The m17n Library –A General Purpose Multilingual Library for Unix/Linux Applications–

Nishikimi Mikiko and **Handa Kenichi** and **Takahashi Naoto** and **Tomura Satoru**
National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba
{nisikimi, handa, ntakahas, tomura}@m17n.org

Abstract

We have designed and implemented a software library to multilingualize application programs on Unix/Linux systems. This library, named *the m17n library* provides functions necessary to handle multilingual text. Since most of these functions are implemented by modifying or expanding widespread C library and X library, it is easy and straightforward for developers to multilingualize existing software using this library. The m17n library reduces the cost of internationalization/multilingualization in software development, and users of every language, common or rare, can benefit from the same computer interface in their own languages.

The basic part of the m17n library was released under GNU Lesser General Public License (LGPL) on March 1st, 2004.

1 Introduction

When working with computers, we use written languages as a significant means for conveying information. However, not all languages in the world are equally well supported. Some languages, such as English, enjoy various convenient tools at low cost, while some other languages, especially Asian ones, still have difficulties even in displaying simple text. This situation is a kind of digital divide and should be improved with multilingualization, which means making software handle multiple cultural conventions including characters, scripts, languages, and orthographies.

When developers create application programs for English speaking users, they can use various libraries to display, input, and handle the language. To write internationalized or multilingualized application programs, on the other hand, such libraries are rarely at hand. Which means each developer has to write his/her own version of language handling codes. This is a great waste of time and effort because most application programs require the same functionalities for internationalization/multilingualization.

In order to improve the situation, we have designed and implemented a software library to multilingualize application programs on Unix/Linux systems. Our library, named *the m17n library* [1] provides functions that are necessary to handle multilingual characters and scripts. At the time of this writing, the m17n library can display more than 20 scripts correctly and provides 12 input methods for Asian languages. Figure 1 shows a sample output rendered by the m17n library.

Section 2 outlines the m17n library and in Section 3 we show how multilingual characters are represented in our system. In Sections 4 and 5, input methods and display routines of the m17n library are explained.

[1] The word “multilingualization” is sometimes abbreviated to “m17n”, that is, ‘m’ followed by 17 letters followed by ‘n’.

English	Hello
French (français)	Bonjour, Salut
German (Deutsch)	Guten Tag, Grüß Gott
Slovak (slovensky)	Dobrý deň
Russian (русский)	Здравствуйте!
Greek (ελληνικά)	Γειάσας
Turkish (Türkçe)	Merhaba
Georgian (ქართული)	გამარჯობათ!
Arabic (اَلْعَرَبِيَّةُ)	اَلسَّلاَمُ عَلَيْكُم
Persian (فارسی)	سلام عليكم
Hebrew (עִבְרִית)	שָׁלוֹם
Amharic (አማርኛ)	ሠላም
Hindi (हिंदी)	नमस्ते, नमस्कार ।
Malayalam (മലയാളം)	നമസ്കാരം (ex: ന+്+ര=ന്ര)
Tamil (தமிழ்)	வணக்கம்
Tibetan (བོད་སྐད་)	བཀྲ་ཤིས་བདེ་ལེགས༎ (ex: བསྒྲུབས, ཧཱུྃ)
Khmer (ខ្មែរ)	ជំរាបសួរ (ex: ក្រឿ)
Vietnamese (**Tiếng Việt**)	**Chào bạn**
Thai (ภาษาไทย)	สวัสดีครับ, สวัสดีค่ะ
Lao (ພາສາລາວ)	ສະບາຍດີ, ຂໍໃຫ້ໂຊກດີ
Japanese (日本語)	こんにちは
Chinese (汉语,普通话)	你好
Korean (한글)	안녕하세요

