Guide to Localization of Open Source Software

NepaLinux Team
Madan Puraskar Pustakalaya

www.mpp.org.np

www.idrc.ca

Published by

Center for Research in Urdu Language Processing
National University of Computer and Emerging Sciences
Lahore, Pakistan

on Behalf of

Madan Puraskar Pustakalaya
Kathmandu, Nepal

Printed by Walayatsons, Pakistan

ISBN: 978-969-8961-02-2

This work was carried out with the aid of a grant from the International Development Research Centre (IDRC), Ottawa, Canada, administered through the Centre for Research in Urdu Language Processing (CRULP), National University of Computer and Emerging Sciences (NUCES), Pakistan.

Preface

The release of NepaLinux 1.0 in December 2005, by Madan Puraskar Pustakalaya was a major breakthrough for software localization in Nepal. The open source nature and no licensing cost of NepaLinux provided a viable alternative to more costly proprietary software in Nepali. This localization work was based on existing open source distributions in Linux. While the open source movement has provided free, open and easy access to the source code, accelerating localization development, the need to document the processes involved has also become equally important to trigger further localization for under-resourced languages.

This Linux localization guide is a compilation of the experiences of the Madan Puraskar Pustakalaya localization team while they worked on the localization of Debian and Morphix based GNU/Linux Distribution in Nepali. Special attention has been given in making the content useful to those undertaking the localization work for the first time for other languages in their respective languages. Illustrations in most cases are based on the works in the Nepali language. However, information about the basic steps and procedures for localization has been made as generic as possible, in order to ensure that any language may fit into the description provided. During the preparation of this guide, a large number of resources invaluable to both beginners and experts of localization have been consulted. References have been provided for further reading for these topics.

We would like to acknowledge G. Karunakar of SARAI and IndLinux, Javier Sola of the Khmer OS initiative, Jaldhar Vyas, Suyash Shrestha for their advice and help during the development of NepaLinux 1.0. Our acknowledgements are also due to networks like bytesforall, debian.org etc.

This work has been made possible through the support of National University of Computer and Emerging Sciences (NUCES), Pakistan, and Pan Asia Networking (PAN) program of International Development Research Center (IDRC), Canada.

Madan Puraskar Pustakalaya (MPP) and the PAN Localization Project

MPP, a non-profit institution, is principally an archive house of published materials in the Nepali Language. It has been working in the field of software localisation and Nepali Language Computing since the year 2000. MPP's involvement in computing began with the Pustakalaya's decision to electronically catalogue its collections nearly a decade ago. Since the available technology (a number of Nepali fonts like Preeti, JagHimali, Kanchan, etc.) lacked data processing capabilities, as well as had inconsistencies in the keyboard mappings and layout, there was no alternative other than to develop a standard keyboard input method for Nepali.

It was against this backdrop that MPP undertook the Font Standardization Project, supported by the United Nations Development Project and the Ministry of Science and Technology. A direct outcome of the project was the two keyboard drivers, Nepali Unicode Romanized and the Nepali Unicode Traditional, developed by MPP. This solved the problems of data processing constraints and the inconsistencies of keyboard mapping that existed for the Nepali language. To continue the work initiated in Nepali Language Computing, MPP collaborated in the PAN Localization Project (1) in the year 2004.

PAN Localization, a multinational project developed for the purpose of enabling local language computing capacities in South- and Southeast Asia**.** Supported by the International Development and Research Centre (IDRC), Canada it is being run simultaneously in ten countries of South- and Southeast Asia, with MPP representing the Nepal component. MPP's focus has been on reducing the distance prevalent among the general Nepali people and computers due to the language barrier. MPP released the NepaLinux 1.0, which was enthusiastically taken up by the users; MPP is presently working on improving and refining the existing release by incorporating user feedbacks. Work is also underway to localize handheld devices like the PDA and mobile devices into Nepali, due to be released in December 2006 through this project. In the second phase of the PAN Localization Project, MPP plans to develop the Nepali Optical Character Recognition system and deploy the existing NepaLinux system to the end-users.

Natural Language Processing (NLP) applications like the Spell Checker for Nepali, Grammar Checker for Nepali, Machine Translation System for Nepali, Optical Character Recognition System for Nepali, Text-to-Speech System for Nepali are also slowly emerging. However, Nepali continues to be an under resourced

language in terms of the NLP tools and linguistic resources required for conducting the computational linguistics. Work is underway to build these resources.

Another factor that requires urgent attention is the dearth of NLP experts with adequate knowledge and expertise in language computing. MPP plans to establish a National Language and Technology Centre in Nepal which would serve as a common ground for conducting research and development in Nepali Language Computing, and also provide the institutional follow-up to all the activities carried out in the past decade. With the Centre established, the MPP also hopes to support the research works carried out by different individuals and institutions, in order to develop a strong NLP base in Nepal.

NepaLinux Team
Madan Puraskar Pustakalaya

PAN Localization Project

Enabling local language computing is essential for access and generation of information, and also urgently required for development of Asian countries. PAN Localization project is a regional initiative to develop local language computing capacity in Asia. It is a partnership, sampling eight countries from South and South-East Asia, to research into the challenges and solutions for local language computing development. One of the basic principles of the project is to develop and enhance capacity of local institutions and resources to develop their own language solutions.

The PAN Localization Project has three broad objectives:

> To raise sustainable human resource capacity in the Asian region for R&D in local language computing
>
> To develop local language computing support for Asian languages
>
> To advance policy for local language content creation and access across Asia for development

Human resource development is being addressed through national and regional trainings and through a regional support network being established. The trainings are both short and long term to address the needs of relevant Asian community. In partner countries, resource and organizational development is also carried out by their involvement in development of local language computing solutions. This also caters to the second objective. The research being carried out by the partner countries is strategically located at different research entry points along the technology spectrum, with each country conducting research that is critical in terms of the applications that need to be delivered to the country's user market. Moreover, PAN Localization project is playing an active role in raising awareness of the potential of local language computing for the development of Asian population. This will help focus the required attention and urgency to this important aspect of ICTs, and create the appropriate policy framework for its sustainable growth across Asia.

The scope of the PAN Localization project encompasses language computing in a broader sense, including linguistic standardization, computing applications, development platforms, content publishing and access, effective marketing and dissemination strategies and intellectual property rights issues. As the PAN Localization project researches into problems and solutions for local language computing across Asia, it is designed to sample the cultural and linguistic diversity in the whole region. The project also builds an Asian network of researchers to share learning and knowledge and publishes research outputs, including a comprehensive review at the end of the project, documenting effective processes, results and recommendations.

Countries (and languages) directly involved in the project include Afghanistan (Pashto and Dari), Bangladesh (Bangla), Bhutan (Dzongkha), Cambodia (Khmer), Laos (Lao), Nepal (Nepali), Sri Lanka (Sinhala and Tamil) and Pakistan, which is the regional secretariat. The project started in January 2004 and will continue for three years, supporting a team of seventy five resources across these eight countries to research and develop local language computing solutions. The project will be going into a second phase, extending the scope of partnership, countries and research, focusing on deploying the local language technology to the end-users. The second phase of the project will continue till 2010. Further details of the project, its partner organizations, activities and outputs are available from its website, www.PANL10n.net.

Contributors of the Guide

Editing: Bal Krishna Bal, bal@mpp.org.np

Chapter 1. Localization and Localization key concepts
- Bal Krishna Bal

Chapter 2. Locale Development
- Paras Pradhan, Subir Bahadur Pradhanang, paras@mpp.org.np, subir@mpp.org.np

Chapter 3. Rendering and Rendering Engines
- Paras Pradhan, Pawan Chitrakar, pawan@mpp.org.np, Minal Koirala, Sarin Pradhan, Srishtee Gurung, srishtee@mpp.org.np

Chapter 4. GNU/Linux and Fonts
- Subir Pradhanang

Chapter 5. Input Methods for Linux
-Paras Pradhan, Basanta Shrestha, basanta@mpp.org.np

Chapter 6. Translation Aspects in Localization
-Bal Krishna Bal, Pawan Chitrakar, Srishtee Gurung, Shiva Pokharel,shiva@mpp.org.np

Chapter 7. GNOME Localization
-Pawan Chitrakar

Chapter 8. Mozilla Suite Localization
-Basanta Shrestha

Chapter 9. Mozilla FireFox Localization
-Basanta Shrestha

Chapter 10. Openoffice.Org Localization
-Subir Bahadur Pradhanang, Prajol Shrestha, prajol@mpp.org.np

Chapter 11. Linux Distribution Developement for Localization
-Paras Pradhan

Chapter 12. Developement of Internationalized Applications
-Dibyendra Hyoju, dibyendra@mpp.org.np

Chapter 13. Building Free Open Source Software (FOSS) Communities
-Bal Krishna Bal, Subir Bahadur Pradhanang

Chapter 14. Localization Project Management Techniques, Expertiences of Madan Puraskar Pustakalaya under the PAN Localization Project
-Srishtee Gurung

Table of Contents

1 Introduction

This Localization Guide comprises of fourteen chapters with an overview about localization basics in the first chapter followed by locale development in Chapter 2 which deals with the experiences collected by the Nepal Component Team, while working for the Nepali Language. The rendering issue vital to localization is dealt with briefly in Chapter 3. , A brief discussion in Chapter 4 about font technology and usage of fonts in GNU/Linux is followed in Chapter 5, by the different types of input methods available in Linux and the required procedures for their development. Chapter 6 discusses different aspects of translation, a major activity in localization, as well as the requirements of the Translation Manager, the translation process management, human resource estimation required for translation and translation tools . The next four Chapters (7,8,9 and 10) are dedicated to the details of localization of applications viz., Gnome Desktop, Mozilla Suite, OpenOffice.org, Mozilla FireFox and OpenOffice.org respectively, starting from general introduction, historical information, the localization components and frameworks, tools required for localization, procedures for building from source to the submission of the files to official sites. In Chapter 10, we deal with OpenOffice.org, as well as discuss the development and implementation of the Spell Checker and Thesaurus for non-English languages. Chapter 11 deals with the localization of Linux Distribution, different types of Linux Distributions and the development of the Live CD, Debian-based Linux Distribution. In Chapter 12, the development of Internationalized Open Source Applications and a brief overview on internationalization in terms of software development is discussed. An overview on developing QT based Applications and localizing them is also included. The issues on building Free and Open Source Software (FOSS) communities have been addressed briefly in Chapter 13. The localization guide concludes with a general overview of the Localization Project Management Techniques gained by the Nepal Component of the PAN Localization Project in chapter 14.

The final judgement to the quality of the Localization Guide rests with the readers. We do not claim this to be a complete guide to localization; however, in terms of writing the guide, we have tried to provide the information on the specified topics to the best of our knowledge and expertise.

1.1 What is Localization?

The widely accepted definition of localization defines it as the process of adapting, translating and customizing a product (software) for a specific market. Hence this involves dealing with a specific locale or cultural conventions. By locale, we generally understand convents such as sort order, keyboard layout, date, time, number and currency format.

To many of us, localization might seem identical or similar to translation. However, the process of localization is much broader than simply translation. In this context, it would be highly relevant to put forward the definition of localization by the Localization Industry Standards Association (LISA). As per LISA, localization is defined as "the process of modifying products or services to account for differences in distinct markets". Hence, in order to have rightly called a software being localized in the truest sense, it should provide the local "look-and-feel" while working with it. This involves input support in the software, proper display of the input text, the support for date, currency and time for a particular local language and locality. In practice, this means that localization needs to address three main issues[1.5.a]:

a) Linguistic Issues
This essentially covers the translation of a product's user interface and documentation.

b) Content and Cultural Issues
This relates to adapting the information and functionality in products as per the norms acceptable to the local audience. Issues like the necessity of designing and developing specific software as per the locally accepted norms and regulations in terms of preference over colors, graphics, icons etc. is generally taken into consideration under content and cultural issues.

c) Technical Issues
While rendering support for local languages and content, there are certain issues that need to be addressed in localization. Handling bi-directional texts requires some extra effort in design and engineering. For example, the Arabic script as compared to Roman or Latin. Similarly, the fact that Far Eastern languages require twice the disk space as is required for English, where each character also

needs to be considered while dealing with the localization of software in these languages.

1.2 Factors to be Considered while Deciding to Localize a Software

The following factors need to be considered before deciding whether or not to localize a software.

a) Nature and Scope of the Software Product
The applicability of the software product in the local market needs to be taken into consideration. If the software has inapplicable features in the local context, one has to initially add the required features before translating it.

b) Size of the Target Market and Audience
The size of the target market and audience also plays a major role in the localization, especially to individuals and commercial companies working in the field of localization.

c) Length of the Product Life Cycle and Anticipated Update Frequencies
This issue also needs to be taken into consideration as unnecessarily lengthy product life cycles and high or low update frequencies also adversely effect the general usability of the software.

d) Competitor Behavior
The demand of any product in the market primarily depends upon the competitiveness it can offer as opposed to it's alternatives. Hence, this factor also needs to be taken into consideration.

e) Market Acceptance
Hand in hand with the analysis of the available features in the software, before localizing it one also has to take into consideration whether the software retains the quality of being accepted by the market in terms of qualitative service or not.

f) National or International Legislation
This involves the licensing issues, distribution, and redistribution rights related to the software.

