i18n-best-practices Documentation

Release 1

Pravin Satpute

Feb 14, 2017

Contents:

CHAPTER

ONE

INTRODUCTION

Internationalization is not a new concept. From the time computer has invented thoughts were started regarding removing language barriers from it and making it usable for the whole world. Its general phenomenon as more and more people start using particular products, more ideas start getting generating on further better usage of product and further innovation around that product.

In fact same evolution has been happened with the computer and we are witnessing tremendous development around computing.

Few terms:

Internationalization (a.k.a i18n) Development of application in a way to make it adaptable to the requirements of different regionals and languages with minimal codebase change.

Localization (a.k.a l10n) Adapting application to the different regional and language requirement, mostly with translation. This is bit more than translation in some cases i.e. one might need to bring significant cultural changes to application looks, including Date, Time format, localization of images etc.

Globalization (aka g11n) There are numerous definition of globalization in different context. Internationalization and Localization together forms software globalization. This is more kind a strategy for launching i18n & l10n product into new market.

1.1 Need of Internationalization

Out of the entire world population 7+billion, only 341M people have English as their first language. Importance of English in global economies has increased number of fluent English speaker significantly. But one should note that, English is their second language. Major part of world still expect computing in there own language and script.

This is described in more details in talk “Why globalization? World wide picture?”

1.1.1 Why globalizatoin? World wide picture

Add information here from talk.

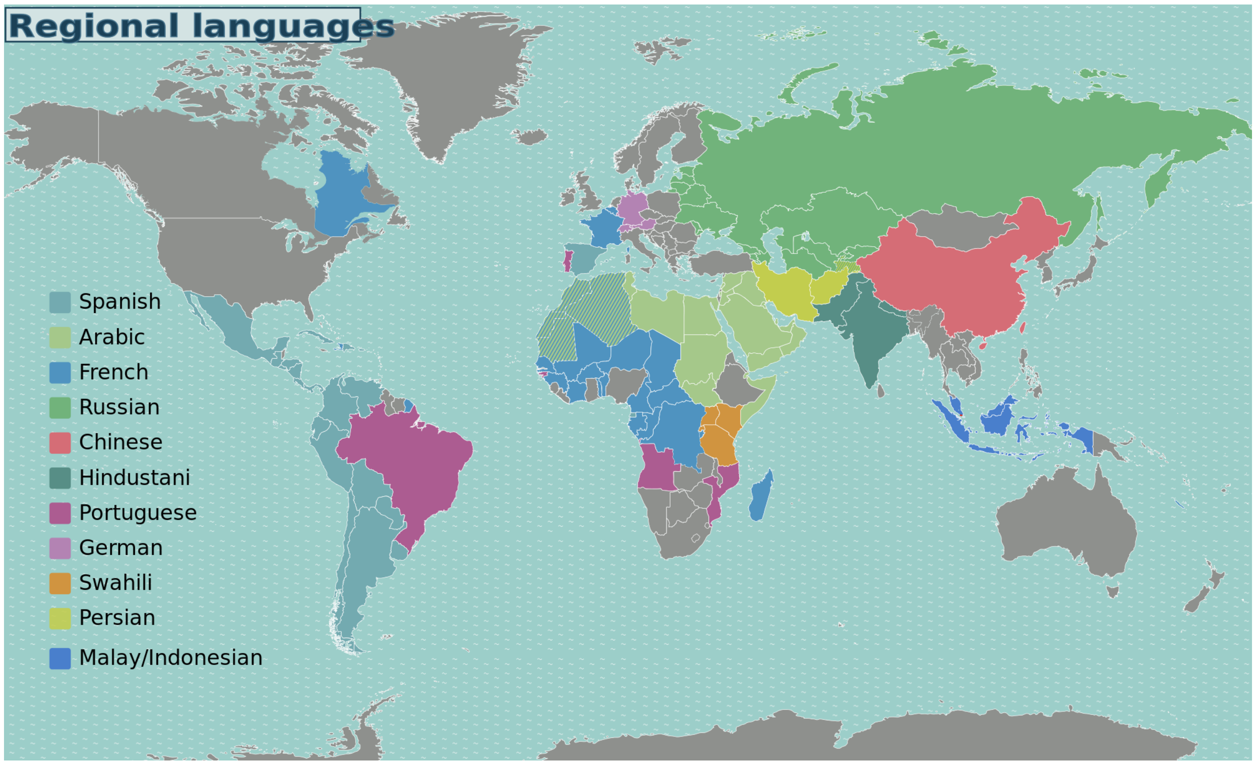


Fig. 1.1: World regional languages

CHAPTER

TWO

BASICS OF INTERNATIONALIZATION

Internationalization can be further divided into two types. It is core internationalization and application internationalization.

Core internationalization is the support from platform for the application getting hosted on it. Platform here means an Operating systems. These support mostly comes in the form of api's. This includes encoding conversions, fonts rendering libraries, locales and input methods.

Application internationalization is more in the form of developing application in a way to use i18n libraries provided by the platform and same time should be adaptable with minimum changes to the requirements of different regional languages.

Please note here Operating system itself is one kind a application and requires its own internationalization.

If one asked things that has helped to language computing world over the year, will say it's standardardization.