Figure 1: Sample of rendering by the m17n library

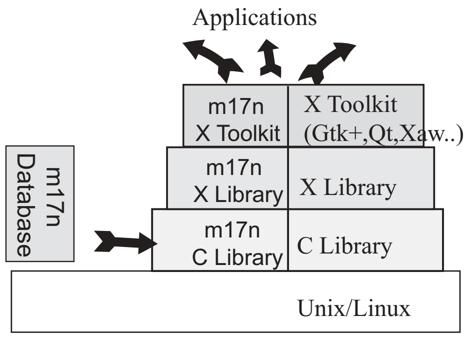

Figure 2: The structure of the m17n library

2 The m17n Library

2.1 parallelism

The m17n library consists of three layers: *m17n C library*, *m17n X library*, and *m17n X Toolkit*. Each layer corresponds to a legacy library (See Figure 2). These layers are language independent; information specific to each language is stored in *the m17n database* and loaded to the m17n library on demand.

The m17n library is designed to be parallel to legacy libraries; functions provided by the m17n library are multilingual version of corresponding functions provided by legacy libraries. This parallelism has the following merits.

- Developers who are familiar with legacy libraries can easily learn the usage of the m17n library to write new multilingual programs.
- Currently existing monolingual programs can be easily multilingualized.

Considering the richness of existing application programs, the latter point is very important. For example, suppose an application program draws an English phrase in a window using a legacy monolingual function. Changing the data string from English to another language may not be enough to display a foreign phrase because some languages require complicated rendering (changing glyph forms, reordering character orders, etc.) to be displayed correctly. If, however, the drawing function is replaced with the corresponding m17n function, language specific rendering is performed automatically. Which means, developers can develop a multilingual program as easily as developing a monolingual program.

2.2 Language dependent information vs. language independent functions

In designing the m17n library, we clearly separated language/script dependent information and language/script independent functions; the latter is assigned to the three layers described above and the former is stored in *the m17n database*.

The m17n database keeps information about, for example, which font should be used to display a certain script, how a certain font is encoded, what kind of input systems are supported for a certain language, and so on. The m17n library dynamically loads necessary information from the m17n database to handle multilingual text.

Because of this separation, it is fairly easy to add supports for new languages and scripts to the m17n library. Preparing the information necessary to handle new language/script in the format of the m17n database is much easier than writing programs for that purpose. Therefore users who can extend the m17n database are not restricted to application programmers or multilingualization specialists.

Currently, the m17n database provides the information listed below. However, this list should be considered as an example for handling limited tasks in limited languages. We keep adding new data and, if necessary, the users of the m17n library can extend the m17n database themselves as described above.

- Various character properties defined in the Unicode standard: general category, combining class, BIDI category, case-folding mapping, complicated case-folding mapping, character name, script name.
- 29 input methods for various languages and scripts including Bengali, Chinese, Devanagari, Gujarati, Gurmukhi, Japanese, Kannada, Korean Malayalam, Oriya, Persian, Tamil, Telugu, Thai, Tibetan.
- 16 Font Layout Tables for 9 scripts (Arabic, Devanagari, Hebrew, Khmer, Lao, Malayalam, Thai, Tibetan, Tamil). Font Layout Tables keep font specific data required for

complex text layout. See Section 5 for the details.

- 3 fontsets: TrueType, X font, and the default. Fontsets provide rules to select a font for each character in the string based on the script, language, or charset property of the character. The TrueType fontset consists of freely available TrueType fonts. The X fontset provides fonts that are accessible via the X protocol. The default fontset allows the access to the both.
- Miscellaneous data including encodings of fonts, sizes of fonts, list of charset definitions, and coding system definitions.

3 M-text: New Way to Represent Characters

Legacy C string consists only of character codes. Character codes, however, do not contain all the information required for processing text. A character code does not specify its usage; in what language the character is used, how and with what font or glyph the character should be displayed, etc. In order to represent and convey such information in addition to character codes, application programs often define ad hoc data structures and use function parameters.

In the m17n library, objects called *M-texts* represent multilingual text. An M-text can handle mixture of characters of various scripts, including all the Unicode characters. This is an indispensable facility when handling multilingual text. M-text is designed to substitute for string in C, and we keep the parallelism described in Section 2 between M-texts and C strings, that is, the m17n library provides many functions to manipulate M-texts just in the same way to manipulate C strings.

