


Chapter 8. I18N and L10N

Table of Contents

Multilingualization (M17N) or Native Language Support for an application software is done in 2 steps.

- Internationalization (I18N): To make a software potentially handle multiple locales.
- Localization (L10N): To make a software handle an specific locale.

Tip

There are 17, 18, or 10 letters between "m" and "n", "i" and "n", or "l" and "n" in multilingualization, internationalization, and localization which correspond to M17N, I18N, and L10N.

The modern software such as GNOME and KDE are multilingualized.

They are internationalized by making them handle UTF-8 data and localized by providing their translated messages through the gettext(1) infrastructure. Translated messages may be provided as separate localization packages. They can be selected simply by setting pertinent environment variables to the appropriate locale.

The simplest representation of the text data is **ASCII** which is sufficient for English and uses less than 127 characters (representable with 7 bits). In order to support much more characters for the international support, many character encoding systems have been invented. The modern and sensible encoding system is **UTF-8** which can handle practically all the characters known to the human (see Section 8.3.1, “Basics of encoding”).

See Introduction to i18n for details.

The international hardware support is enabled with localized hardware configuration data.

8.1. The keyboard input

The Debian system can be configured to work with many international keyboard arrangements using the **`keyboard-configuration`** and **`console-setup`** packages.

```
# dpkg-reconfigure keyboard-configuration
# dpkg-reconfigure console-setup
```

This configures the keyboard for the Linux console and the X Window updates configuration parameters in "**`/etc/default/keyboard`**" and "**`/etc/default/console-setup`**". This also configures the Linux console font.

Many non-ASCII characters including accented characters used by many European languages can be made available with dead key, AltGr key, and compose key.

For Asian languages, you need more complicated input method support such as IBus discussed next.

8.1.1. The input method support with IBus

Multilingual input to the application is processed as:

```
Keyboard                                      Application
   |                                               ^
   |                                               |
   +-> Linux kernel -> Input method -> Gtk, Qt, or X
```

Setup of multilingual input for the Debian system is simplified by using the IBus family of packages with the **im-config** package. The list of IBus packages are the following.

Table 8.1. List of input method supports with IBus

package	popcon	size	supported locale
ibus	V:7, I:9	2170	input method framework using dbus
ibus-mozc	V:1, I:1	871	Japanese
ibus-anthy	V:1, I:3	746	, ,
ibus-kkc	V:0, I:0	266	, ,
ibus-skk	V:0, I:0	230	, ,
ibus-pinyin	V:1, I:2	1452	Chinese (for zh_CN)
ibus-chewing	V:0, I:0	325	, , (for zh_TW)
ibus-hangul	V:0, I:1	297	Korean
ibus-table	V:1, I:2	706	table engine for IBus
ibus-table-thai	I:0	143	Thai
ibus-unikey	V:0, I:0	276	Vietnamese
ibus-m17n	V:0, I:1	163	Multilingual: Indic, Arabic and others

The kinput2 method and other locale dependent Asian classic input methods still exist but are not recommended for the modern UTF-8 X environment. The SCIM and uim tool chains are an slightly older approach for the international input method for the modern UTF-8 X environment.

8.1.2. An example for Japanese

I find the Japanese input method started under English environment ("**en_US.UTF-8**") very useful. Here is how I did this with IBus for GNOME3:

1. Install the Japanese input tool package **ibus-anthy** with its recommended packages such as **im-config**.

2. Execute "**im-config**" from user's shell and select "**ibus**" as the input method.
3. Select "Settings" → "Keyboard" → "Input Sources" → click "**+**" in "Input Sources" → "Japanese" → "Japanese (anthy)" and click "Add".
4. Select "Japanese" and click "Add" to support the Japanese layout keyboard without character conversion. (You may chose as many input sources.)
5. Relogin to user's account.
6. Verify setting by "**im-config**".
7. Setup input source by right clicking the GUI toolbar icon.
8. Switch among installed input sources by SUPER-SPACE. (SUPER is normally the Windows key.)

Please note the following.

- im-config(8) behaves differently if command is executed from root or not.
- im-config(8) enables the best input method on the system as default without any user actions.
- The GUI menu entry for im-config(8) is disabled as default to prevent cluttering.

8.1.3. Disabling the input method

If you wish to input without going through XIM (mechanism used by the X), set "**$XMODIFIERS**" value to "none" while starting a program. This may be the case if you use Japanese input infrastructure **egg** on emacs(1) while disabling **ibus**. From shell, execute as the following.

```
$ XMODIFIERS=none emacs
```

In order to adjust the command executed by the Debian menu, place customized configuration in "**/etc/menu/**" following method described in "**/usr/share/doc/menu/html**".

