a file in his home directory. This mechanism has been extended to deal with getting the correct fonts for each locale, even for user's that switch between different locales. When GTK+ reads in a RC file — for example, the file `.gtkrc` in the user's home directory — it also looks for the same file with an extension corresponding to the current locale. If the locale is `ja_JP` then GTK+ will check for the file `.gtkrc.ja`. GTK+ ships with gtkrc files for Japanese, Korean, and Russian, and a system administrator can easily create them for additional languages as needed.

Future Directions

The current internationalization facilities in GTK+ are sufficient for creating applications that work well for a wide range of languages - the languages of both Western and Eastern Europe (including Cyrillic and Greek) and of East Asia (Chinese, Japanese, and Korean.) However, there are still a considerable range of languages that are not covered. Most prominently, currently released versions of GTK+ cannot handle languages where the primary writing direction is right-to-left, such as Arabic and Hebrew. Handling these languages is a challenge, because texts usually mix together text read from right-to-left and from left-to-right, so a complicated reordering process is needed to take the input text and display it on the screen.

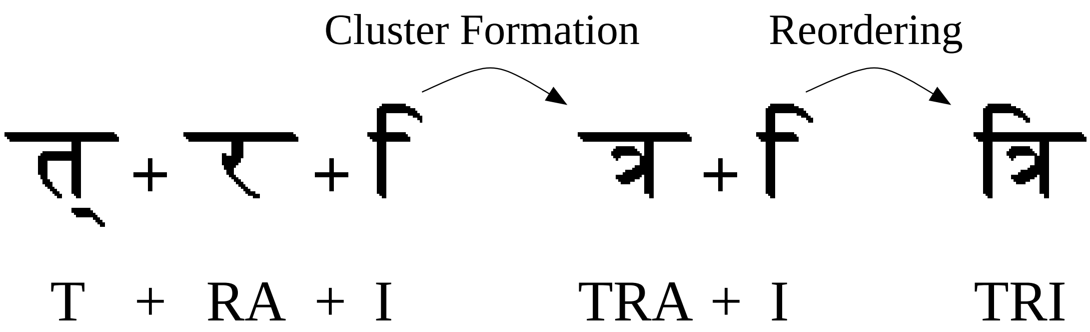

Figure 2. Transformations while displaying complex-text languages.

Another class of languages for which support is currently being developed are the so-called *complex text* languages. In the writing systems of South and South-East Asia, when letters are put together, they combine to form clusters which can differ considerably in shape from the original letters. See Figure 2.

To support these scripts, and also to make it easier for application developers to fully use the support GTK+ already has for international scripts, GTK+ will be moving to using Unicode to encode all strings, instead of the current system where the encoding is chosen per locale. Because the encoding is the same for all locales, code to manipulate strings is easier to write and more efficient. In addition, the conversion will improve interoperability with the many other systems that are currently standardizing on Unicode.

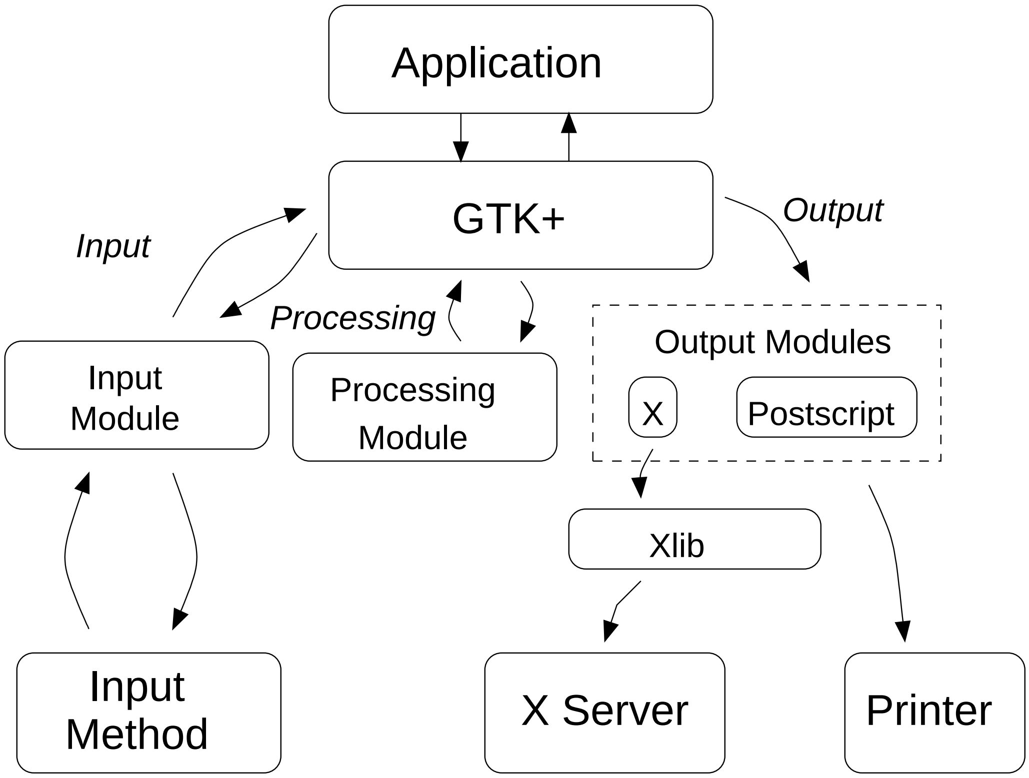

Figure 3. Proposed future architecture of internationalization in GTK+.

Because the rules for forming the writing for each different script are complex, it is not desirable to build all the necessary intelligence into GTK+ directly. Instead we will use modules that contain all the intelligence necessary. A module will be written for a language or group of related languages and will contain the necessary knowledge to input, process, and output the text for that language. Actually, each module will be composed of multiple parts, so that the portions of code specific to one toolkit or output device can be separated out from the portions that can be shared between in a system-independent manner. The proposed architecture is shown in Figure 3.

GTK+ already provides facilities to allow developers to internationalize their applications for a wide range of languages. When the above changes are complete, GTK+ will be able to handle all of the worlds languages in a sophisticated and flexible manner.