An M-text has not only a character string but also arbitrary number of attributes called *text properties*. A text property is attached to a part of an M-text and used to represent various information such as language, script, etc. on that part.

A text property consists of a *key* and *values*. *Key* specifies what kind of information is stored as the text property. Many text properties that are indispensable for multilingual processing are predefined as a part of the m17n library. Also, application programmers can create their own text properties in their applications. Predefined properties include *face*, which controls appearance of M-text, and *language*, which specifies the language in which M-text is written.

Figure 3 shows how text properties work with a character string. In this example, a text property whose key is ‘face’ is used. This text property controls the appearance of the character string to which it is attached. A text property can have multiple values, and in this example its values are bold, italic, small, and large. Using those values, the m17n library displays the character string in the specified complex appearance.

As an M-text provides a universal framework, different application programs or routines can share the same information coded as text properties. Moreover, as rich information can be stored in M-texts, functions in application programs can be simple and straightforward.

Text properties can be used both to give information to and to receive information from functions/routines. For example, a text generation function passes a text property to a display function to impose constraints on displaying the string, and the display function returns another text property to notify its caller how well the constraints were satisfied. The m17n library can serialize M-text in an XML format so that other application programs can reuse the generated M-texts.

4 Input Method

An input method is a facility for inputing characters that do not have corresponding keys on the keyboard. When required, the m17n library dynamically loads the necessary input method from the m17n database. An input method converts each key sequence typed by the user into a character or a character sequence.

There are several types of input methods.

Keyboard mapping Each key on the keyboard produces a character different from the keytop.

Key sequence A key sequence produces a character. For example, an input method for an European language would generates á (a-acute) when `a` and `'` (apostrophe) are typed in this order.

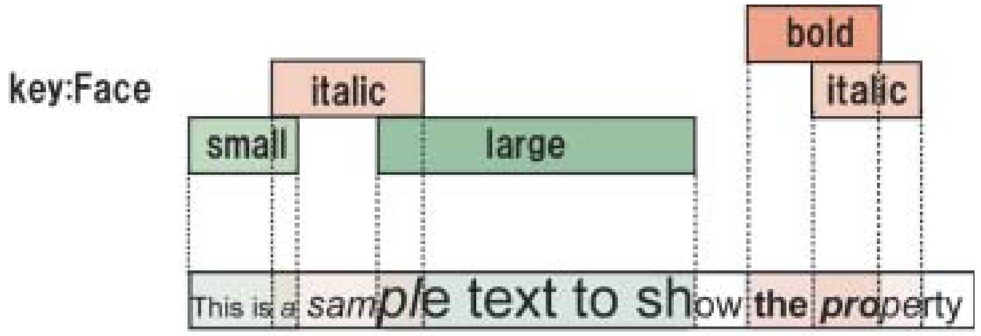

Figure 3: How text properties work

Keyboard mapping + key sequence Combination of the above two.

Transliteration The script to be input has a mapping to Latin script. Users input Latin letters and they are converted into the characters of the target script. Examples include *Roma-ji* for Japanese and Itrans for Indic scripts.

Any of the above + conversion server Usually used for Han characters. For example, users first input *kana* using transliteration and then converts the *kana* into *kanji* (Han) using a conversion server.

In the framework of the m17n library, all the input methods other than conversion servers are described in the m17n database in the form of rules that define state transitions. Rules of a conversion server define how data should be transfered between the m17n library and a loadable module that communicates with the conversion server.

Here is an example of keyboard mapping for Russian. Its rules are written as follows. (Only an excerpt is shown.)

```
(map
 (keyboard
  ("q" ?я) ("w" ?в) ("e" ?е)
  ("r" ?р) ("t" ?т) ("y" ?ы)
  ...))

(state
 (init
   (keyboard)))
```

A “state” block defines which rule groups are used in each state and to which state the input system transits. In the example above, there is only one state (“init”) where the “keyboard” rule group is used.