8.2. The display output

Linux console can only display limited characters. (You need to use special terminal program such as jfbterm(1) to display non-European languages on the non-X console.)

X Window can display any characters in the UTF-8 as long as required font data exists. (The encoding of the original font data is taken care by the X Window System and transparent to the user.)

8.3. The locale

The following focuses on the locale for applications run under X Window environment started from gdm3(1).

8.3.1. Basics of encoding

The environment variable "**LANG=xx_YY.ZZZZ**" sets the locale to language code "**xx**", country code "**yy**", and encoding "**ZZZZ**" (see Section 1.5.2, “The "**$LANG**" variable”).

The current Debian system normally sets the locale as "**LANG=xx_YY.UTF-8**". This uses the UTF-8 encoding with the Unicode character set. This UTF-8 encoding system is a multibyte code system and uses code points smartly. The ASCII data, which consist only with 7-bit range codes, are always valid UTF-8 data consisting only with 1 byte per character.

The previous Debian system used to set the locale as "**LANG=C**" or "**LANG=xx_YY**" (without "**.UTF-8**").

- The ASCII character set is used for "**LANG=C**" or "**LANG=POSIX**".
- The traditional encoding system in Unix is used for "**LANG=xx_YY**".

Actual traditional encoding system used for "**LANG=xx_YY**" can be identified by checking "**/usr/share/i18n/SUPPORTED**". For example, "**en_US**" uses "**ISO-8859-1**" encoding and "**fr_FR@euro**" uses "**ISO-8859-15**" encoding.

> **Tip**
>
> For meaning of encoding values, see Table 11.2, “List of encoding values and their usage”.

8.3.2. Rationale for UTF-8 locale

Unicode character set can represent practically all characters known to human with code point range from 0 to 10FFFF in hexadecimal notation. Its storage requires at least 21 bits.

Text encoding system UTF-8 fits Unicode code points into a sensible 8 bit data stream compatible with the ASCII data processing system. **UTF** stands for Unicode Transformation Format.

I recommend to use UTF-8 locale for your desktop, e.g., "**`LANG=en_US.UTF-8`**". The first part of the locale determines messages presented by applications. For example, gedit(1) (text editor for the GNOME Desktop) under "**`LANG=fr_FR.UTF-8`**" locale can display and edit Chinese character text data while presenting menus in French, as long as required fonts and input methods are installed.

I also recommend to set the locale only using the "**`$LANG`**" environment variable. I do not see much benefit of setting a complicated combination of "**`LC_*`**" variables (see locale(1)) under UTF-8 locale.

Even plain English text may contain non-ASCII characters, e.g. slightly curly left and right quotation marks are not available in ASCII.

```
“double quoted text” is not "double quoted ASCII"
‘single quoted text’ is not 'single quoted ASCII'
```

When ASCII plain text data is converted to UTF-8 one, it has exactly the same content and size as the original ASCII one. So you loose nothing by deploying UTF-8 locale.

Some programs consume more memory after supporting I18N. This is because they are coded to use UTF-32(UCS4) internally to support Unicode for speed optimization and consume 4 bytes per each ASCII character data independent of locale selected. Again, you loose nothing by deploying UTF-8 locale.

The vendor specific old non-UTF-8 encoding systems tend to have minor but annoying differences on some characters such as graphic ones for many countries. The deployment of the UTF-8 system by the modern OSs practically solved these conflicting encoding issues.

8.3.3. The reconfiguration of the locale

In order for the system to access a particular locale, the locale data must be compiled from the locale database. (The Debian system does **not** come with all available locales pre-compiled unless you installed

the **locales-all** package.) The full list of supported locales available for compiling is available in "**/usr/share/i18n/SUPPORTED**". This lists all the proper locale names. The following lists all the available UTF-8 locales already compiled to the binary form.

```
$ locale -a | grep utf8
```

The following command execution reconfigures the **locales** package.

```
# dpkg-reconfigure locales
```

This process involves 3 steps.

1. Update the list of available locales
2. Compile them into the binary form
3. Set the system wide default locale value in "**/etc/default/locale**" for use by PAM (see Section 4.5, “PAM and NSS”)

The list of available locale should include "**en_US.UTF-8**" and all the interesting languages with "**UTF-8**".

The recommended default locale is "**en_US.UTF-8**" for US English. For other languages, please make sure to chose locale with "**UTF-8**". Any one of these settings can handle any international characters.