1.3 Why is Localization Important?

Research has shown that the lack of availability of information in the locally understandable language is the main reason for the slow progress in Information and Communication Technology (ICT) sector by most of the underdeveloped countries. In today's age, access to ICT plays a major role in the overall development of a country, it has become a challenge to bridge the digital divide caused by the language barrier. Even having learnt English, one has to pay hundreds of dollars to license foreign software, or take to widespread software piracy to gain access to ICT. The solution to all these is using the localized Free Open Source Software (FOSS).

1.4 Localization – Key Concepts

1.4.1 Standardization

Standardization is one of the baselines to be followed in localization. Standardization deals with certain universally accepted standards that need to be followed, so that two developers from any part of the globe could interact through the application developed without having to meet in person. Standardization becomes applicable in almost everything specific to the language – for instance, a standard glossary of terms for translation, a standard keyboard layout for input system, a standard collation sequence order for sorting and other data processing, a standard of fonts etc.

Hence, standards provide ultimate contracts or agreements for all computing systems in the world. Software developers need such conventions to conform to prevent disorders. Therefore, standardization should be the very first step for any type of software construction[1.5.c].

To start localization, it is a good idea to study related standards and use them throughout the projects. Nowadays, many international standards and specifications have been developed to covermost of the

languages of the world. Important organizations on standardization include[1.5.c]:

a) ISO/IEC JTC1 (International Organization for Standardization and International Electrotechnical Commission Joint Technical Committee 1)
b) Unicode Consortium (http://www.unicode.org)
c) Free Standards Group (http://www.freestandards.org)

1.4.2 Character Sets and Encoding

Every language is characterised by a set of characters. A character is a basic element of a text. Characters are used to form larger textual units like words[1.5.d]. In mathematical terms, the character set defines the set of all characters used in a language. In order to store characters used in human languages in a computer, we need to store them in a way the computer understands. Since computers deal only with numbers, it is necessary to devise some kind of mapping whereby a particular character corresponds to a particular number. This mapping, in other words is often known as character encoding. Applications developed for internationalisation take into consideration the support required for representing the character sets of various different languages. Similarly, when localizing a software into a specific language, the application should take into consideration an encoding scheme that can represent characters of the target language.

The first step in representing the human language in the computer is to identify the characters in the language and collect all of them to form a set of characters called the Character Repertoire. Once the Character Repertoire is formed, the next step is to define an encoding scheme which maps each character in the Character Repertoire to a unique integer, the mapping being the encoding. Encoding schemes refer to this unique integer as the code-point or encoded values[1.5.d].

1.4.3 SingleByte and MultiByte Encodings

Encodings are classified as SingleByte or MultiByte[1.5.d]. SingleByte encoding schemes use a single byte to represent each character. They are regarded as the most efficient encodings available. They take least amount of space and are easy to process because one character is represented by one byte. 7-bit encoding and 8-bit encoding schemes come under this category. A 7-bit encoding scheme can define up to 128 characters and normally support a single language. A 8-bit encoding scheme can define up to 256 characters and generally support a group of related languages.

MultiByte encoding schemes use multiple bytes to represent a single character. These schemes are either a fixed or variable number of bytes to represent a character. A fixed-width multibyte encoding scheme uses a fixed number of bytes to represent every character of its Character Repertoire. A variable width multibyte encoding scheme uses one or more bytes to represent a single character.

1.4.4 Different Encoding Systems

There are various encoding systems in use today. Since detailed information about them is easily available, we simply list them below.

a) ASCII
b) Base64
c) CODE-PAGES
d) ISO 8859-1
e) UCS (defined by ISO 10646)
f) Unicode (UTF-32,UTF-16,UTF-8,UTF-7)
g) UCS 2 and UCS 4

1.4.5 Encodings and Localization

Encoding in localization plays a major role as the input and output text in a localized software would need the encoding information for data processing. Similarly, string processing and display also requires encoding information. For example, if the encoding used is not understood by the rendering engine, any text will appear as gibberish.

1.4.6 Fonts and Output Methods

Provided that the character set and encoding of a script are defined, the first step to enabling it onto a

system is to display it. Rendering text on screen requires some resource to describe the shapes of the characters i.e. the fonts, and some process to render the character images as per script conventions. The process is called the output method[1.5.c].

1.4.7 Characters and Glyphs

A font is a set of glyphs for a character set. A glyph is an appearance form of a character or a sequence of characters. It's important to understand the concepts of characters and glyphs. For some scripts, a character can have more than one variation, depending on the context. In that case, the font may contain more than one glyph for each of those characters. On the other hand, the concept of ligatures, such as "®" in English , also allows some sequence of characters to be drawn together. This introduces another kind of mapping from more than one character into a single glyph[1.5.c].

1.4.8 Bitmap and Vector Fonts

Generally, there are two methods of describing fonts: bitmaps and vectors. While bitmap fonts describe glyph shapes by plotting the pixels directly into a two dimensional grid of determined size, vector fonts describe the outlines of the glyphs with line and curves. Both the font types have their own pros and cons. Since bitmap fonts are designed for a particular size, the quality of bitmap fonts always drops when scaled up. Such a problem does not apply to vector fonts. From this perspective, vector fonts seem to be a good choice. But at the same time, vector fonts have poor display in low resolution devices, such as computer screens, to which bitmap fonts are better alternatives.

1.4.9 Output Methods

An output method is a general procedure for drawing texts on output devices. It converts text strings into sequences of properly positioned glyphs of the given fonts. With English, the character-to-glyph mapping is straightforward but when it comes to other scripts of greater complexity, output methods are more complicated.

Traditional font technologies do not store the information required for handling the complex scripts in the fonts itself. Owing to this, the output methods bear the burden. In contrast, OpenType fonts function according to the rules stored in the fonts. This makes the task for the output methods relatively easier as all that is required is the capability to read and apply the rules. Different output methods exist for different implementations. In case of X Window(for more details on X Window, refer to chapter 7, GNOME Localization.), the output method is the X Output Method(XOM). A separate module called Pango is used for GTK+. Output method is handled by some classes as far as Qt is concerned.

OpenType fonts are now widely supported by modern rendering engines. So, basically OpenType fonts may be used with OpenType tables that describe rules for glyph substituions and positionings. But if TrueType or Type 1 fonts are used, an output method capable of processing and typesetting characters of the particular script is needed.

1.4.10 Input Methods

Many factors need to be considered relating to design and implementation of the input method. This involves the size of the character set for a particular language, the capability of the input device and so on. Consider typing English characters from a normal keyboard and from a mobile phone keypad. Again think of languages with huge character set like the CJK, the input turns out to be complicated even from the normal keyboard. Hence, it is understandable that analysis and design are important stages in the input method development.

First and foremost, all the characters required for input should be figured out. This should include digits and punctuation marks. Once the decision on the required input characters has been finalized, then the input scheme may be formulated, whether the characters be matched one-by-one with the available keys, or some combination or conversion is required for dealing with multiple keystrokes to input some characters.

After deciding the input scheme, the next step is designing the keyboard layout. While designing the keyboard layout, special attention should be given to make it easy and comfortable for the typists. From this view, the general principle is putting the most frequently used characters in the home row followed by the upper and the lower rows. In case the upper case and lower case concepts do not exist for a certain language, rarely used characters are generally placed in the shifted positions.

In terms of implementation of the input method, there are two major steps required. First, the description of the keyboard layout has to be created looking at the available keyboard maps after which one has to write the input method module based on the keyboard map. The input method module developed may be plugged into the system to be used.

1.4.11 Locales

Locale is a term introduced by the concept of internationalization (I18N), in which generic frameworks are made so that software can adjust its behaviors to the requirements of different native languages, cultural conventions and coded character sets, without modification or re-compilation[1.5.c]. Locales, describing particular cultures are defined within such frameworks. Under such arrangement, users when configuring their systems find their locales picked up in the respective applications. Provided the locale definition file, internationalized applications may easily accomplish functions specific to a particular locale of any country.Hence the only thing required for an internationalized software to support a new language or culture is creating a locale definition for the specific language and filling up the required information. The most interesting part is that things work perfectly without changing the actual source code. For more detailed information on locales, refer to Chapter 2, Locale Development.

1.4.12 Basic Steps for Linux Localization

As evident from the above information on localization, the basic procedures for Linux Localization are as follows:

a) Creating locales
b) Font development
c) Choosing the input method and creating keyboard mappings
d) Rendering engines (rendering engines need to be updated if necessary)
e) Translation
f) Localization of applications like OpenOffice.org, Gnome Desktop , Mozilla Suite etc for supporting local language support

1.5 References for Further Reading

a) The Localization Industry Primer, LISA- The Localization Industry Standards Association. 2nd Edition 2003. Available on http://www.lisa.org/interact/LISAprimer.pdf
b) http://www.lisa.org/products/primer
c) The Primer: Localization of Free/Open Source Software. Anousak Souphavanh and Theppitak Karoonboonyanan.
d) How-to Guide for Localization by International Open Source Network and Center for Development of Advanced Computing, Mumbai. Draft for Feedback Edition. Published October, 2004.
e) http://nepalinux.org/docs/l10nhowtoguide.pdf

2 Locale Development

2.1. Introduction

In Chapter 1, we have briefly touched upon the general introduction of locales and the actual need of locales in localization. In this Chapter, we will mainly focus on the technical aspects related to locale development. It is evident from the information on locales in Chapter 1 that every language has its own locale. In this regard, one needs to note that many localized softwares are dependent on locales. For example, gnome desktop, sort utilities etc. As noted earlier, a locale is built using the locale definition file. In Linux,the locale definition file is part of the Glibc package. Installation of the Glibc package copies almost all of the Linux locale definition files created across the globe in the local computer which can be later used to build locale. For example, the Nepali locale developed and submitted is named 'ne_NP' which denotes Nepali Language for the country, Nepal.

2.2. Locale Naming

The general naming convention of a locale is as follows:

{lang}{territory}{codeset}[@{modifier}]

A brief explanation follows:
lang = 2 letter language code as defined in ISO 639:1988. Three letter language code is defined in ISO 639-2 which is used in the absence of the two letter version. The ISO 639-2 Registration Authority at Library of Congress has a complete list of language codes.

territory = 2 letter country code as defined in ISO 3166-1:1997. You can get these codes from the ISO 3166 Maintenance agency.

codeset = Character set used.

modifiers = Optional and is meant to be used for adding more information to the locale by setting options. Options are separated by commas. Ex: fr_CA.ISO-8859-1, denotes French language spoken in Canada and the character set being used as defined by the ISO-8859-1.

2.3. Locale and Applications

Individual applications may require separate locales to be developed. To cite an example, since Gnome or Gtk based applications use glibc library, the locale for glibc should be developed. However, in the case of openoffice, as the glibc locale is not used, the development of a separate locale is required. To simplify matters, provided that you intend to use just Gnome Desktop, Gnome/GTK based applications, and OpenOffice.org office suite only, then the development of glibc and OpenOffice.org locales are sufficient.

2.4. Basics Steps of Locale Development

Locale development involves the following basic procedures:

a) Gathering the standardized locale information for the specific country;
b) Developing the Locale Definition File;
c) Submitting the developed locale in the main stream.

2.5. Glibc Locale Development

In the following section, we describe the glibc locale development for the country, Nepal. The information presented can be refered to by other countries for their own locale development. However, we do not claim the information to be complete. It is advisable to refer to the resources listed at the end of this chapter for further reading. As noted earlier, the locale definition file has some predefined sections in it, which must be defined in conformance with the standardized locale information of a particular country. Below, we explain

each section illustrating the case of Nepal and the Nepali language, thus named as ne_NP.

a) LC_CTYPE

This category begins with LC_CTYPE and ends with END LC_CTYPE. It defines character classification and specifies characters that are alphanumeric, numeric, punctuation, hexadecimal, blank, control characters etc. The following keywords are used inside LC_CTYPE.

copy

This specifies the name of the existing locale from which the definition of this category has to be copied. If this is specified, no other keywords can be used. For example: copy “i18n” (refers to the default definition)

upper

Upper case letter characters. Cntrl, digit, punct or space characters cannot be specified here. For example: upper <A>;<B>.....<Z>

lower

Lower case characters. Cntrl, digit, punct or space characters cannot be specified here. For example: lower <a>;<b>.....<z>

alpha

All letter characters. Cntrl, digit, punct or space characters cannot be specified here. For example: alpha <A>;<a>.....<z>

digit

All the digit characters. For example: digit <zero>;<one>;<two>;<six>

alnum

All the alphanumeric characters. Alpha and Digit category are included automatically. Cntrl, punct or space characters cannot be specified here. For example: alnum <A>;<B>;<one>

space

All whitespace characters. Characters specified with the blank keyword must be specified. Cntrl, alpha, upper, lower,graph,digit,xdigit characters cannot be specified here. For example: space <tab>;<newline>;<carriage-return>;<space>

cntrl

All control characters. Alpha, upper, lower,digit,graph,punct,print,space or xdigit cannot be specified here. For example: cntrl <alert>;<backspace>;<tab>;<ESC>

punct

Specifies punctuation characters. However alpha, upper, lower,digit, space or xdigit cannot be specified here. For example: punct <exclamation-mark>;<quotation-mark>;<dollar-sign>;<colon>;<backslash>

x. graph

All printable characters excluding <space> character. If not specified, all characters defined by alpha,upper,lower,digit,xdigit and punct are automatically included in this category. Cntrl characters cannot be specified here.

print

All printable characters including <space> character. If not specified, all characters defined by alpha, upper, lower, digit, xdigit and punct are automatically included in this category. Cntrl characters cannot be specified here.

xdigit

Hexadecimal digit characters. For example: xdigit <zero>;<nine>;<A>;<F>;<a>;<f>

blank

Defines blank characters. For example: blank <space>;<tab>

charclass
For example: toupper (<a>,<A>);(<z>,<Z>)

Given below is a sample of the LC_CTYPE.