A “map” block defines rule groups. Again, there is only one rule group (“keyboard”) in the example above. In this rule group, each rule associates a keytop and a Cyrillic character. When a key is pressed, the m17n library finds the corresponding rule and inputs the associated Cyrillic character.

In the case of Devanagari input using transliteration, the input system is a little bit more complex, because the same key (e.g. ‘a’) should be handled differently depending on the context. Here is an excerpt of the rules.

```
(map
 (consonant
  ("k" "क्") ("kh" "ख्")
  ("g" "ग्") ...)
 (independent
  ("a" "अ") ("aa" "आ")
  ("i" "इ") ...)
 (dependent
  ("a" (delete @-))
  ("aa" (delete @-) "ा")
  ("i" (delete @-) "ि")
  ...))

(state
 (init
  (consonant (shift second))
  (independent))

 (second
  (consonant)
  (dependent (shift init))))
```

In this case, there are two states, namely “init” and “second”. Each state has rule groups to convert input sequences into characters. Let us see how the input sequence `kkaa` is handled. The first `k` is processed in the “init” state where two rule groups (“consonant” and “independent”) are used. As `k` appears in a rule in the “consonant” group, the m17n library produces a Devanagari consonant K (U+0915) and a Halant (U+094D), then goes to the “second” state before processing the rest of the input sequence.

In the “second” state, the “consonant” rule group and the “dependent” rule group are used. The second `k` in the input sequence produces, like before, a consonant K (U+0915) and a Halant (U+094D). So far, the generated character sequence is U+0915 U+094D U+0915 U+094D. This time no transition occurs and the next inputs `aa` are processed in this “second” state. According to the second rule in the “dependent” group, `aa` first deletes the last character U+094D (Halant) and produces a new character, Devanagari vowel AA (U+093E). Now the generated character sequence becomes U+0915 U+094D U+0915 U+093E.

5 Display Engine

Displaying, or rendering multilingual text is one of the most important facilities of the m17n library. Many Asian scripts such as Thai and Devanagari require very complex processing for rendering. In such scripts, a sequence of characters may have to be drawn as a single ligature glyph, or glyphs may have to be drawn at 2-dimensionally shifted positions. Furthermore, glyphs may require reordering. Technology for such rendering is known by the name “Complex Text Layout” (CTL for short).

CTL typically includes the following five steps: 1. clustering, 2. reordering, 3. character to glyph mapping, 4. glyph substitution, and 5. glyph positioning. In each step of CTL, the knowledge specific to each script is required. The problem is where such knowledge should be kept.

To date, CTL has been implemented in various fashions. All of them put the necessary knowledge in the renderer, in the fonts, or both of them. However, neither renderer nor font is a suitable place to keep such knowledge. Embedding knowledge in the renderer leads to inflexibility, and embedding in fonts leads to duplication of the same knowledge.

5.1 Font Layout Table

In the m17n library, CTL related knowledge is stored in Font Layout Tables. A Font Layout Table (FLT for short) is a resource that provides

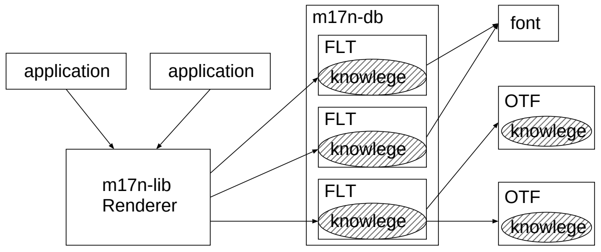

Figure 4: Font Layout Table and CTL knowledge

knowledge necessary for rendering. FLTs bridges fonts and the m17n renderer. A single FLT can be used to drive multiple fonts, and multiple FLTs can drive the same font for multiple ways of writing text (e.g. ancient style ligatures vs. modern style ligatures). Figure 4 illustrates the rendering system in the m17n library using FLT.

Like input methods, FLTs are stored in the m17n database and loaded into the m17n library on demand. An FLT consists of set of rules. Each rule specifies a conversion from a character/glyph sequence into another glyph sequence. The conversion process can be cascaded, i.e. the input character sequence can take intermediate states before converted into the final glyph sequence.