Note

Although setting locale to "**C**" uses US English message, it handles only ASCII characters.

8.3.4. The value of the "$LANG" environment variable

The value of the "**$LANG**" environment variable is set and changed by many applications.

- Set initially by the PAM mechanism of login(1) for the local Linux console programs

- Set initially by the PAM mechanism of the display manager for all X programs
- Set initially by the PAM mechanism of ssh(1) for the remote console programs
- Changed by some display manager such as gdm3(1) for all X programs
- Changed by the X session startup code via "**`~/.xsessionrc`**" for all X programs
- Changed by the shell startup code, e.g. "**`~/.bashrc`**", for all console programs

Tip

It is a good idea to install system wide default locale as "**`en_US.UTF-8`**" for maximum compatibility.

8.3.5. Specific locale only under X Window

You can chose specific locale only under X Window irrespective of your system wide default locale using PAM customization (see Section 4.5, “PAM and NSS”) as follows.

This environment should provide you with your best desktop experience with stability. You have access to the functioning character terminal with readable messages even when the X Window System is not working. This becomes essential for languages which use non-roman characters such as Chinese, Japanese, and Korean.

Note

There may be another way available as the improvement of X session manager package but please read following as the generic and basic method of setting the locale. For gdm3(1), I know you can select the locale of X session via its memu.

The following line defines file location of the language environment in the PAM configuration file, such as "**`/etc/pam.d/gdm3`**.

```
auth    required        pam_env.so read_env=1
envfile=/etc/default/locale
```

Change this to the following.

```
auth    required        pam_env.so read_env=1
envfile=/etc/default/locale-x
```

For Japanese, create a "**/etc/default/locale-x**" file with "**-rw-r--r-- 1 root root**" permission containing the following.

```
LANG="ja_JP.UTF-8"
```

Keep the default "**/etc/default/locale**" file for other programs as the the following.

```
LANG="en_US.UTF-8"
```

This is the most generic technique to customize locale and makes the menu selection dialog of gdm3(1) itself to be localized.

Alternatively for this case, you may simply change locale using the "**~/.xsessionrc**" file.

8.3.6. Filename encoding

For cross platform data exchanges (see Section 10.1.7, “Removable storage device”), you may need to mount some filesystem with particular encodings. For example, mount(8) for vfat filesystem assumes CP437 if used without option. You need to provide explicit mount option to use UTF-8 or CP932 for filenames.

Note

When auto-mounting a hot-pluggable USB memory stick under modern desktop environment such as GNOME, you may provide such mount option by right clicking the icon on the desktop, click "Drive" tab, click to expand "Setting", and entering "utf8" to "Mount options:". The next time this memory stick is mounted, mount with UTF-8 is enabled.

Note

If you are upgrading system or moving disk drives from older non-UTF-8 system, file names with non-ASCII characters may be encoded in the historic and deprecated encodings such as ISO-8859-1 or eucJP. Please seek help of text conversion tools to convert them to UTF-8. See Section 11.1, “Text data conversion tools”.

Samba uses Unicode for newer clients (Windows NT, 200x, XP) but uses CP850 for older clients (DOS and Windows 9x/Me) as default. This default for older clients can be changed using "**dos charset**" in the "**/etc/samba/smb.conf**" file, e.g., to CP932 for Japanese.

8.3.7. Localized messages and translated documentation

Translations exist for many of the text messages and documents that are displayed in the Debian system, such as error messages, standard program output, menus, and manual pages. GNU gettext(1) command tool chain is used as the backend tool for most translation activities.

Under "Tasks" → "Localization" aptitude(8) provides an extensive list of useful binary packages which add localized messages to applications and provide translated documentation.

For example, you can obtain the localized message for manpage by installing the **manpages-<LANG>** package. To read the Italian-language manpage for <programname> from "**/usr/share/man/it/**", execute as the following.

```
LANG=it_IT.UTF-8 man <programname>
```

8.3.8. Effects of the locale

The sort order of characters with sort(1) is affected by the language choice of the locale. Spanish and English locale sort differently.

The date format of ls(1) is affected by the locale. The date format of "**LANG=C ls -l**" and "**LANG=en_US.UTF-8**" are different (see Section 9.2.5, “Customized display of time and date”).

Number punctuation are different for locales. For example, in English

locale, one thousand one point one is displayed as "**1,000.1**" while in German locale, it is displayed as "**1.000,1**". You may see this difference in spreadsheet program.



Chapter 7. The X Window System

Chapter 9. System tips