LC_CTYPE sample:

```
LC_CTYPE
# The following is the POSIX locale LC_CTYPE.
# "alpha" is by default "upper" and "lower"
# "alnum" is by definition "alpha" and "digit"
# "print" is by default "alnum", "punct", and the <space>
# "graph" is by default "alnum" and "punct"
#
upper    <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
        <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
#
lower    <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
        <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
#
digit    <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
        <seven>;<eight>;<nine>
#
space    <tab>;<newline>;<vertical-tab>;<form-feed>;\
        <carriage-return>;<space>
#
cntrl    <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
        <form-feed>;<carriage-return>;\
        <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
        <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
        <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
        <IS1>;<DEL>
#
punct    <exclamation-mark>;<quotation-mark>;<number-sign>;\
        <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
        <left-parenthesis>;<right-parenthesis>;<asterisk>;\
        <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
        <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
        <greater-than-sign>;<question-mark>;<commercial-at>;\
        <left-square-bracket>;<backslash>;<right-square-bracket>;\
        <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
        <vertical-line>;<right-curly-bracket>;<tilde>
#
xdigit   <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
        <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
#
blank    <space>;<tab>
#
toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
       (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
       (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
       (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
       (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);(<z>,<Z>)
#
tolower (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);\
       (<F>,<f>);(<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);\
       (<K>,<k>);(<L>,<l>);(<M>,<m>);(<N>,<n>);(<O>,<o>);\
       (<P>,<p>);(<Q>,<q>);(<R>,<r>);(<S>,<s>);(<T>,<t>);\
       (<U>,<u>);(<V>,<v>);(<W>,<w>);(<X>,<x>);(<Y>,<y>);(<Z>,<z>)
END LC_CTYPE
```

LC_CTYPE used in ne_NP:

```
LC_CTYPE
copy "i18n"
END LC_CTYPE
```

b) LC_COLLATE

This category is related to sorting and is assumed to be the most complicated among all the locale categories. Collation of Unicode strings follow the standard ISO/IEC 14651. International string ordering and glibc locale is based on this standard.
Given below is the sample of the LC_COLLATE.

LC_COLLATE sample:

```
order_start
forward;backward
UNDEFINED
IGNORE;IGNORE
<LOW>
<space>
<LOW>;<space>
...
<LOW>;...
<a>
<a>;<a>
<a-acute>
<a>;<a-acute>
<a-grave>
<a>;<a-grave>
<A>
<a>;<A>
<A-acute>
<a>;<A-acute>
<A-grave>
<a>;<A-grave>
<ch>
<ch>;<ch>
<Ch>
<ch>;<Ch>
<s>
<s>;<s>
<eszet>
"<s><s>";"<eszet><eszet>"
order_end
```

LC_COLLATE in ne_NP

The standardized collation for the Nepali language is implemented in ne_NP locale definition file. The sample of LC_COLLATE category defined in ne_NP is like the one presented below.

```
LC_COLLATE
collating-element <ksha> from "<U0915><U094D><U0937>"
collating-element <tra> from "<U0924><U094D><U0930>"
collating-element <jna> from "<U091C><U094D><U091E>"
order_start forward
<U0901>
<U0902>
<U0903>
<U0905>
<U0906>
<U0907>
```

<U0908>
<U0909>
<U090A>
<U090B>
<U090F>
<U090E>
<U0913>
<U0914>
<U0915>
<U0916>
<U0917>
<U0918>
<U0919>
<U091A>
<U091B>
<U091C>
<U091D>
<U091E>
<U091F>
<U0920>
<U0921>
<U0922>
<U0923>
<U0924>
<U0925>
<U0926>
<U0927>
<U0928>
<U092A>
<U092B>
<U092C>
<U092D>
<U092E>
<U092F>
<U0930>
<U0932>
<U0935>
<U0936>
<U0937>
<U0938>
<U0939>
<ksha>
<tra>
<jna>
<U093C>
<U093D>
<U093E>
<U093F>
<U0940>
<U0941>
<U0942>
<U0943>
<U0947>
<U0948>
<U094B>
<U094C>
<U094D>
<U0950>
<U0951>
<U0952>

```
<U0953>
<U0954>
<U0964>
<U0965>
<U0966>
<U0967>
<U0968>
<U0969>
<U096A>
<U096B>
<U096C>
<U096D>
<U096E>
<U096F>
<U0970>
order_end
END LC_COLLATE
```

c) LC_MONETARY

This deals with the format to show monetary numbers, currency symbols, comma or period, columns etc.

```
LC_MONETARY
# This is the POSIX locale definition for
# the LC_MONETARY category.
#
int_curr_symbol      ""
currency_symbol      ""
mon_decimal_point    ""
mon_thousands_sep    ""
mon_grouping         -1
positive_sign        ""
negative_sign        ""
int_frac_digits      -1
p_cs_precedes        -1
p_sep_by_space       -1
n_cs_precedes        -1
n_sep_by_space       -1
p_sign_posn          -1
n_sign_posn          -1
#
END LC_MONETARY
```

LC_MONETARY used in ne_NP is as follows:

```
LC_MONETARY

int_curr_symbol "<U004E><U0050><U0052><U0020>"
currency_symbol "<U0930><U0942>"
mon_decimal_point "<U002E>"
mon_thousands_sep "<U002C>"
mon_grouping 3;2
positive_sign "<U002B>"
negative_sign "<U002D>"
int_frac_digits 2
frac_digits 2
p_cs_precedes 1
p_sep_by_space 1
n_cs_precedes 1
n_sep_by_space 1
```

```
p_sign_posn 1
n_sign_posn 1
%
END LC_MONETARY
```

d) LC_NUMERIC
Specifies number format, position of decimal digit separators.

Example:

```
LC_NUMERIC
# This is the POSIX locale definition for
# the LC_NUMERIC category.
#
decimal_point    "<period>"
thousands_sep    ""
grouping         -1
#
END LC_NUMERIC
```

Sample used in ne_NP is as:

```
LC_NUMERIC
decimal_point "<U002E>"
thousands_sep "<U002C>"
grouping 3
END LC_NUMERIC
```

e) LC_TIME
This category is related to the formatting of date and time information. The following keywords are used in this category:

copy
abday: Abbreviated week names as: sun, mon.
day: Full spelling of week names: Sunday, Monday.
abmon: Abbreviated month names as: Jan, Feb.
mon: Full spelling of month named as: January, February.
am_pm: Represent AM and PM.
d_f_fmt: Standard date and time format
d_fmt: Standard date format
t_fmt: Standard time format
t_fmt_ampm_time: Specifies 12-hour clock format that includes the am_pm value.

Example of LC_TIME

```
LC_TIME
# This is the POSIX locale definition for
# the LC_TIME category.
#
# Abbreviated weekday names (%a)
abday      "<S><u><n>";"<M><o><n>";"<T><u><e>";"<W><e><d>";\
          "<T><h><u>";"<F><r><i>";"<S><a><t>"
#
# Full weekday names (%A)
day        "<S><u><n><d><a><y>";"<M><o><n><d><a><y>";\
          "<T><u><e><s><d><a><y>";"<W><e><d><n><e><s><d><a><y>";\
          "<T><h><u><r><s><d><a><y>";"<F><r><i><d><a><y>";\
          "<S><a><t><u><r><d><a><y>"
#
# Abbreviated month names (%b)
```

```
abmon      "<J><a><n>";"<F><e><b>";"<M><a><r>";\
        "<A><p><r>";"<M><a><y>";"<J><u><n>";\
        "<J><u><l>";"<A><u><g>";"<S><e><p>";\
        "<O><c><t>";"<N><o><v>";"<D><e><c>"
#
# Full month names (%B)
mon        "<J><a><n><u><a><r><y>";"<F><e><b><r><u><a><r><y>";\
        "<M><a><r><c><h>";"<A><p><r><i><l>";\
        "<M><a><y>";"<J><u><n><e>";\
        "<J><u><l><y>";"<A><u><g><u><s><t>";\
        "<S><e><p><t><e><m><b><e><r>";"<O><c><t><o><b><e><r>";\
        "<N><o><v><e><m><b><e><r>";"<D><e><c><e><m><b><e><r>"
#
# Equivalent of AM/PM (%p)      "AM";"PM"
am_pm      "<A><M>";"<P><M>"
#
# Appropriate date and time representation (%c)
#    "%a %b %e %H:%M:%S %Y"
d_t_fmt    "<percent-sign><a><space><percent-sign><b>\
        <space><percent-sign><e><space><percent-sign><H>\
        <colon><percent-sign><M><colon><percent-sign><S>\
        <space><percent-sign><Y>"
#
# Appropriate date representation (%x)   "%m/%d/%y"
d_fmt      "<percent-sign><m><slash><percent-sign><d>\
        <slash><percent-sign><y>"
#
# Appropriate time representation (%X)   "%H:%M:%S"
t_fmt      "<percent-sign><H><colon><percent-sign><M>\
        <colon><percent-sign><S>"
#
# Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
t_fmt_ampm "<percent-sign><I><colon><percent-sign><M><colon>\
        <percent-sign><S> <percent_sign><p>"
#
END LC_TIME
```

Sample of LC_TIME used in ne_NP locale definition file is as follows:

```
LC_TIME
% Abbreviated weekday names (%a)
abday "<U0906><U0907><U0924><U0020>";/
"<U0938><U094B><U092E><U0020>";/
"<U092E><U0919><U094D><U0917><U0932><U0020>";/
"<U092C><U0941><U0927><U0020>";/
"<U092C><U093F><U0939><U0940><U0020>";/
"<U0936><U0941><U0915><U094D><U0930><U0020>";/
"<U0936><U0928><U093F><U0020>"
%
% Full weekday names (%A)
day "<U0906><U0907><U0924><U092C><U093E><U0930><U0020>";/
"<U0938><U094B><U092E><U092C><U093E><U0930><U0020>";/
"<U092E><U0919><U094D><U0917><U0932><U092C><U093E><U0930><U0020>";/
"<U092C><U0941><U0927><U092C><U093E><U0930><U0020>";/
"<U092C><U093F><U0939><U0940><U092C><U093E><U0930><U0020>";/
"<U0936><U0941><U0915><U094D><U0930><U092C><U093E><U0930><U0020>";/
"<U0936><U0928><U093F><U092C><U093E><U0930><U0020>"
%
```

```
% Abbreviated month names (%b)
abmon "<U091C><U0928>";/
"<U092B><U0947><U092C>";/
"<U092E><U093E><U0930><U094D><U091A>";/
"<U0905><U092A><U094D><U0930><U093F>";/
"<U092E><U0947>";/
"<U091C><U0942><U0928>";/
"<U091C><U0941><U0932><U093E>";/
"<U0905><U0917>";/
"<U0938><U0947><U092A><U094D><U091F>";/
"<U0905><U0915><U094D><U091F>";/
"<U0928><U094B><U092D><U0947>";/
"<U0921><U093F><U0938><U0947>"
%

% Full month names (%B)
mon "<U091C><U0928><U0935><U0930><U0940>";/
"<U092B><U0947><U092C><U094D><U0930><U0941><U0905><U0930><U0940>";/
"<U092E><U093E><U0930><U094D><U091A>";/
"<U0905><U092A><U094D><U0930><U093F><U0932>";/
"<U092E><U0947>";/
"<U091C><U0942><U0928>";/
"<U091C><U0941><U0932><U093E><U0908>";/

 <U0905><U0917><U0938><U094D><U0924>";/
"<U0938><U0947><U092A><U094D><U091F><U0947><U092E><U094D><U092C><U0930>";/
"<U0905><U0915><U094D><U091F><U094B><U092C><U0930>";/
"<U0928><U094B><U092D><U0947><U092E><U094D><U092C><U0930>";/
"<U0921><U093F><U0938><U0947><U092E><U094D><U092C><U0930>"
%
% Equivalent of AM PM
am_pm "<U092A><U0942><U0930><U094D><U0935><U093E><U0939><U094D><U0928>";/
"<U0905><U092A><U0930><U093E><U0939><U094D><U0928>"
%
% Appropriate date and time representation
% %Y %B %d %I:%M:%S %p (%Z-->optional) EX: 2004 November 01 11:30:40 PM
d_t_fmt
"<U0025><U0059><U0020><U0025><U0042><U0020><U0025><U0064><U0020><U0025><U0049><
    U003A><U0025><U0
04D><U003A><U0025><U0053><U0020><U0025><U0070><U0020>"
%
% Appropriate date representation
% %Y %B %d %A
d_fmt
    "<U0025><U0059><U0020><U0025><U0042><U0020><U0025><U0064><U0020><U0025><U004
    1><U0020>"
%
END LC_TIME
```

f) LC_MESSAGES

This category is related to the language in messages, which the software outputs. This category is used for gettext. For more detailed information on gettext, please refer to Chapter 7, GNOME localization.