In each stage of conversion, a sequence of characters or glyphs is converted into another glyph sequence using rules specific to the stage. The resulting new sequence is passed further to the next stage. Figure 5 shows a sample rendering performed by an FLT. This FLT has two stages in order to display the word “HINDI” properly in the Devanagari script. In the first stage, the order of the input character sequence is changed from logical order (the order in the computer memory) into visual order (the order on the display or paper). Then in the second stage, the glyphs are substituted depending on the adjacent glyphs, and their display positions are adjusted.

As described in Section 2, data in the m17n database can be added by users. FLT is no exception. By splitting complex conversion into multiple stages, users can perform CTL themselves with ease even for the scripts that are currently unsupported.

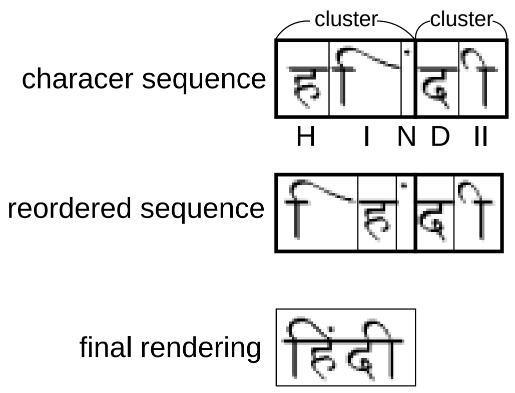

Figure 5: Rendering by multiple stages: “HINDI”

6 Discussion

6.1 Related works

Besides the m17n library, there are several systems for handling multilingual text on Unix/Linux systems. For example, the Pango project [2] has been developing software to layout and render multilingual text. IIIMF (Internet/Intranet Input Method Framework) [3] provides a framework for developing input methods for various languages. GNU libiconv [4] is a library to convert character encodings from one to another. Unlike the m17n library, however, these are single-purpose software and each concerns only one aspect of multilingual text processing: display, input, or code conversion.

The ICU (International Components for Unicode) libraries [5] support, like the m17n library, various aspects of multilingual text processing. While the main purpose of the m17n library is to help developing multilingual user interface, ICU is more suitable to process multilingual text accumulated in the computer. For example, ICU does not provide interactive keyboard input, but supports non-interactive transliteration between two different scripts. On the other hand, the m17n library provides interactive keyboard input, but does not support non-interactive transliteration.

[2]http://www.pango.org/
[3]http://www.openi18n.org/subgroups/im/IIIMF/index.html
[4]http://www.gnu.org/software/libiconv/
[5]http://oss.software.ibm.com/icu/

6.2 Contribution to NLP

The m17n library is useful for NLP researchers as well as for software developers. For example, the flexibility of M-texts must be convenient to represent annotated text. By representing annotations as text properties, it is possible to attach both nested and overlapping annotations anywhere to the text without embedding tags in the original text string.

7 Conclusion

In these days of global communication with computers, multilingualization or localization of application software is indispensable. However, there are many languages that are difficult to support in application programs. We have designed and implemented the m17n library so that all languages are equally well supported and all users, regardless of their language, can benefit from information technology.

So far, we have implemented m17n C library, m17n X library, and the m17n database. We released them under GNU General Public License on March 1st, 2004.

We will develop the third layer of the m17n library, namely m17n X Toolkits. The first target of multilingualization is Gtk+, but other toolkits are also in the scope. Moreover, we are planning to create language bindings (wrapper software) to use the m17n library in other programming languages such as Perl and Ruby.

We also act as a member of the Open Internationalization Initiative, one of the working groups in the Free Standards Group, to promote the m17n library as a global standard.

Acknowledgements

Writing the document of the m17n library is partially funded by Information-technology Promotion Agency (IPA), Japan.

Appendix

The m17n library can be obtained from http://www.m17n.org/m17n-lib under GNU Lesser General Public License (LGPL).