- Many countries
- Many languages
-
  - Many languages uses multiple script for writing. Maithili language in India can be written by Devanagari, Tirhuta, Kaithi.
- Many scripts
- Many users
- Few languages are used across the borders.
-
  - Bengali is spoken in India as well in Bangladesh.

Just imagine how it's difficult to get consensus. Thanks to all efforts that has been done by various companies and standardization organization for many year for getting consensus and developing standards suitable for all.

2.1 Encoding

Computer deals with binary !!

Everything we do is get stored in computer in binary. Without standard around encoding we can't do much things.

That is why in 1963 during the start of computing era ASCII (American Standard Code for Information Interchange) was standardized. Earlier ASCII was 7-bit (0-127) later it further extended for 8-bit and few other language characters were added from 128-255.

Main intention behind encoding standard was portability of data across computers.

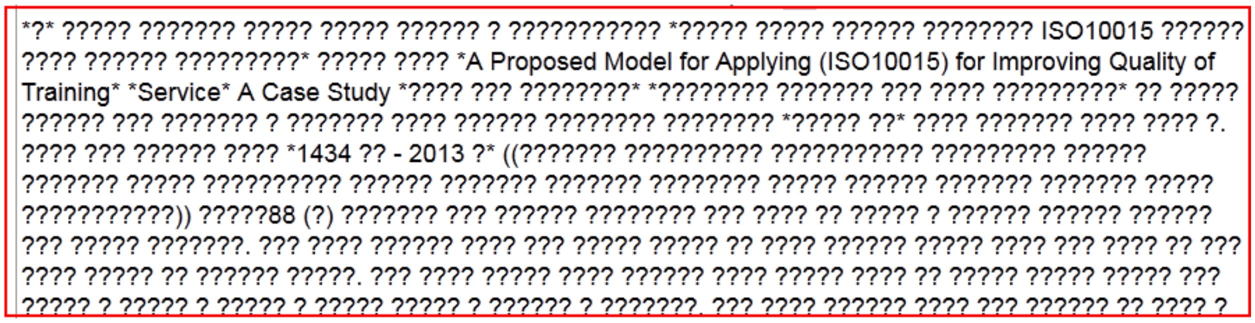
Multiple uses of extended ASCII:

Based on extended ASCII, other countries started creating their own encoding standards to accommodate respective scripts characters. Few of those are

- ISCII (India)
- VISCII (Vietnam)
- YUSCII (Yugoslavia)

All local encoding made sure to keep lower bit of ASCII 0-127 same as with original ASCII standard. Therefor no problem of encoding interoperability with English but with Data from 128-255 all encodings were incompatible.

Have you seen how does encoding issues looks like?



```
*?* ????? ??????? ????? ????? ?????? ? ??????????? *????? ????? ?????? ???????? ISO10015 ??????
???? ?????? ?????????* ????? ???? *A Proposed Model for Applying (ISO10015) for Improving Quality of
Training* *Service* A Case Study *???? ??? ????????* *???????? ??????? ??? ???? ?????????* ?? ?????
?????? ??? ??????? ? ??????? ???? ?????? ???????? ???????? *????? ??* ???? ??????? ???? ???? ?.
???? ??? ?????? ???? *1434 ?? - 2013 ?* ((??????? ?????????? ??????????? ????????? ??????
??????? ????? ?????????? ?????? ??????? ??????? ???????? ????? ?????? ??????? ??????? ?????
???????????)) ?????88 (?) ??????? ??? ?????? ???????? ??? ???? ?? ????? ? ?????? ?????? ??????
??? ????? ???????. ??? ???? ?????? ???? ??? ????? ????? ?? ???? ?????? ????? ???? ??? ???? ?? ???
???? ????? ?? ?????? ?????. ??? ???? ????? ???? ?????? ???? ????? ???? ?? ????? ????? ????? ???
????? ? ????? ? ????? ? ????? ????? ? ?????? ? ???????. ??? ???? ?????? ???? ??? ?????? ?? ???? ?
```

Fig. 2.1: Encoding issues

Above image is nothing but the typical example, where your data is created with different encoding but viewed or wrongly converted to incompatible encoding. Whenever one see this, first thing to check your encoding related stuff.

2.1.1 Unicode and ISO

Incorporated in January, 1991 Unicode Consortium started to work on resolving problem of many encoding standards and incompatibility while transferring data from one computer to other.

Unicode started with simple mission

Unicode provides a unique number for every character,

no matter what the platform,

no matter what the program,

no matter what the language.

Unicode is huge each block of 16-bits and total 21 blocks. Total 21-bit. Most of the active scripts are covered in its first block i.e. 0000-FFFF

In the initial version Unicode identified all the encodings in the world and started providing different codes to it. Though It took long time but now Unicode is accepted globally.

During the same time ISO also started to standardize encoding for different encoding systems. They came up with ISO 10464. Presently both Unicode and ISO 10464 are in sync.

Unicode is huge resource for the world scripts knowledge. To get glimpse one should look into Unicode character database.

The most commonly used encodings of Unicode are:

1 UTF-32

This is 32 bit format. (UTF stands for Unicode Transformation Format). A character is represented as a 32 bit integer. This is the only UTF encoding which is fixed length, UTF-16 and UTF-8 are both variable length.

In UTF-32, each 32 bit value represents one code point and is exactly equal to that code point's numerical value.