Sample of LC_MESSAGES:

```
# This is the POSIX locale definition for
# the LC_MESSAGES category.
#
yesexpr  "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
#
```

```
noexpr   "<circumflex><left-square-bracket><n><N><right-square-bracket>"
#
yesstr   "yes"
nostr    "no"
END LC_MESSAGES
```

ne_NP Sample of LC_MESSAGES:

```
yesexpr "<U005E><U005B><U0079><U0059><U005D><U002E><U002A>"
noexpr "<U005E><U005B><U006E><U004E><U005D><U002E><U002A>"
yesstr "<U0059><U003A><U0079><U003A><U0079><U0065><U0073><U003A><U0939><U094B>"
nostr
    "<U004E><U003A><U006E><U003A><U006E><U006F><U003A><U0939><U094B><U0907><U0
    928>"
END LC_MESSAGES
```

g) LC_PAPER

Specifies standard paper width and height.

Sample used in ne_NP:

```
LC_PAPER
height 297
width 210
END LC_PAPER
```

h) LC_NAME

Specifies the standard format to write name.

Sample used in ne_NP:

```
LC_NAME
name_fmt
"<U0025><U0070><U0025><U0074><U0025><U0067><U0025><U0074><U0025><U006D><U0025>
    <U0074><U0025><U0066>"
name_gen "<U091C><U094D><U092F><U0942>"
name_mr "<U0936><U094D><U0930><U0940><U092E><U093E><U0928><U094D>"
name_mrs "<U0936><U094D><U0930><U0940><U092E><U0924><U0940>"
name_miss "<U0938><U0941><U0936><U094D><U0930><U0940>"
% salutation_fmt "<U0928><U092E><U093A><U094D><U0915><U093E><U0930>"
END LC_NAME
```

i) LC_ADDRESS

Specifies the standard way of writing Address.

Sample used in ne_NP:

```
% %f %N %h %s %N %T
postal_fmt
    "<U0025><U0066><U0025><U004E><U0025><U0068><U0025><U0073><U0025><U004E><U00
    25><U0054>"
END LC_ADDRESS
```

j) LC_TELEPHONE

Specifies the standard way of writing telephone numbers.

ne_NP Sample:

```
LC_TELEPHONE
tel_int_fmt
```

"<U002B><U0025><U0063><U0020><U0025><U0061><U0020><U0025><U006C><U0020>"
int_prefix "<U096F><U096D><U096D>"
END LC_TELEPHONE

k) LC_MEASUREMENT
Specifies default measurement unit.

Sample used in ne_NP:

LC_MEASUREMENT
measurement 1
END LC_MEASUREMENT

2.6. Glibc Locale Submission

After the development of locale, you have to submit it to the glibc main stream following the URL http://sourceware.org/bugzilla/

For this, you would need to create a new account,, login and start the process using NEW link under Actions. Then you will need to choose glibc link under "Enter Bug" and continue the process.

2.7. References for Further Reading

a) http://publibn.boulder.ibm.com/doc_link/en_US/a_doc_lib/files/aixfiles/Locale_Definition.htm
b) http://www.opengroup.org/pubs/online/7908799/xbd/locale.html
c) http://www.khmeros.info/tools/
d) http://www.it46.se/localegen/

3 Rendering and Rendering Engines

3.1. Introduction

In this Chapter, we briefly introduce the term "rendering". Among the different rendering engines available, we will give a brief introduction of Pango and ICU along with their features. Other rendering engines like Gecko will be dealt with in Chapter 8 and 9 under Mozilla Suite and FireFox Localization. We do not give details on rendering and rendering engines, as they do not relate to our expertise. The links for further reading on rendering and rendering engines are provided at the end of the Chapter, which can be referred to, after having gained an idea of the topic through the information provided.

3.2. Rendering

The glossary developed as per the Unicode Standard. Version 3 defines the term rendering as the process of selecting and laying out glyphs for the purpose of depicting characters on display devices.

Rendering internationalized text is often assumed to be a simple matter of dealing with fonts, however it entails several complications.This is justified by a number of languages like Arabic and Hebrew, which are written from right-to-left, instead of from left-to-right. In order for the text in these languages to appear on the screen properly, the rendering process needs to be able to deal with that ordering. The situation is made even more complicated by the fact that text in these languages usually consists of a mix of right-to-left and left-to-right text (numbers, foreign words). So, a complicated reordering process is needed between the in-memory representation and the actual drawing process. One more complication related to theArabic language is the varying shape of each character depending upon the actual occurrence of the character whether at the beginning of the word, in the middle of the word, at the end of the word, or by itself. In this context, the right glyph needs to be selected depending upon the situation. The languages of South Asia, often known as complex text languages also require special attention. In these languages, the characters making up a syllable interact in complex ways to produce the final rendered form. This can involve reordering, combining characters to make ligatures that appear very different from the original character, and stacking multiple glyphs on top of each other vertically. Similarly, a group of characters interact with each other, known as a cluster in some of the languages which follow the Devanagari script. All these facts need to be addressed fully for the proper rendering of the texts in these languages. Other issues like line-breaking algorithms being used for the rendering process would require specific linguistic information of the language in terms of implementation. This is because languages of East Asia or Thailand, as opposed to English do not use white spaces at all.

3.3. Rendering Engines

3.3.1. Pango

Pango is a library, which provides an open-source framework for layout and rendering of the internationalized text. Having used Unicode for all of its encodings, it aims to support output in all of the major languages. Pango can work on top of multiple display systems – including traditional X fonts, or client-side OpenType fonts.

The architectural and design features of Pango are characterized by the following[3.5.a]:

a) Unicode is used as a common character set throughout the system;

b) It is modular in terms of design;

The code specific to each language is contained in a separate, dynamically loaded module. This has several benefits. Firstly, it reduces the amount of code that is contained in the main library. Secondly, it allows modules for specific languages to be developed and distributed by teams familiar with those languages, instead of tying the development of support for a particular language to the release cycle of the core system.

c) Pango language module is divided into pieces, the language module and the shaper module.

The Language module takes into consideration system independent rendering – the same tasks needs to be performed whether using Xfonts to draw to the screen, drawing into an off-screen buffer in some other fashion, or printing to paper.

The Shaper module takes into consideration system dependent rendering – this deals with the positioning of the glyphs with respect to each other.

d) Pango has an abstract class named “Pango Font” which determines overall metrics of a font, metrics for an individual glyph and finds out which Unicode characters a font covers. “PangoXfont” subclass is used for handling Xfonts.

e) Pango Layout object is the higher level abstraction class which is initialized with a block of Unicode text, attributes for the text (font family, size, color, line-width, line spacing, indention etc). Pango layout deals with interactive editing also such as cursor movement with arrow keys etc).

3.3.2. ICU by IBM

ICU (International Components for Unicode) is the set of C/C++ and java libraries for Unicode support. Before, ICU was the internalization API of JDK 1.1 and later on it turned out to be the most advanced Unicode/i18n support. ICU supports the most current version of the Unicode standard. It produces the same results across various platforms without sacrificing performance.

The features of ICU are characterised by the following[3.5.b]:

a) Handles Unicode text;
b) Has the Locale and Resource Bundle packed;
c) Supports language sensitive collation and searching;
d) Supports Normalization, case conversion and script transliterations;
e) Has the features for the representation of comprehensive locale data;
f) Supports MultiCalendar and time zone;
g) Has the features for formatting and parsing of time, date, numbers, currency etc.

3.4. Basic Steps for Text Rendering

Text rendering basically involves the following steps[3.5.a]:

a) Itemization
In itemization , the input text is divided into Unicode strings, which are analyzed and broken into items. Each item is handled by a single language module dealing with a single direction, either left-to-right or right-to-left. If the font size and style is also set for the text, the items are further sub-divided into pieces of the same font category.

b) Boundary resolution
In this step, textual boundaries such as word boundaries and line breaks are determined for each item. The boundary resolution is handled by the function pango_break().

c) Shaping
Characters are taken within each item which are later converted into glyphs. The pango_shape() function is used for this purpose.

d) Line breaking
The results of shaping and boundary resolution are used to choose where to break lines that need to be wrapped. However, pango_shape() might need to be used further in case breaking lines involves dividing items.

e) Rendering
Rendering is the result of the shaping and line breaking process, which is a set of glyph strings (a list of glyphs from the font) along with positioning information for each glyph. Libpangox is used for rendering X fonts and libpangoft2 for rendering True type and postscript fonts via the free type library.

3.5. References for Further Reading

a) Pango: internationalized text handling, Owen Taylor, Red Hat, Inc.
b) Research report for rendering Nepali in Linux. Paras Pradhan, Pawan Chitrakar, Minal Koirala, Sarin Pradhan and Srishtee Gurung, Madan Puraskar Pustakalaya, Nepal.

4 GNU/Linux and Fonts

4.1 Introduction

In Chapter 1, we briefly discussed fonts. In this Chapter, we will further deal with different types of fonts, font systems in GNU/Linux and general instructions for installing fonts. We also list the font resources and font development tools. Lastly, we deal with Openoffice.Org fonts. As evident from the discussion in Chapter 1, the primary prerequisites for Localization are encoding of the script of the language in Unicode, development of Unicode compatible fonts etc. Basically fonts are classified as: 8 bit True Type Font, which is limited to 256 glyphs out of which only 200 are usable,16 bit Unicode true type font covering Unicode range supporting 65000 glyphs, Open Type for advanced typography in which GSUB, GPOS tables are used for complex scripts like Indic.

4.2 Types of fonts

In Chapter 1, we briefly talked about Bitmap and Vector fonts. Here we add a few others in the category of the two fonts and briefly introduce them [4.9.b].

BitMap fonts
These are matrices of dots. Two types of bitmap fonts exist:

a) bitmap printer fonts (eg: pk)
b) bitmap screen fonts for use in X windows and console (eg: bdf, pcf)

Vector/Outline fonts

a) Type1 fonts
These fonts are devised by adobe and are supported by adobe's postscript standard. Distributed as: afm (adobe font metric) or pfm (postscript fonts for windows) and outline file as pfb (printer font binary) or pfa (printer font ascii).The outline file contains all the glyphs and the metric file contains the metrics.

b) Type 3 fonts
These fonts are distributed similar to type 1 but are not supported by X. They are only supported by the Postscript standard.

c) Type 42 fonts
These fonts are the same as true type fonts in addition to the headers that enable them to be rendered by a postscript interpreter.

d) Open Type Fonts
OTF font format is an extension of TTF adding support for postscript font data. OTF is developed jointly by Microsoft and Adobe. As withTTF fonts, OTF fonts allow the handling of large glyph sets using Unicode encoding.

4.3 Font Systems in GNU/Linux

Xfree86 includes two independent font systems:

a) Core X11 font system
This system previously could only handle bitmap fonts but it can now support scalable fonts like Type1, Speedo, TrueType and OpenType. Xfree86 has a font path and font servers where it searches for fonts.

Example:
Font path: In file /etc/X11/XF86Config-4 or /etc/X11/xorg.conf file and written as:
FontPath “/usr/lib/X11/fonts/misc”
Font Server: In file /etc/X11/XF86Config-4 or /etc/X11/xorg.conf and written as
FontPath “Unix/:7100”
which looks for the path /etc/X11/fs/config file.

b) X freetype interface library for font system ie Xft

Xft provides client side font API for X applications. They were written to provide X applications witha convenient interface to the FreeType font rastarizer and X rendering extension.It uses fontconfig library to select fonts and X protocol to render them.

4.4 Installing Fonts

The following section looks at the instructions required for installing fonts in the three different font systems, X11 core fonts system, X font server and Xft font system respectively.

Using the X11 core fonts system

a) mkdir /usr/local/share/fonts/truetype
b) copy fonts to the directory just created by the above command
c) mkfontscale /usr/local/share/fonts/truetype
d) mkfontdir /usr/local/share/fonts/truetype

Here are the commands,
mkfontdir creates fonts.dir file making the current directory a font directory.
mkfontscale is used for indexing all the fonts using font.scale file.
mkfontdir cannot recognize scalable fonts without indexing so this should be used first.

Using X font server

a) Add font path to /etc/X11/fs/config
b) ii) Restart font server

Using Xft font system

Fontconfig installs fonts in a set of well known directories including Xfree86 directories and also in theuser's home directory as ~/.fonts. Run fc-cache after copying fonts. You can use fc-list to view all the installed usable fonts.

4.5 Font Selection

When an application requests a particular font, X system searches in all of the directories in the font path one at a time until it finds the best match for it. It selects true type fonts over bitmap and unscalable bitmaps over bitmaps and so on.

4.6 Font Resources

Below we have listed the links to some useful font resources.

a) Free fonts
   http://cgm.cs.mcgill.ca/~luc/fonts.html
b) TrueType Indic fonts from NCST (indix)
   http://www.ncst.ernet.in/projects/indix/download.shtml
c) OpenType Unicode Indic fonts for Debian GNU/Linux
   http://packages.debian.org/testing/x11/ttf-indic-fonts

4.7 Font Development tools

Below we have listed the links to some of the useful font development tools.

a) Fontforge (http://fontforge.sourceforge.net)
b) GNU font editor (http://www.gnu.org/software/gfe/gfe.html)
c) Fontlab (http://www.fontlab.com)

4.8 Openoffice.Org Fonts

Next we look at the basics required for dealing with fonts in Openoffice.Org.

Changing the user interface font [4.9.a]:

To change the user interface font:
-go to Tools -> Options -> OpenOffice.org -> Fonts
-check the 'Apply Replacement Table' check box
-replace the font Andale Sans UI with the desired font

Note: You will probably have to write 'Andale Sans UI' manually into the 'Font' field.

- please choose one of the fonts from the dropdown list box for the 'Replace with' entry and press the 'OK' button.

Installing Fonts [4.9.b]
The number of fonts vary in OpenOffice.org depending on the type of the document being used. The reason being that not all of the fonts can be used in every case. In the case of HTML document or in online layout, only those fonts that are available on the screen are offered whereas in the case of text document, only those fonts are shown which can also be printed. On the other hand, in the case of spreadsheets and drawings, all of the fonts that can be either printed or shown on the screen can be used.

a) Adding Fonts
In order to integrate additional fonts in OpenOffice.org, the following steps need to be performed:
1. Go to the <OOo_install_path>/program directory and start spadmin by entering ./spadmin
2. Click Fonts
3. The dialog lists all the fonts added for OpenOffice.org. The fonts can be added with the Add button whereas they can be removed with the Remove button.
4. When you click Add, the Add Fonts dialog will appear.
5. Enter the directory from which you want to add the fonts by pressing the ... button and selecting the directory from the path selection dialog or by entering the directory directly.
6. Select the fonts you want to add from the list of fonts that appears from this directory. In order to add all the fonts, click Select All.
7. You can either copy the fonts in the OpenOffice.org directory (for cases when data medium is not always available such as CD-ROM, the fonts must be copied) or create only the symbolic links with the Create soft link only check box.
8. By clicking OK, the fonts will be added.

b) Deleting fonts
In order to delete fonts, the following steps have to be performed:
1. Start spadmin as mentioned in step 1 of 'Adding Fonts'
2. Click Fonts.
3. Select the fonts you want to delete from the list that will appear from the dialog box out of all the fonts that are added to OpenOffice.org and click Delete.

Note: You can delete only the fonts that have been added for OpenOffice.org.

c) Renaming Fonts
In order to rename the fonts that have been added for OpenOffice.org, the following steps have to be performed:
1. Start spadmin as mentioned in step 1 of 'Adding Fonts'
2. Click Fonts.
3. Select the fonts that you want to rename and click Rename.
4. Enter a new name in the dialog that appears. In cases where the font contains several names, these names will come up as suggestions in the combo box where you can enter the new name.
5. Click OK.

Note: This is especially useful for fonts that contain several localized names.

In order to rename several fonts, one dialog appears for each selected font.

4.9 References for Further Reading

a) http://www.openoffice.org/FAQs/fontguide.html
b) http://documentation.openoffice.org/online_help/htmlhelp/text/shared/guide/spadmin.html
c) http://nepalinux.org/docs/l10nhowtoguide.pdf

5 Input Methods for Linux

5.1 Introduction

In Chapter 1, we briefly introduced Input Methods. We discussed the crucial parts of the design and implementation of the Input Method. In this chapter, we will concentrate on the three different input methods available for Linux, viz., xkb, iiimf and scim. References to some important links on input methods are provided at the end of the chapter for further reading.

5.2 Different Types of Input Methods

5.2.1 xkb Keyboard Layout for X11

The X Window System used on most Unix-like systems today uses X Keyboard Extension (xkb) for translating keystrokes into character codes. While creating the xkb layout for a particular language, we have to map symbols according to the layout of the keyboard as illustrated below. This input method can be used in any type of applications that are based on X as Gtk based applications , Qt based applications etc.

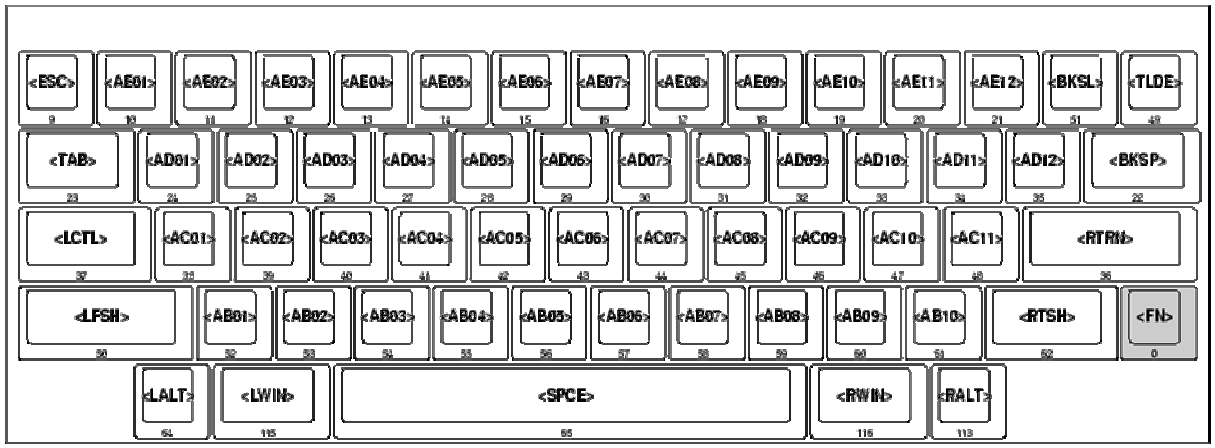

Figure 1: Keyboard layout for mapping the symbols

Steps to create a new layout

This section will guide you on how to create a new xkb input.Execute the steps below to create a new keyboard layout.

a) Create a file named using your language code. For eg: ne in case of Nepali Language

b) The content of the file would be similar to the one below:

```
partial default alphanumeric_keys
xkb_symbols "basic" {
    name[Group1]= "FullLanguageName";

    key <AD01> { [      0x100091F,  0x1000920        ]      };

 };
```

This defines that key AD01 ie Q key has 2 letters (the second letter can be used by pressing the shift key while typing) which are denoted by their respective Unicode values. Like wise for every key you need to assign the Unicode values. Refer to the above picture for creating the layout.

c) Save it and copy it to /usr/X11R6/lib/X11/xkb/symbols/pc and /usr/X11R6/lib/X11/xkb/symbols/.

Then, add the following lines to /usr/X11R6/lib/X11/xkb/rules/xorg.xml or

/usr/X11R6/lib/X11/xkb/rules/xfree86.xml file.

```
<layout>
<configItem>
<name>xx</name>
<description>yy</description>
</configItem>
<variantList/>
</layout>
```

Where, xx = language code and yy = Any Description

Note that in xkb only one to one keystrokes and character mappings can be achieved but not the combination of three characters to a single keystroke. Instead, use scim or iiimf for these kinds of purposes.

d) Using xkb

On the top panel:

1. Right click the mouse
2. Click "Add to Panel"
3. Click "Keyboard Layout Indicator"
4. Click the "ADD" button
5. Right click the added icon
6. Click "Open keyboard preferences"
7. Click the "Layout" tab
8. Select "FullLanguageName" from available layouts
9. Click "ADD and Close"
10. Open any text editor applications and type

5.2.2 IIIMF

IIIMF is the framework that is directed towards platform independence when entering input texts. It has three major components[5.3.d]:

a) IIIMP : IIIM protocol is a platform independent protocol, window system independent and language independent.

b) IIIMCF: IIIM client framework uses IIIMP to access input methods of the IIIM server. This is independent of the operating system.

c) c)IIIMSF: IIIM server framework acts as agent and provides the IM services to IIIMCF using IIIMP

IIIMF can handle one to many symbols mapping, i.e a single keystroke can produce a compound character.

Creating a keyboard layout for IIIMF

a) Install iiimf binary and development packages using source files or using distribution packages. The following example will show you how to install iiimf in Debian GNU/Linux system

Run as root : apt-get update && apt-get install iiimf*

b) The input methods are stored in /usr/lib/im/locale. There are directories for different locales, there is one called UNIT - which is a Unicode table based input method. It is similar to keymaps, but is more flexible in the sense that you can assign any string of characters to any sequence of keystrokes. Inside UNIT dir , you will see directories for languages which have files like HINDI/data/inscript.data, GUJARATI/data/inscript.data etc. In this directory, create a new folder with your language name as: NEPALI, HINDI. Shift to the newly created directory and create a directory named data in it.

c) Create a file named myinput.txt . In this file we will do the actual key mapping. The file should look like the following:

```
----------------------------------------------------------------------
## HANZI codetable input table

[ Description ]
Locale Name:            Your Locale Name
Layout Name:            YourLayoutName
Encode:                 UTF-8
UsedCodes:
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890!@#$%^&*()_-
+=|\~`:;"'{[}]<,>.?/

WildChar:
MaxCodes:       1

[ Function_Key ]
PageUp
PageDown
BackSpace

[ Options ]
KeyByKey_Mode:              ON
HelpInfo_Mode:              ON
AutoSelect_Mode:            ON
KeyPrompt_Mode:             ON
SelectKey_Mode:             Number

[ Single ]
s         x

## where x = a single character typed by your desired input method.

[ Phrase ]
q         y

## where y = a  character formed by pressing two or more than two characters using desired input
method
----------------------------------------------------------------
```

Note that in the above file:

Section [Single] is a mapping of single keys to single characters.
Section [Phrase] is a mapping of a single key to multiple characters (conjunctsmay be inserted)

d) Save the file inside the folder UNIT/common. There are binaries bin2txt and txt2bin. Run it as follows (assuming you are in /usr/lib/im/locale/UNIT/common).

```
# ./txt2bin myinput.txt ../YOURLANGUAGENAME/data/myinput.data
```

e) 5.Then add an entry in UNIT/sysime.cfg for your language , like

```
[ xx_XX ]
myinput common/ctim.so YOURLANGUAGENAME
```

Using IIIMF input

a) Open xterm
b) Export GTK_IM_MODULE=iiim

c) Run gedit , swriter etc
d) Press ctrl +o and Type

5.2.3 SCIM

Smart Common Input Method platform, SCIM, provides user friendly, full featured input method user interface for POSIX-style operating systems (including Linux, FreeBSD and other Unix). SCIM also is a development platform to make input method development easier [5.3.c]. SCIM is highly modular. It consists of 4 basic modules, each provided by different packages. Listed below are the package names and their brief module descriptions [5.3.e].

a) **SCIM**
SCIM is the core package, which provides the fundamental routines and data types. This package contains the main binary "scim" and other support programs. it provides a common platform for various modules to be plugged in. It also includes a set of programs and modules of its own.

b) **SCIM-GTK2-immodule**
This is the GTK+2 input method module with scim as backend. This means, if GTK_IM_MODULE is set to use scim, this package is responsible for making gtk+ application(eg. gedit, firefox) to use scim as input module by default. This input method should work within all GTK+ 2.x platforms, including gtk-x11.

c) **SCIM-modules-socket**
This package provides the socket modules for SCIM.

d) **SCIM-tables-additional**
This package contains data tables for several languages.