2 UTF-16 This is a variable length encoding.

A character is represented as one or two 16 bit integers.

3 UTF-8

This is more interesting to understand. UTF-16 to UTF-8 conversion happens in following way.

Insert Table

- Important point to note here, UTF-8 is backward compatible with ASCII. UTF-8 take same storage space as ASCII for English characters 1 Byte.
- This is more expensive for Asian script specifically in the range U+0800- U+FFFF since it takes 2 to 3 Bytes.
- One always wonders why to use multi-byte encoding when we can use either UTF-16 or UTF-32 directly but UTF16/UTF32 are easy for computation but are not efficient for storage.
- UTF-8 is a more popular encoding of Unicode and widely used than ASCII.

Byte order mark

This is special character in Unicode. U+FEFF. This character is special and provide following information

- What is the byte order, or endianness, the text stream is stored in;
- Confirmation that string is Unicode.
- Whether it is UTF8/UTF16 text?
- - UTF8 - 0xEF,0xBB,0xBF
- - UTF16 (BE) - 0xFE 0xFF

2.1.2 Other Encodings

Though Unicode has became de-facto standard still some countries have mandated use of there custom standards. Following are the some list.

SJIS

Shift Japanese Industrial Standards

2.2 Fonts

How does your computer or any other digital device will looks without fonts?

Most of the platform's by default provide fonts for most of the actively used Unicode scripts. Mostly provides fonts for Basic Multilingual plane of Unicode (0000-FFFF). Still if anytime yousee characters like this or empty/cross square boxes instead of character, make sure you have fonts available for that script.

Basic Multilingual plane: This is first 0000-FFFF range of Unicode and it is used for actively used scripts and languages.

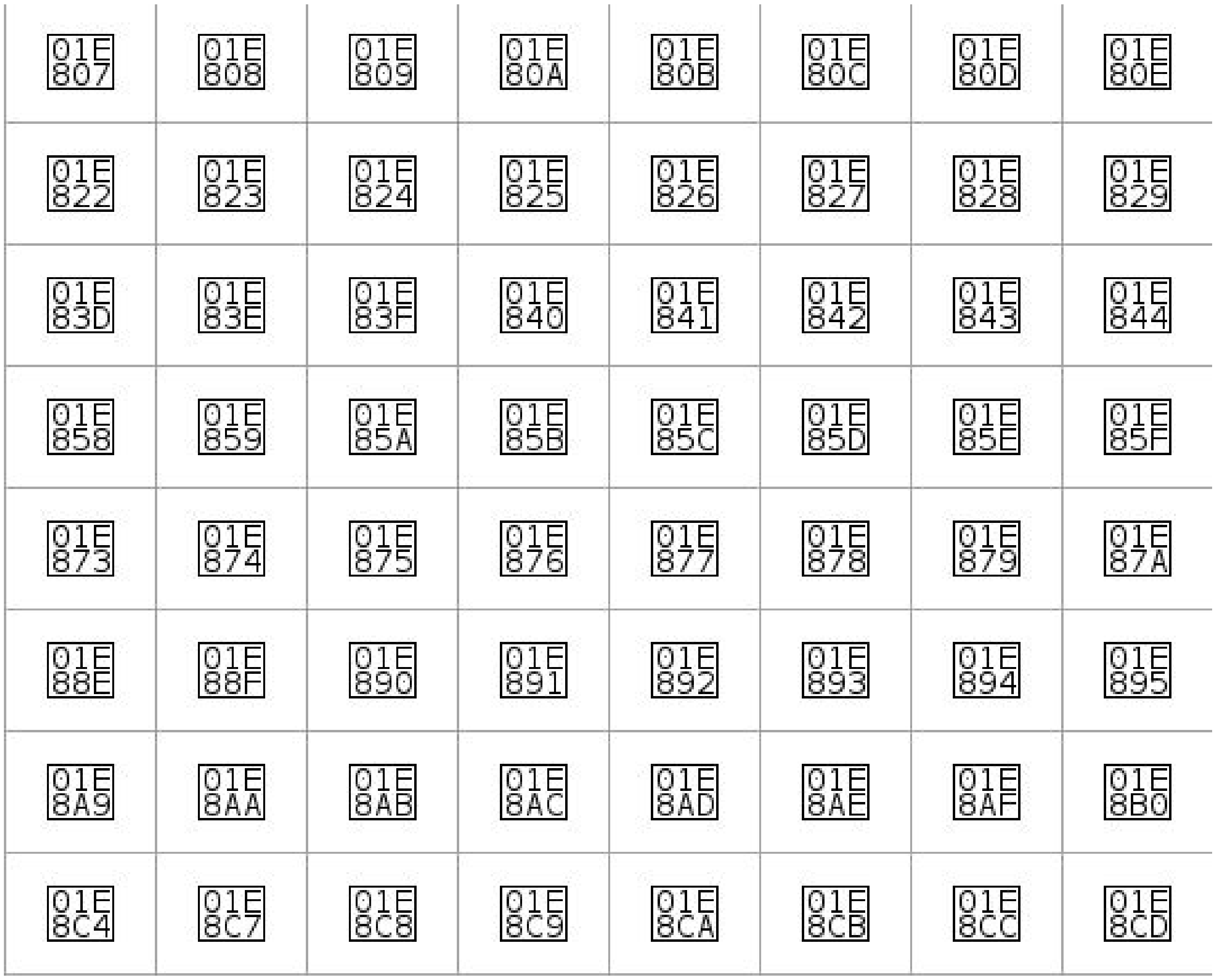

01E807 01E808 01E809 01E80A 01E80B 01E80C 01E80D 01E80E
01E822 01E823 01E824 01E825 01E826 01E827 01E828 01E829
01E83D 01E83E 01E83F 01E840 01E841 01E842 01E843 01E844
01E858 01E859 01E85A 01E85B 01E85C 01E85D 01E85E 01E85F
01E873 01E874 01E875 01E876 01E877 01E878 01E879 01E87A
01E88E 01E88F 01E890 01E891 01E892 01E893 01E894 01E895
01E8A9 01E8AA 01E8AB 01E8AC 01E8AD 01E8AE 01E8AF 01E8B0
01E8C4 01E8C7 01E8C8 01E8C9 01E8CA 01E8CB 01E8CC 01E8CD

Fig. 2.2: Text without fonts

2.2.1 Open Type Fonts

To handle these complexities Adobe and Microsoft together developed OpenType fonts format. This format provide more scope to font for embedding script sensitive data into it.

Open type fonts has more data rather than simple mapping of character image to code. This data includes substitution tables and positioning table. Rendering engine process GSUB and GPOS table provide final output onto screen. In rendering engine there is specific component called OpenType layout shaper (OTLS) which does this magic.

2.2.2 Web Fonts

Multilingual websites were dependant on target platform for expected rendering of the websites. It was big issue and many times websites were not rendered perfectly and end-user need to download and install fonts.

Websites providing multilingual contents became more intelligent with the invention of Web

Fonts. When browsing website using Web fonts technologies fonts get loaded into your cache and you see text using download font rather than default fonts in system. As per platform we have different formats for Web fonts.

EOT

- Microsoft developed .eot (Embedded Open Type) for uses with internet explorer. Once can create this font with Web Embedding Fonts Tool (WEFT).

WOFF

- This is more popular. WOFF is Web open font format. Initial drafted by Mozilla Foundation, Opera Software and Microsoft. WOFF has been also recommended standard by W3C. This can be created from generated from wide variety of fonts editors and also utilities.
- One of the best thing from WOFF, it loads only characters required for rendering from font and save bandwidth.

Web fonts as a Brand

Earlier intention of Web fonts were to remove TOFU from the Websites. In present generation with the technology improvements and more knowledge around fonts many web designers using it for uniquely styling Websites.

We do have Brand fonts now a days and those get used as a Web Fonts to provide unique styling for home web page.

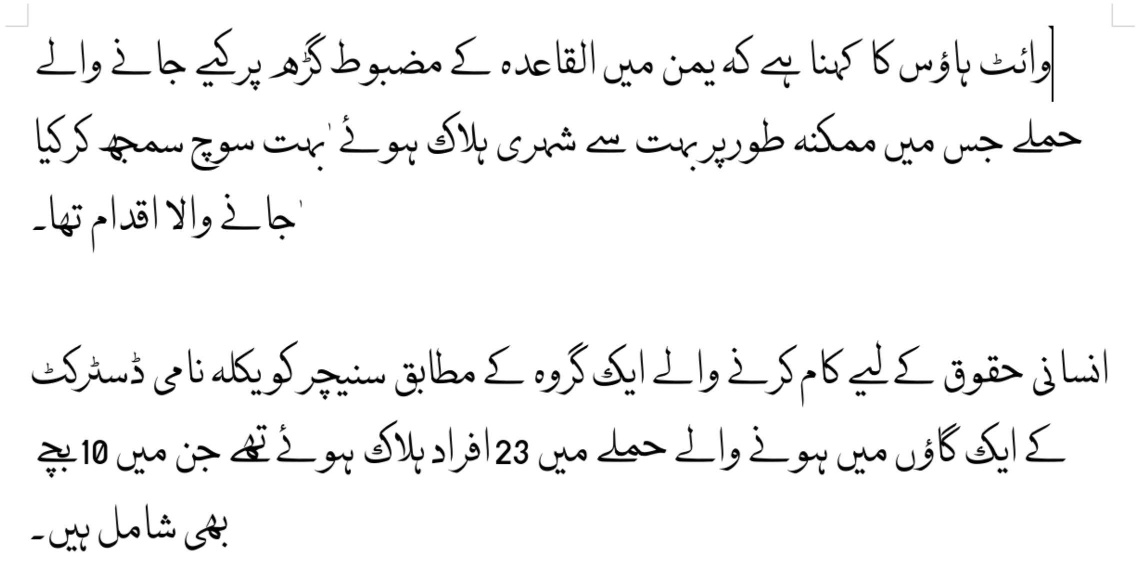
2.3 Rendering Engines

For Latin script it is what you type that you see on screen. This case is not same for complex script like Indic, Arabic and many more.