Steps for SCIM installation in Debian GNU/Linux

1. Install SCIM core packages
#apt-get update
#apt-get install scim
#apt-get install scim-gtk2-immodule
(Note: This will also install package scim-modules-socket)
#apt-get install scim-modules-tables
(Note: The above package provides "scim-make-table" binary which we will use in the example below to create a new input method data table)

2. **Creating a new Input Module Data Table**
Create a file named myinput.txt. The fileshould look like this and can be divided into 2 sections:

- TABLE DEFINITION: Entries like table name, language code, locale & author name are defined.
- TABLE DATA : Character mapping is done in this section

```
------------------------------------------------------------
### File header must not be modified
### This file must be encoded into UTF-8.
### This files tries to implement the Traditional
### keyboard layout modified by MPP for PAN Project
SCIM_Generic_Table_Phrase_Library_TEXT
VERSION_1_0

### Begin Table definition.
BEGIN_DEFINITION

### An unique id to distinguish this table among others.
### Use uuidgen to generate this kind of id.
UUID = 16f49d28-677b-4ac7-a93c-9f714b070a5a

### A unique number indicates the version of this file.
### For example the last modified date of this file.
### This number must be less than 2^32.
SERIAL_NUMBER = 20051103

ICON = /usr/share/scim/icons/Nepali.png
```

```
### The default name of this table
NAME = Traditional

### The local names of this table
NAME.ne_NP = ट्रेडिस्नल

### Supported languages of this table
LANGUAGES = ne_NP

### The author of this table
AUTHOR = Harkhe <harkhe@gmail.com>

### Prompt string to be displayed in the status area.
STATUS_PROMPT = NP

### If true then the first candidate phrase
### will be selected automatically during inputing.
AUTO_SELECT = TRUE

### If true then a multi wildcard will be appended
### at the end of inputing string automatically.
AUTO_WILDCARD = FALSE

### If true then the result string will be committed to client automatically.
### This should be used with AUTO_SELECT = TRUE.
AUTO_COMMIT = TRUE

### If true then the inputed string will be automatically splitted during inputing.
AUTO_SPLIT = FALSE

### If true then the phrases' frequencies will be adjusted dynamically.
DYNAMIC_ADJUST = FALSE

### If true then the preedit area will be filled up by the current candidate phrase automatically.
AUTO_FILL = FALSE

### If true then the lookup table will always be shown if there is any candidate phrase.
### Otherwise the lookup table won't be shown unless the user requires it by moving the pre-edit caret
left.
ALWAYS_SHOW_LOOKUP = FALSE

### Enable full width punctuation property
USE_FULL_WIDTH_PUNCT = FALSE

### Use full width punctuation by default
DEF_FULL_WIDTH_PUNCT = FALSE

### Enable full width letter property
USE_FULL_WIDTH_LETTER = FALSE

### Use full width letter by default
DEF_FULL_WIDTH_LETTER = FALSE

### The maxmium length of a key.
MAX_KEY_LENGTH = 1

### Valid input chars.
VALID_INPUT_CHARS =
```

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890!@#$%^&*()_-+=|\~`:;'"{[}]<,>.?/

Single wildcard char, can have multiple chars.
SINGLE_WILDCARD_CHAR = ?

Multi wildcard char.
MULTI_WILDCARD_CHAR = *

The key strokes to split input string.
SPLIT_KEYS = quoteright

The key strokes to commit the convert result to client.
COMMIT_KEYS = space

The key strokes to forward the inputed string to client.
FORWARD_KEYS = Return

The key strokes to select candidiate phrases.
SELECT_KEYS = 1,2,3,4,5,6,7,8,9

The key strokes to page up the lookup table.
PAGE_UP_KEYS = Page_Up

The key strokes to page down the lookup table.
PAGE_DOWN_KEYS = Page_Down

END_DEFINITION

Begin Table data.
BEGIN_TABLE
" ੍
ਘ
% ਛ
^ ਟ
& ਠ
* ਭ
' ੁ
(ਢ
) ਣ
+ ਂ
= 0x200c
, ਸ
- ਔ
. ।
/ ਰ
0 ੦
1 ੧
2 ੨
3 ੩
4 ੪

5 ५
6 ६
7 ७
8 ८
9 ९
; स
< ङ
? रु
@ ई
A आ
B ौ
C ऋ
E ऐ
F ँ
H झ
J ो
K फ
L ी
O इ
P ए
U ऊ
V ॐ
\ ्
] े
_ ओ
` ञ
a ब
b द
c अ
d म
e भ
f ा
g न
h ज
i ष
j व
k प
l ि
m ः
n ल
o य

p ठ
r च
s क
t त
u ग
v ख
w ध
x ह
y थ
z श
{ ृ
| 0x200d
} ै
~ ॥
PHRASES
! ज्ञ
$ द्ध
: ट्ठ
> श्र
D ङ्ग
G द्द
I क्ष
M ड्ड
N द्य
Q त्त
R द्ब
S ङ्क
T ट्ट
W ड्ढ
X ह्य
Y ठ्ठ
Z क्क
[र्
q त्र
END_TABLE

---

3. Converting the table into binary format and installing

scim-make-table myinput.txt -b -o myinput.bin
#mkdir -p /usr/share/scim/tables
#cp myinput.bin /usr/share/scim/tables/

Please refer to following page for more information on creating new table data.
http://www.scim-im.org/development/contribute/how_to_create_a_new_ime_in_about_15_minutes_with_scim_and_scim_tables

4. Installing already available Input Method data tables

If the input method data table for your language has already been uploaded into scim upstream and is available in debian, you can simply install the package as:

#apt-get install scim-tables-additional

This package contains IM data tables for non CJK languages. Currently it supports Arabic, Nepali, Russian, Thai, Vietnamese, Bengali, Gujrati, Hindi etc.

Using SCIM

Create a file /etc/X11/Xsession.d/95scim_start and export following environment.

#touch /etc/X11/Xsession.d/95scim_start
#vi /etc/X11/Xsession.d/95scim_start

---
export XMODIFIERS=@im=SCIM
export GTK_IM_MODULE=xim
/usr/bin/scim -d

Execute any program and hit cntrl+space to activate scim input.

5.3 References for Further Reading

a) http://hektor.umcs.lublin.pl/~mikosmul/computing/articles/custom-keyboard-layouts-xkb.html
b) http://www.charvolant.org/~doug/xkb/html/xkb.html
c) http://www.openi18n.org/subgroups/im/**iiim**f/whitepaper/whitepaper.html
d) http://packages.debian.org/unstable/utils/scim-gtk2-immodule
e) http://www.scim-im.org

6 Translation Aspects in Localization

6.1 Introduction

In this Chapter, we deal with an important part i.e.the translation aspects of Localization. We will give a brief overview of translation then discuss the requirements of the translation manager which covers Concurrent Versioning System (CVS), translation tools, PO file conversion tools etc. Later, we move to the essence of Glossary Development for translation and translation process management. In the translation process management, we will discuss issues like team formation, orientation and training of the translation team, testing and verification etc., References to links for further reading are given at the end of the chapter.

6.2 Translation Overview

Translation is a very important aspect of localization. This involves translating messages in programs, including menus, dialog boxes, button labels, error messages etc thus being an immediate means for bringing the software to the local users, who are not familiar with English. This task, can be done once the output methods and fonts are ready, otherwise the translated messages become useless. Input methods, if available, also help ease the translation process.

There are many message translation frameworks available, but the general concepts are the same. Messages are extracted into a working file to be translated and compiled into a hash table. When the program executes, it loads the appropriate translation data as per locale. Messages are quickly looked up for translated version for use in the outputs.

Translation is a labor-intensive task. It takes time to translate the huge number of messages. Hence, it is always done by a group of people. When forming a team, make sure all members are using consistent languages over all parts of the programs. It is a good idea to work together in a closely discussed forum and build the glossary database collected from the settled decisions. Sometimes, however, you will need to run the program to see the context surrounding the message in question to find proper translation, or investigate the source code in case of conditional messages, such as error messages. Literally translating message by message without running the program can often result in incomprehensible messages.

Like other FOSS development activities, translation is a long term commitment. New messages are usually introduced in every new version. Even though you have completed all messages in the current version, be sure to check for any new messages again in the new release. It is important to note that there is usually a string freeze period before the final release, when no new strings are allowed in the code base, and hence the time slot is allocated for translators[6.5.a].

6.3 Requirements of the Translation Manager

The next section discusses requirements a Translation Manager would need to manage the translation process.

6.3.1 Concurrent Versioning System

The URL link en.wikipedia.org/wiki/CVS defines CVS as the Concurrent Versions System (CVS). It is also known as the Concurrent Versioning System. The CVS implements a version control system: it keeps track of all work and all changes in a set of files, typically the implementation of a software project, and allows several (potentially widely separated) developers to collaborate. CVS has become popular in the open-source world. CVS is released under the GNU General Public License.

An important component of Source Configuration Management (SCM), it has a similar role to the free software RCS, PRCS, and Aegis packages. While CVS stores individual file history in the same format as RCS, it offers the following significant advantages over RCS[6.5.b]:

* It can run scripts which you can supply to log CVS operations or enforce site-specific polices.

* Client/server CVS enables developers scattered by geography or slow modems to function as a single team. The version history is stored on a single central server and the client machines have a copy of all the files that the developers are working on. Therefore, the network between the client and the server must be

up to performing CVS operations (such as checkins or updates) but need not edit or manipulate the current versions of the files. Clients can perform all the same operations which are available locally.

* In cases where several developers or teams want to maintain their own version of the files, because of geography and/or policy, CVS's vendor branches can import a version from another team (even if they don't use CVS), and then CVS can merge the changes from the vendor branch with the latest files if desired.

* Unreserved checkouts, allowing more than one developer to work on the same files at the same time.

* CVS provides a flexible modules database that provides a symbolic mapping of names to components of a larger software distribution. It applies names to collections of directories and files. A single command can manipulate the entire collection.

* CVS servers run on most unix variants, and clients for Windows NT/95, OS/2 and VMS are also available. CVS will also operate in what is sometimes called server mode against local repositories on Windows 95/NT.

For getting acquainted with basic CVS installation and commands, please refer to “Essential CVS, Jennifer Vesperamen” available at www.oreilly.com/catalog/**cvs**/

From the above, it is understandable that CVS installation and usage is the first requirement for any translation manager. With the help of CVS, multiple translators from the translation team may check in and check out from a single central CVS repository devised for translation files directory. Logs and revision history of individual files can be well maintained. This helps to keep track of the actual amount of work done by each translator. In addition to this, the work distribution can also be managed and monitored efficiently. In case of file loss in the local machines, considerable amounts of work as submitted in the previous submissions can be retrieved. This is extremely important as there is always the risk of file or data loss because of system failure or unexpected crashes.

6.3.2 Translation Tools

Several translation tools for all platforms, either windows, linux or others are available on the web. Some of them can be downloaded for free while others demand some amount of money to be paid for downloading. Free open source translation tools relate to the first category while proprietory translation tools relate to the latter catagory.

A list of translation tools and their respective characteristics is given below:

S.No.	Tools	free/non-free	Licensed Under	Online/Offline	Plateform dependency
1.	**Pootle**	Free	GNU GPL	Online	N/A
2.	**Rosetta**	Non-free	-	Online	N/A
3.	**Kartouche**	Free	GNU GPL	Online	N/A
4.	**KBabel**	Free	GNU GPL	Offline	Widows/Linux
5.	**poEdit**	Free	-	Offline	Widows/Linux
6.	**Attesoro**	Free	GNU GPL	Offline	Linux
7.	**passolo**	Free	GNU GPL	Offline	Windows
8.	**IniTranslator**	Free	GNU GPL	Offline	Windows
9.	**GTranslator**	Free	GNU GPL	Offline	Linux
10.	**LocFactoryEditor**	Free	GNU GPL	Offline	Mac OS

Table 1. List of translation tools and their respective characteristics

Source: http://www.i18nguy.com/TranslationTools.html

From the table above, Kbabel, Gtranslator, poEdit, LocFactoryEditor are po file editors. For more information on the po file format, please refer to the “po file format” section of this guide. Besides, we can also use the text editor tools, openoffice and gedit as translation tools for small translation works.

In the following section, we discuss different features of Kbabel as a translation tool. Kbabel is being used as

a translation tool for the localization works at the Madan Puraskar Pustakalaya, Nepal, mainly because it is OpenSource and furnished with a rich set of features essential for the translators.

Translating with Kbabel
Kbabel has different features that assist the translation work[6.5.f]:

1) User-friendly user interface;
2) Automatically searches fuzzy or translated strings;
3) Suggests a list of possible translations for a string;
4) Performs checks on syntax and spell check thus maintaining the format of the file;
5) Handles Unicode encoded files without any problem;
6) Help files on using individual features of Kbabel are available;
7) Statistics of translated messages,untranslated messages, fuzzy translations are displayed on the status bar;
8) Keeps track of the modifications incurred in the file thus keeping the record of the last modification date;
9) Powerful navigational features allowing the translator to move forward or backward or even to a particular string;
10) Drag and drop support;
11) Font configuration support for the message editor;
12) Tips or aids as comments for contextual translation of the strings;
13) Supports GNU gettext tool for PO files (including plural forms) and Qt Linguist catalogs;
14) Multiple views of the same file possible;
15) Spell checking facility available;
16) Syntax highlighting;
17) Word count facility;
18) Automatic file header updates;
19) Automatic translation generation by the system on the basis of the database formed by the system out of already translated terms, known as “Fuzzy Translation”;
20) Support for easy insertion of tags and URLs;
21) Validation and highlighting of tags and XML entities;
22) Automatic syntax check with msgfmt when saving and in cases of errors;
23) Has inbuilt support for running CVS operations.

Among the several features of Kbabel listed above, the automatic translation generation needs explaining. Kbabel maintains a dictionary of the already translated terms in English with their counterpart translations in the target language.While doing so, it scans all files and folders or directories which contain the translated strings. As a result of the scan, a dictionary of translated terms is maintained. Kbabel, with the help of this dictionary, facilitates rough or exact translation thus minimizing the time required for translation.The translation obtained by such means could be word-to-word rather than being contextual and at times require some post-editing .