Example In Indic scripts often,

- Typing 2-3 characters give you single character. – (u+0915) + (u+094D) + (u+0937) =
- Even characters get reordered. (u+0915) + (u+093F) =
- Single characters get split into multiple part. (u+0D15) + (u+0D4A) =

Arabic script brings different type of complexity. First it is written from Right to Left (RTL) and second it is cursive in nature. Furhter it can be Naskh and Nastaliq. Naskh is linear cursive and Nastaliq is (Add Text)



وائٹ ہاؤس کا کہنا ہے کہ یمن میں القاعدہ کے مضبوط گڑھ پر کیے جانے والے
حملے جس میں ممکنہ طور پر بہت سے شہری ہلاک ہوئے 'بہت سوچ سمجھ کر کیا
جانے والا اقدام تھا۔'

انسانی حقوق کے لیے کام کرنے والے ایک گروہ کے مطابق سنیچر کو یکلہ نامی ڈسٹرکٹ
کے ایک گاؤں میں ہونے والے حملے میں 23 افراد ہلاک ہوئے تھے جن میں 10 بچے
بھی شامل ہیں۔

Fig. 2.3: Arabic script Naskh

2.3.1 Uniscribe

In Windows Uniscribe is OTLS and actual file name is usp10.dll. It has been treated as a reference implementation of Open Type specs. Though during the experience different versions of usp10.dll in same Operating system often caused different rendering experience in applications.

Example: Different usp10.dll in MS-Office and Windows OS.

2.3.2 Harfbuzz

In Linux harfbuzz has became de facto standard. Harfbuzz is compatible with Uniscribe and make sure to provide same processing to fonts as it does with Uniscribe. Compatibility between Uniscribe and Harfbuzz has made it possible to increase fonts portability across multiple platform. Harfbuzz is under more permissive Free and open source Apache license and any product can use it without restrictions.

Prior Harfbuzz, each rendering engine example, Pango, ICU and QT were having own codebase for OTLS, over the time it created big incompatiblity issues across rendering engines for processing of fonts. Fonts were not providing same results across rendering engines and platforms. Harfbuzz has significantly helped to make it uniform now.

2.3.3 ATSUI

Apply has different complex script processing mechanism with ATSUI (Apple Type Services for Unicode Imaging) library. ATSUI mechanism has provided more scope to make fonts more intelligent by embedding Finite state machines (FSM) into fonts. It create less dependency on rendering engine and more freedom of design and programming fonts as per required for typographers.

2.4 Locales

What is locale?

Locale helps program to get into the environment required for particular countries and cultures. We have many different conventions need to get changed while running program into particular countries and culture. Like..

- Date and Time format
- Numeric format
- Data Sorting
- Expression of Yes/No. etc.

Locale provide this crucial data to program. During earlier days every OS has its own locale data. Like Glibc has its native Locale data, Microsoft had there own and so on.

But with the start of Unicode CLDR data, most of the partner has been migrated to it.

2.4.1 Unicode CLDR

The term CLDR stands for common locale data repository and it provides key building blocks to support world languages in software. After first release in 2003 in present stage Unicode CLDR has became huge repository for World languages and its growing and growing. Unicode CLDR has been already used and supported by major software companies including Apple, Google, IBM and Microsoft.

CLDR is collaborative efforts of different domain experts. Various countries governments are part of it and many fields get finalized based on votes of members.

Unicode CLDR is even much bigger and has more information for each locales.

2.4.2 Locale Code

Typically locales are available in langcode_countrycode format. (en_US).

Why is so combination?

Many languages are spoken in different countries. Though the languages are same few things are different from country perspective. Like different sorting, different Date/Time format and many more. To handle this differences it is important to have different locale as per lang and country code.

Example:

- bn_IN and bn_BD Bengali language locale for India and Bangladesh.

Language codes are standardized under ISO 639-1 standard. Earlier 2 digit codes were assigned to each language but since 2 digits were insufficient to provide code for all languages in ISO 639-2 languages are represented with 3 digit code.

The rule used while adding locale in glibc is - if language does not have 2 digit code then create locale with 3 digit code.

2.4.3 Currency format

Few countries using "," as a separator while other are using either "." or simply space.

- US and India Format - 1,234,567.89

- German Format - 1.234.567,89
- French Format - 1 234 567,89

2.4.4 Date/Time format

Its depends how you write date. Whether yyyy-mm-dd format or start with date first then month and then year. Dd-mm-yyyy.

- India - dd-mm-yyyy
- USA - yyyy-mm-dd

2.4.5 Collation

Sorting has been remained an interesting topic of research over the time. Sorting varies across countries.

- Few regions like to Caps first then small. A-Z followed by a-z
- Few regions like SMALL and CAPS letter together. Aa Bb etc.

Sorting affect significantly day to day work. Everywhere data represented in sorted order.

2.5 Input methods

Unlike English language, For many languages there any many characters than the number of keys available on a keyboard. To get all these characters of particular language, one need to use different key combination of modifiers to get those characters.

Example:

- As of Unicode 9.0, Unicode defines a total of 80,388 CJK Unified Ideographs.
- In Devanagari script - 256 characters are presently

To get required characters from keyboard, software need add-on intelligence. Input methods provide drivers and keymaps to get characters for these complex scripts.

Input methods also available from platform and also over the year many input methods available on web. Website developers sometime uses online Input method in their applications to avoid dependencies on platforms.