Catalogue Manager
Catalog manager is yet another facility within Kbabel. It lists all the PO files along with their stastistical and other properties. On the basis of the information displayed regarding the files, the translator can then decide whether further work on the respective files is required or not.

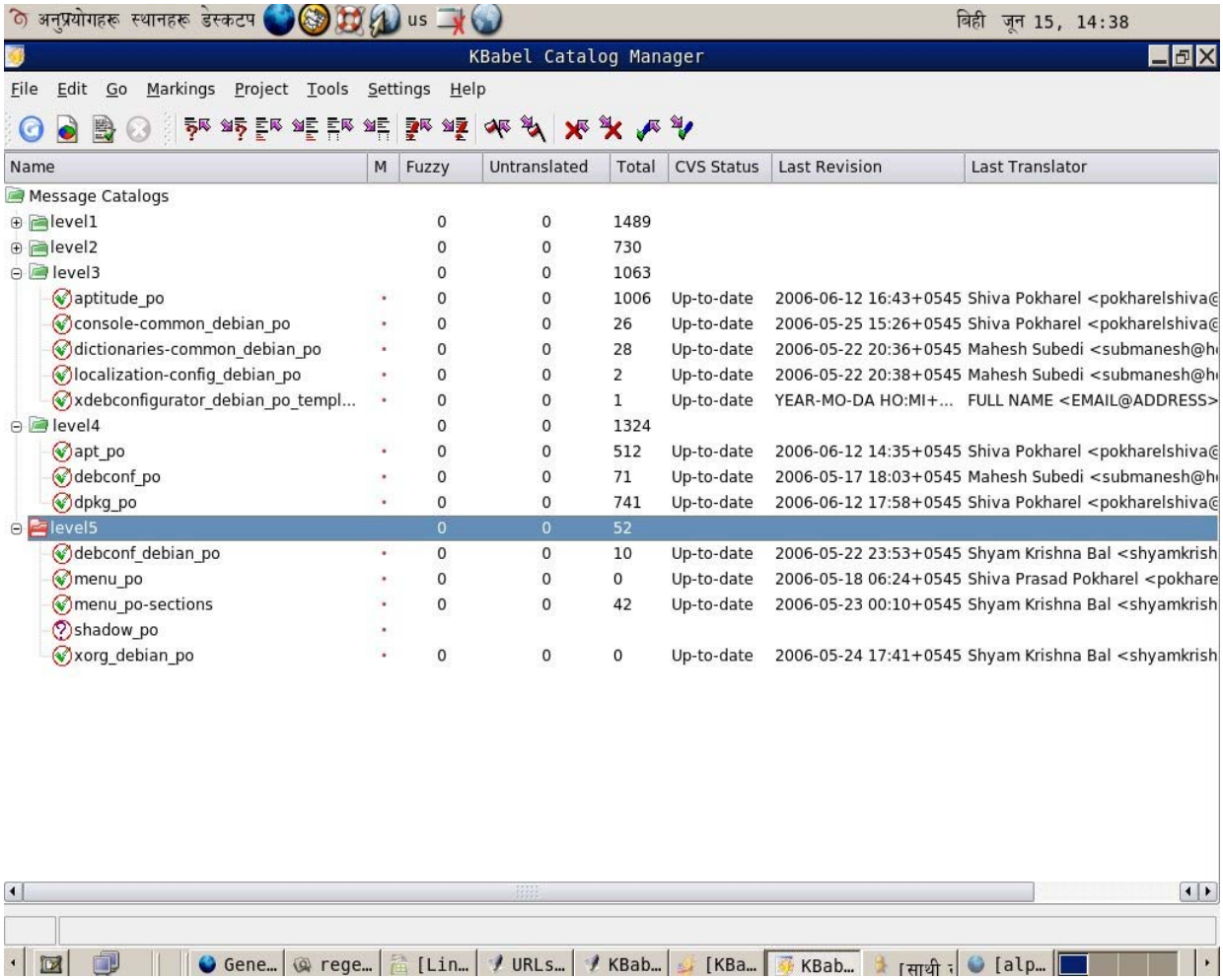

Figure 2. Catalogue manager in KBabel

Project Settings

This is yet another important feature of Kbabel.With the help of this informative window, we can set the path of the PO and POT files. One just needs to choose Project-> Configure from the main menu. In the window that pops up after having chosen Project-> Configure, we may set several attributes like identity of the translator,his/her email id, name of the target language, plural forms of translation etc.

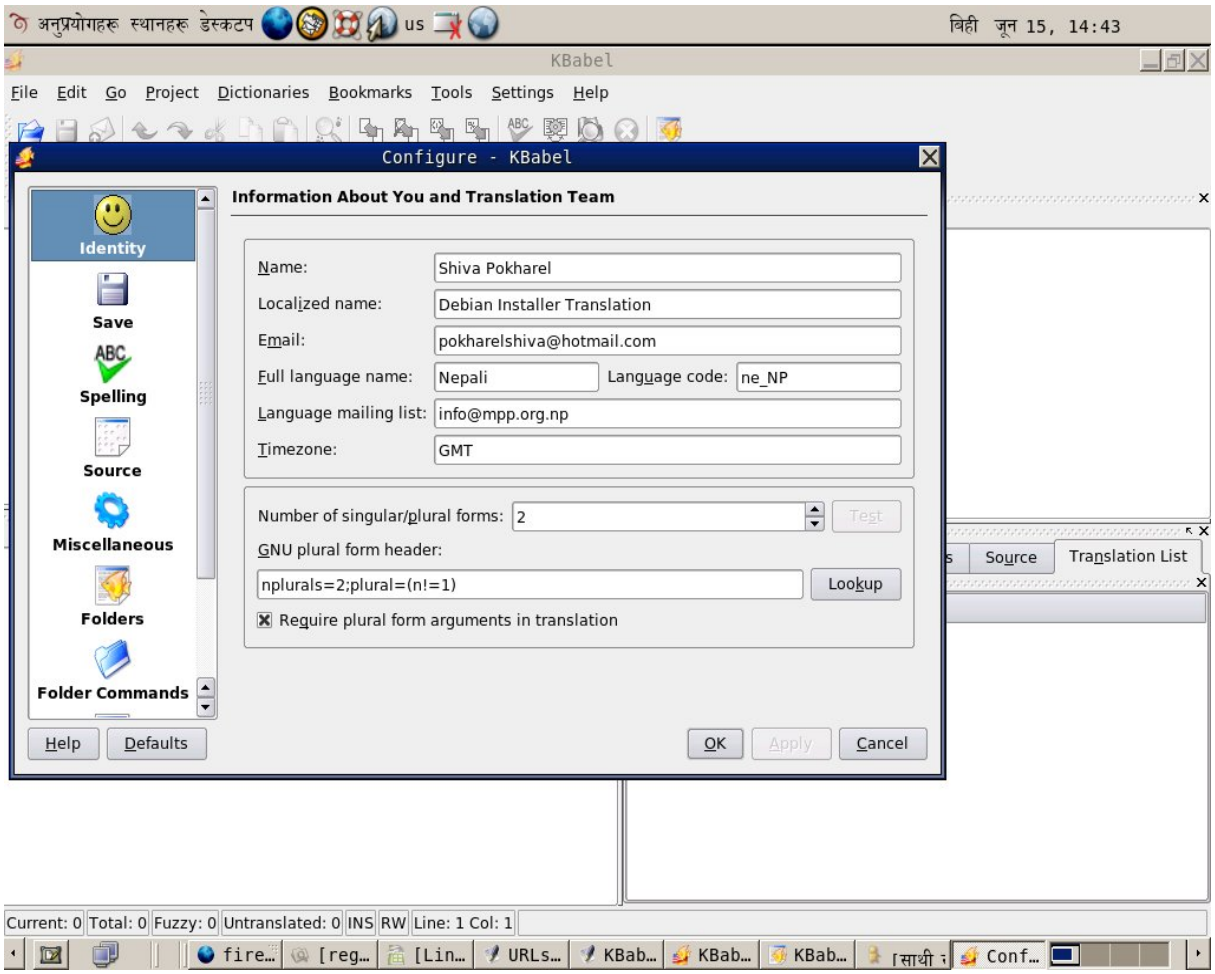

Figure 3. Project settings in KBabel

There is also a timezone field to track your "last modified" time for PO files. You can specify it as character sequence like GMT time. This information is used when updating file headers. You can find the options that control what fields in the header should be updated in the Save section of the Preferences dialog.

Number of singular/plural forms in Kbabel translation:
For GNU gettext tool, plural forms may be formulated as below:
nplurals=2;plural=(n!=1)
For KDE, number of plural forms could be 1 or 2 or it could even be 3 according to the patricular language pattern.

Saving and updating your translated work in Kbabel
This lets you set the save options for files, auto save time setting, header updates etc. Among these, update of header in PO file is the crucial part when working in CVS .

Update header when saving
Check this button, to update the header information of the file every time it is saved. The header normally keeps information about the date and time the file was last updated, the last translator, who worked on the file etc. You can choose which information you would want to update from the Fields to update by checking the checkboxes available. If you want to add additional fields to the header, you can edit the header manually by choosing Edit->Edit Header in the editor window.

Checking the syntax of the file when saving
Check this to automatically check syntax of file with msgfmt tool when saving a file. You should keep this

validation enabled unless you know what you are doing.

If you do not want to touch some fields in a PO file header or want to force updating of specific fields, there are five checkboxes which control this: revision date, PO file language, text encoding, last translator name, charset. If a field does not exist, it is appended to the header. If you want to add other information to the header, you have to edit the header manually by choosing Edit->Edit Header in the editor window. Deactivate Update header when saving above if you don't want to have the header updated.

For date and time of the header field PO-Revision-Date, you can choose one of the formats: Default, Local, Custom.

Warning
An error generation is very certain if the same file is tried for getting access simultaneously by more than one translator.Further, if attempted to multiple commit to a single file, it will result in a conflict. Hence it is recommended that one person works exclusively on a single file. The work could also be locally saved in the local computer. Later through a user-friendly interface, the locally saved copy may be committed to the CVS server.

Important
You should keep the default setting to Default. The two other settings make the generated PO file, not a standard GNU gettext PO file , so this should be avoided.

Default is the format normally used in PO files.
Locale is the format specific to your country.
Custom lets you define your own format, where you can use the following format strings:

Year Setting		
%y	ranges between	00 to 99
%Y	ranges between	0001 to 9999
Month Setting		
%m	sets months as	01 to 12 format
%f	sets months as	1 to 12 format
%b,%h	sets months as	Jan to Dec format
Day Setting		
%j	is for day of the year setting in	001 to 366 format
%d	is for day of the month setting in	01 to 31 format
%e	is for day of the month setting in	1 to 31 format
%a	is for weekday abbreviation setting in	Sun to Sat format
Hour Setting		
%H	sets hours as	00 to 23 format
%k	sets hours as	0 to 23 format
%i	sets hours as	1 to 12 format
%I	sets hours as	01 to 12 format
%p	is for AM or PM setting	
Minute, Second, Timezone setting		
%M	for setting minutes	00 to 59 format
%S	for setting minutes	00 to 59 format
%Z	for setting timezone	(given in identity settings)
%z	for setting timezone	(numeric offset as specified

by system settings

The lower group covers encoding options for PO files when saving. If you work on the KDE project you should be aware that at least PO files must be UTF-8 encoded in KDE. Alternatively you can select the encoding corresponding to your locale. If, for some reason, you do not want to accidentally change the current PO file encoding, turn on “Keep the encoding of the file.”

Remember
The encoding correspondig to your locale might not be suitable sometimes. So KBabel may not be able to handle them. But UTF-8 is always supported by GNU gettext.

Spell Check
With this feature enabled, spell checking is possible in Kbabel. This also aids the translators while translating. Note that you must install an appropriate dictionary for your language. Check your ispell or aspell distribution to find out if you have one.

CVS operations from within KBabel
As listed in the general features of Kbabel, another outstanding feature is the inbuilt support for CVS operations.Once the working copy of the translation files is checked out and the path to the directory of files pointed in the “Project Settings”, by right clicking on any of the files in the Catalogue Manager Window, one will generally see the contextual menu as shown above. But while saying so, it is assumed that both the CVS server and client application is installed respectively in the server machine and the local machine of the translator.Kbabel has the inbuilt support for popular CVS operations like update and commit. In order to bring into effect the changes made in the working copy of the file to the one in the CVS repository, we first run the command “update” and then “commit”. On the consecutive launching of the application Kbabel, the updated information on the files would be depicted in the catalogue manager.