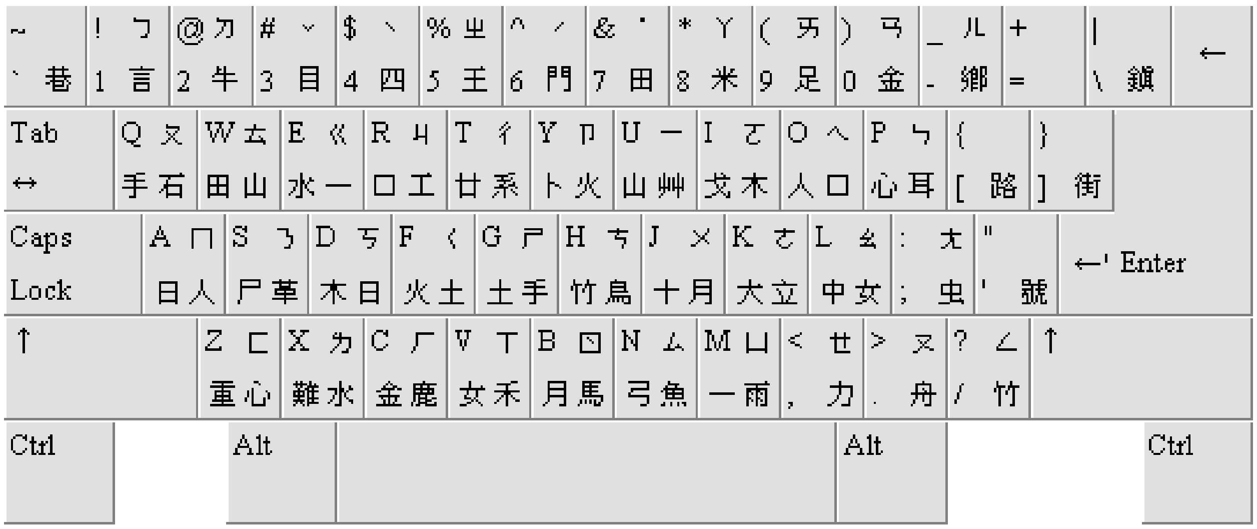


Fig. 2.4: Keyboard layout Chinese traditional

CHAPTER

THREE

IMPORTANT POINTS FOR I18N

When developing internationalized application developer need to consider two important aspects as following:

1 Application should accept system's locale or one provided by user and change its behaviour accordingly.
Example: - **Website**: If you are developing website and end user is viewing it in browser running in Japanese locale. Website dates, time and currency format should get changed into Japanese locale.

- **Desktop application**: It's behaviour should change based on locale format of desktop.

2 Applications UI should get change according to system locale.

This is important aspect of internationalized application. Converting whole UI into target language without any major changes into application. Just by selecting appropriate system locale or browser locale. We will see this with example in coming few sections how can we achieve it.

3.1 Core i18n support on platform

Target language must have been enabled on target platform. If target language is not available on platform.

1 User might not get required locales.

2 Fonts required for rendering target script might be not available. (With Webfonts this can be handled for Websites as discussed in Wen fonts section)

3.2 Application should accept system locale

Non internationalized application starts in system's default locale irrespective of active locale. Like in C program its "C" locale. To get application use system locale or one which user expecting program must be initialized by setting locale. We will see how to do it with different programming language in coming sections.

3.3 Strings and Code separation

One of the important goal of internationalization is minimal code changes while moving products in between different languages. Before technology getting matured developer were maintaining different code bases for different language. I.e. Build for Japanese, English etc.

But this approach was not realistic for supporting many languages and also it was not dynamic.

Also other important issue comes, when we design buttons of application. Translating few words from one language to another results in translated word either bit lengthy or tiny. It affects app design drastically and it must be auto adjustable with overall design.

Example: English equivalent of Jet Black is Rabenschwarz

Separating code and string has been became much simpler now a days with the introductions of a frameworks like gettext.

Framework like gettext not only help to extract string elements from codebase but also help to select exact language for replacement of strings depending upon locale.

CHAPTER

FOUR

L10N STRING EXTRACTION FRAMEWORKS

4.1 Gettext

Prior to gettext framework, handling of translations related activities i.e. Extractions of string, replacing translated string with proper localized string during runtime was mostly handled by program itself.

It was adding one different complexity of writing internationalized program. In fact many applications were shipped for specific languages only. User need to download exact program for his language.

4.1.1 Concepts

As Gettext documentation says

"GNU gettext is designed to minimize the impact of internationalization on program sources, keeping this impact as small and hardly noticeable as possible. Internationalization has better chances of succeeding if it is very light weighted, or at least, appear to be so, when looking at program sources."

Gettext provides

- A set of conventions about how programs should be written to support message catalogs.
- A directory and file naming organization for the message catalogs themselves.
- A runtime library supporting the retrieval of translated messages.
- A few stand-alone programs to massage in various ways the sets of translatable strings, or already translated strings.
- A library supporting the parsing and creation of files containing translated messages.
- A special mode for Emacs1 which helps preparing these sets and bringing them up to date.

Implementation

Following diagram provides list of all the utilities which used throughout implementation of gettext.

4.1.2 Extraction

As information given in above diagram. Modifying source code as per requirement and guidelines given by gettext is first important change required from the developer.

Once marking source strings for translations is done, then gettext has provided number of tools which can be simply automate using Makefile. As shown in above block diagram

- **xgettext** - Extract translatable strings from given input files.

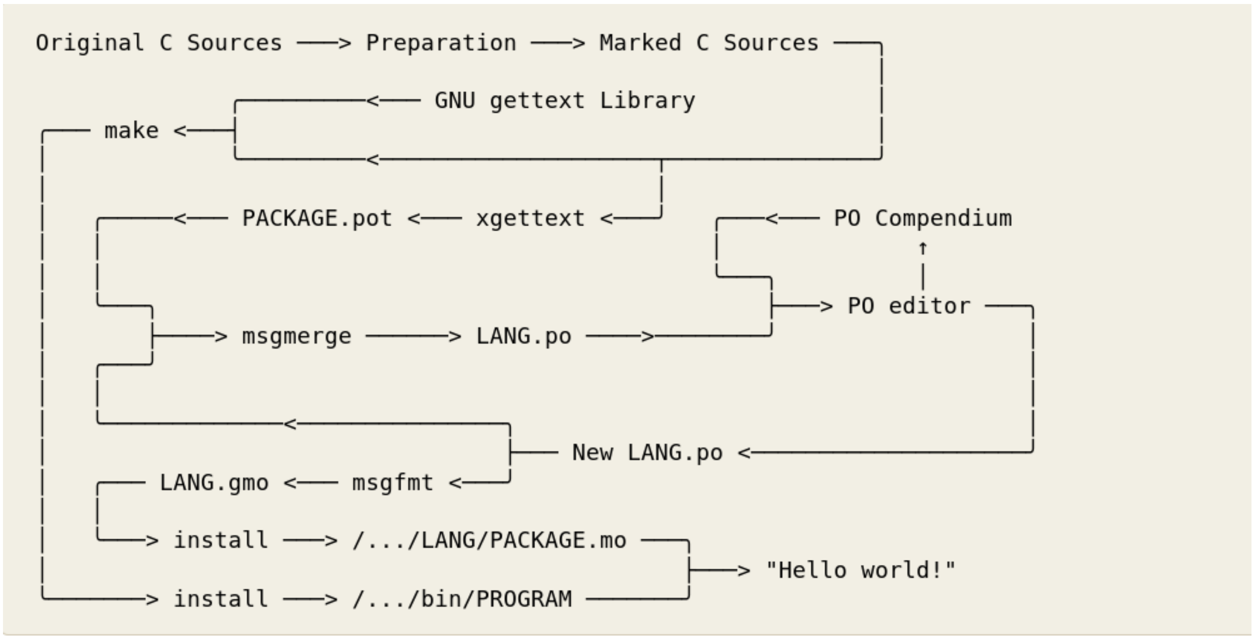


Fig. 4.1: Overview of gettext utilities

- **msgmerge** - For refreshing already generated .pot file without data loss.
- **msgfmt** - Generate binary message catalog from textual translation description.

In example of internationalization of Hello World i18n gettext has been used and it provides practical example of its implementation.

4.1.3 Installation of Gettext

Gettext is available across platforms.

Fedora/Red Hat/Cent OS

```
[user@host]$ sudo yum install gettext
```

Debian/Ubuntu

```
[user@host]$ sudo apt-get install gettext
```

Windows

Precompiled binaries are available for Windows at https://mlocati.github.io/articles/gettext-iconv-windows.html

Available in Shared and Static flavors. Where static means setup size is much bigger, but all the executables may be moved around as you like, no DLL-dependencies. Shared is smaller in size but all DLL must stay together.

MacOS

- Open Terminal
- Run:

```
$ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/
↪install)" -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)" < /dev/null 2> /dev/null

$brew install gettext
```

Detailed information available at http://macappstore.org/gettext/

4.2 XLIFF

XML Localisation Interchange File Format. http://docs.oasis-open.org/xliff/xliff-core/v2.0/os/xliff-core-v2.0-os.html

"XLIFF is the XML Localisation Interchange File Format designed by a group of multilingual content publishers, software providers, localization service providers, localization tools providers and researchers. It is intended to give any multilingual content owner a single interchange file format that can be understood by any localization provider, using any conformant localization tool. While the primary focus is on being a lossless interchange format, usage of XLIFF as a processing format is neither encouraged nor discouraged or prohibited."

4.3 L20N

Stands for "L10N 2.0" developed by Mozilla. Its main core is users should be able to benefit from the entire expressive power of the natural language and keeps simple things simple for developer and at the same time makes complex things possible. Presently in used by Mozilla.

4.4 Qt translation tools

This is bit similar to gettext but provide good utilities but this is only limited to QT framework does not work outside.

4.5 Java properties

This is .properties file extension also know as Property Resource Bundles. These files used to store strings for localizations.

4.6 Windows resources

.Net and various Windows technologies use concept of resource file for localization of application.

CHAPTER

FIVE

I18N OF C APPLICATION

In this section we will see what are important points developers must consider and include during the development of C programming applications.

5.1 Data types

As discussed in earlier sections Unicode is utmost important element of i18n. Your application must be capable of processing and saving Unicode text.

- **char** : This is 8-bit Data type and this can be used for UTF-8.
- **wchar_t** : Here w stands for wide char. This datatype was introduced to support Unicode but its size is dependant on compiler. To avoid this confusion now we have char16_t and char32_t.
- **char16_t** : This is for 16-bit code units. Unicode basic multilingual plane i.e. from u+0000 - u+FFFF can be represented by this data type.
- **char32_t** : This is for 32-bit code units. Unicode is of 21-bit, so all Unicode data can be represented by this datatype.

As discussed in Encoding sections thought UTF16 and UTF32 are easy for computation from storage efficiency perspective it's good to use UTF-8.

Tip: Use UTF-8 as much as possible and convert it to either UTF16 or UTF32 only when need for computation or string operations like uppercase kind conversions.

5.2 Libraries

Gettext is widely used libraries. One can also use other libraries as per requirement or choice. For Gettext libraries see section about L10N string extraction frameworks.

5.3 I18N of "Hello World"

5.3.1 Non i18n hellworld.c

Most of the developer already familiar with Helloworld.c program.

```
#include<stdio.h>

int main()
{
  printf("Hello World\n");
  return 0;
}
```

5.3.2 Internationalized helloworld.c

Now lets see, How to internationalized above Helloworld.c with Gettext framework.

```
#include<libintl.h>
#include<locale.h>
#include<stdio.h>
#define _(String) gettext (String)

int main()
{
  setlocale(LC_ALL,"");
  bindtextdomain("helloworld","/usr/share/locale");
  textdomain("helloworld");
  printf(_("Hello World\n"));
  return 0;
}
```

Explainations

libintl.h

This header for defines the macros __USE_GNU_GETTEXT, __GNU_GETTEXT_SUPPORTED_REVISION, and declares the functions gettext, dgettext, dcgettext, ngettext, dngettext, dcngettext, textdomain, bindtextdomain, bind_textdomain_codeset.

Gnulib module: gettext provides this header file.

locale.h

This header file provided the macros required for location specific information. This includes LC_ALL, LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC, LC_TIME

Provides functions like localeconv and setlocale.

#define _(String) gettext (String)

For code readability this is usual practice of defining #define and calling gettext function just by adding "_" before string.

setlocale(LC_ALL,””);

As per ISO C all programs start by default in the standard ‘C’ locale. To use the locales specified by the environment, one has to call setlocale. Once you call above function locale environment variables will be set to your system or present locale.

More information about in in Locales section.

Bindtextdomain

This function can be used to specify the directory which contains the message catalogs for domain domainname for the different languages. Good to provide absolute path to avoid confusion.

textdomain(“helloworld”)

The textdomain function sets the default domain, which is used in all future gettext calls, to domainname. Before the first call to textdomain the default domain is messages.

printf(_(“Hello Worldn”));

Watch “_” here, we are calling gettext function here.

This is your internationalized program capable of extracting string for translation and substitute as per your locale. Now in next section see how to extract string, translate those and actually get translation while running application.

5.3.3 Preparing PO file

From your program folder run following commands

```
$mkdir po

$xgettext -d helloworld -o po/helloworld.pot -k_ -s helloworld.c
```

- xgettext - extract gettext strings from source.
- ‘-d’ domainname, ‘-o’ output filename, ‘-s’ generate sorted output.

```
$cat po/helloworld.pot

# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE
package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2017-01-13 15:18+0530\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
```

```
"Language-Team: LANGUAGE <LL@li.org>\n"
"Language: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"
#: i18nhello.c:12
#, c-format
msgid "Hello World\n"
msgstr ""
```

- helloworld.pot is the Portable Object Template (.pot) file developer suppose to share with translators.

5.3.4 Translations

Just rename .pot file to respective language code and provide it for translations.

Example

- For Hindi language - hi.po
- For particular language spoken in multiple countries. Like Bengali (Spoken in India as well in Bangladesh) - hi_IN.po or hi_BD.po
- For languages with particular encoding. - zh_CN_GB2312.po

Translators need to update header of PO file as per required and add string for particular language as a substitute.

Example : In above case

```
msgid "Hello World\n"
msgstr ""
```

Here for Hindi

```
msgid "Hello World\n"
msgstr "Tt tdflkvsnfv "
```

5.3.5 Execution of program

Creating .mo file from .po file

MO - Machine Object file

Mo file is compiled binary version of .po file and it is more efficient for processing by computers.

```
$msgfmt helloworld.po -o helloworld.mo
```

Msgfmt is program to compile message catalog to binary format.

Placing .mo file in folder

In helloworld.c program, we have given path of "/usr/share/locale" in bindtextdomain function.

To make our translations accessible to program during execution with particular locale those must be kept in this location.

Gettext looks for particular locale code under this directory while searching for translation file.

```
$sudo mkdir /usr/share/locale/hi/LC_MESSAGES/
$cp helloworld.mo /usr/share/locale/hi/LC_MESSAGES/
```

Demonstration of application

- Compile helloworld.c, make sure you are in directory of helloworld.c

```
$gcc -o helloworld helloworld.c
```

- Run program

```
$./helloworld
Hello Wolrd
$LANG=hi_IN ./helloworld
```

Automation for i18n

In above helloworld.c internationalization we did many step one by one. In real life programming once .POT file is generated developer need to copy it for many languages and distribute them to translators and after collecting build application accordingly.

Example:

```
$xgettext -d helloworld -o po/helloworld.pot -k_ -s helloworld.c
cp po/helloworld.pot hi.po
# Get translated hi.po from Translator

$msgfmt helloworld.po -o helloworld.mo
cp helloworld.mo /usr/share/locale/hi/LC_MESSAGES/
```

One need to generalize above steps using autotools or Makefile as per requirement of program.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search