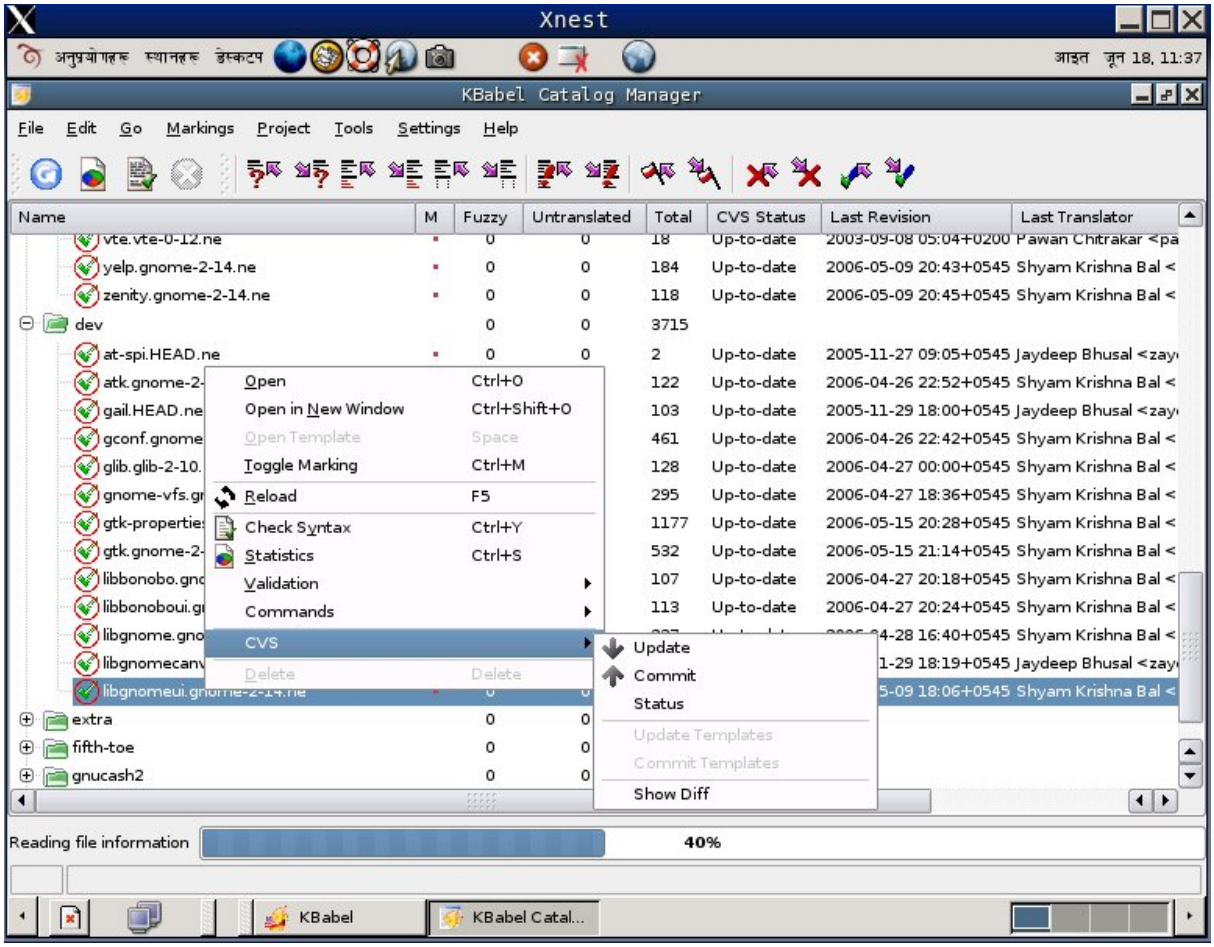

Figure 4. CVS operations from within KBabel

6.3.3 PO File Format

The PO File is Portable Object file where the translatable strings are placed. This file contains some information about the file and its author and contact information.

This file is divided into two parts, one being the header, where the information about file and author, copyright,language and team information is placed and other being message strings which is the one to be actually translated.

Every PO has a "header", containing the copyright information, charset, package name, etc. An example of a blank header is shown below.

```
# 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"
"POT-Creation-Date: 2003-05-01 13:15+0200\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"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"

Header for Nepali translation
Nepali Translation Project.
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.
#
msgid ""
msgstr ""
"Project-Id-Version: Gnome 2-10\n"
"POT-Creation-Date: 2006-02-01 13:15+0200\n"
"PO-Revision-Date: 2006-02-01 13:15+0200\n"
"Last-Translator: Jyotsana Shrestha <jyotsana@mpp.org.np>\n"
"Language-Team: Nepali <info@mpp.org.np>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"

After the header information, the actual message strings starts
It is paired with msgid and msgstr as an example shown below
#: src-/01/main.c:500
msgid "GNOME"
msgstr ""

The line #: src-/01/main.c:500 is the message id and the location where it occurs .

Line msgid "GNOME" is the main message to translate
Line msgstr "" is the translatedmessage.

so after translation this example will be

#: src-/01/main.c:500
msgid "GNOME"
msgstr "जिनोम"

Fuzzy translations
If the translator is not sure about the translation then s/he can mark the translated string as fuzzy so that the file when converted to machine readable MO Form ignores the strings being marked as fuzzy.
eg.

#: src-/01/main.c:500
#,fuzzy
msgid "GNOME 2-2"
msgstr "जिनोम 2-2"

Managing the version of PO Files
The version of the PO files must match with the version of the application itself . In order for the application interfaces to be displayed fully in the local language, the PO files and the applications versions should conform to each other.

For example, if one is doing the translation of Gedit application and if he picks the file for version 2.10 then he has to use this translation in the Gedit – 2.10 version.

Once the exact version of file obtained, the translation can be initiated.

Conversion of PO Files
Some applications do not provide PO files or they don't use gettext format for the message translation and localization. To work with such files, there is a translation tool named translation-toolkit which has the features for format conversion from different formats to PO format and viceversa.

For instance, the docbook xml file can be converted to PO format by using xml2po tool and then translated using PO translation tools like KBabel, after which the translated file may be converted back to xml format with the po2xml tool.

Likewise the translation tool contains different features for converting PO to other formats and other formats to PO.

Some of formats possible for conversion are as listed below:
ts - po
moz - po for mozilla suite formats
sdf - po for openoffice formats

6.3.4 Standard Glossary for Translation

One of the prime issues concerning translation in localization is the glossary development. Without the glossary development, translators would be facing a hard time in maintaining a consistent translation throughout the project as with multiple translators working in the team, multiple translation variants are bound to come up. In order to develop a glossary, first and foremost, widely used technical terms, most of them being computer related should be collected from different sources. Then the collected glossary terms have to be provided the translation. The translated glossary of terms has to be approved by an authorised body, preferably representing the Government. The approval should be preceeded by formal discussion and consultations among multiple stakeholders like the linguists, enterpreneurs, scientists, technical experts, grammarians and so on. While developing the glossary, it must be properly researched in the sense of providing the translation. This is to prevent changing the translation of a given term as frequent changes in the translation part of the glossary might result in inconsistent translation and confusion among the end-users of localized software. However, there has to be periodical addition of terms making the glossary development a continuous and an evolving process.

In the year 2005, a glossary of 2,300 technical terms was translated from English to Nepali. The initiative of this work was taken by Nepali Language in Information Technology steering committee (NLIT) formed under High Level Commission for IT for National Standardization, a body formed under the then His Majesty's Government, Ministry of Science and Technology, Nepal. It has already been felt that the existing glossary is insufficient in terms of its word coverage as works on software localization gather momentum. Necessary homework is currently being done by Madan Puraskar Pustakalaya for supplementing the already existing glossary. In addition, proposals for changing the translation currently being used in the glossary is also being worked out.

6.4 Translation Process Management

6.4.1 Forming the Translation Team

It might be surprising but experiences collected have shown that in under developed countries, where the translation profession has not taken much appeal among the people, it is a challenging job to hire competent and professional translators. Hiring competent translators for the software localization adds a few more challenges to the market as noted below:

a) It is not conventional translation and hence the knowledge of a general translator would not suffice. The translator to be involved in software localization should be aware of the computing terminologies used in the English language and also should know the equivalent counterpart in the target language.
b) The translator to be involved in software localization has to be somebody comfortable using software tools, preferably in the Linux Operating System. Besides he/she should also have some basic concepts on Linux commands to be able to work on the terminal.
c) Prior knowledge or basic concepts of Concurrent Versioning System (CVS) as a groupware for file sharing and revision control would be highly preferable.

As per the practical experiences collected, a combination of a qualified translator with all or at least some of the the technical expertise as listed above is a rare fact. Often potential candidates for translators are people from the computer science background with some interest and expertise in the target language but again people from this sphere are not interested in the translation profession. This profession is taken to be not matching their qualification which is indeed misleading.

Having said this, the only option left for managers is to compromise in the qualification and skills of the translators to be hired. Not surprisingly, instead of professional translators, the management ends up hiring enthusiasts with basic computer skills, a good grasp of English as well as satisfactory knowledge of the target language.

6.4.2 Human Resource Estimation

Yet another challenging job for the translation manager, as the exact number of human resource required for translation can hardly be objective. The stastical measure of the required translator may be achieved by looking at the number of strings required for translation and the average number of strings translated by a translator on an average per day. Experiences have shown that the increased number of translators does not necessarily help accomplish qualitative translation on time. Rather qualified and competent manpower, though small in number may effectively complete the work in time and without the necessity later on to spend several hours for verification. Hence the human resource estimation has to be subjective rather than objective.

6.4.3 Orientation and Training to the Translation Team

First and foremost, the translation team has to become familiar with the standard glossary of terminologies that they would be following throughout the translation process. This implies that the standard glossary is a pre-requisite for starting up any translation work for software localization in order to to produce consistent translation from all those involved in the translation process. Without a common standard, the translation may lack uniformity and consistency leading to unproductive work.

6.4.4 Orientation to the Translation Team Regarding Translation Guidelines

Translation is the most important aspect in localization. Metalanguage works or works that discuss language could be very difficult to translate. Comic texts can also be very difficult to translate. Translating the messages and menus in an application is a heavy budget process and inaccurate translation would result in making the application unusable in that particular language. In short, translation should be complete, grammatically correct and should be terminology consistent.

Below are some suggested guidelines which should be followed during the translation process[6.5.c,d,e]:

1. First of all find out if someone is already working on your language. If so, contact them to get assistance for translating the commonly used terms.
2. The translator should be very well acquainted with both the source and target languages.
3. The translator should also be aware of the context of translation. For example, in the sentence 'Sita made him a duck' , the word 'duck' can be translated only if the context is understood. Such issues will come up when you are translating multiple paragraphs of texts.
4. Join a mailing list. You can use this to discuss translation of difficult words.
5. It would be a good idea to create a website to tell people about your work, keep glossaries etc.
6. You should make sure that your translation consistently uses the same terms.
7. Maintaining a glossary of terms would ensure that you don't use multiple terms to refer to the same thing. As far as possible avoid creating new terms.
8. Try to find out a standard body for your language to get terms.
9. The gettext is a good package with tools for internationalisation, managing different versions of the application etc. If you do not know about gettext, it would be a good idea to learn about it from different sources.
10. Avoid word-to-word translation and try to perform sense-to-sense translation.This means that the translator should always bear in mind the intended meaning in the source language.
11. The translator should take care to produce the intended overall effect with the appropriate tone by making the right choice of words.
12. Be sure not to use terms that are jargon or slang.
13. Do not use terms that have several meanings.

14. Do not use the same words for different meanings.
15. Person and number should be retained wherever possible in translation so that singular does not change to plural and third person statements do not change to first or second person statements.
16. In some cases, a sentence would be highly compressed in English but would run into 2 or 3 sentences when translated. Such sentences need special attention. In such cases word-to-word translation would cause problems.
17. Be sure that the reference to menus and buttons (like "EDIT") matches the term used in the localized operating system.
18. Try to get the translation work reviewed by at least two translators independently.

6.4.5 Making the Translation Team Familiar with the Translation Environment

The newly recruited translation team members should be made familiar with the translation environment. For this, he/she has to be trained in the following:

a) General concepts of CVS;
b) Training with the translation tools;
c) Concept of the Standard Glossary to be followed;

Our experience has shown that a new translation team member will get trained best and be familiar with the system if he/she undergoes a two weeks to a month's training by translating some sample translation strings. Specific feedbacks from the lead translator or the translation team manager at this period will be highly beneficial.

6.4.6 Translation Monitoring and Tracking

Translation monitoring and tracking is indeed a tedious job for the translation manager. Periodical meetings may be used for updating the status of the translation activity. However, greater efficiency will be achieved by taking the help of the CVS and the translation tools like KBabel, which takes the log of the exact amount of work done by a perticular translator by date. The fact that translation would require close monitoring and tracking, it is advisable that the works be conducted in the same organization. Decisions of outsourcing may be very risky unless an on-line viewing and monitoring provision is maintained.

6.4.7 Testing and Verification

This process is about carefully scanning, detecting and implementing remedial measures to the errors found in the targeted text. However, it needs to be noted that soemtimes the source strings may be faulty themselves and could mislead translators to bring a completely different meaning in the text.

The testing and verification can be divided into two phases:

a) Phase I

In this phase, the following list of actions should be implemented:

1. Checking all the User Interface cases. It depends on the level of menus and sub menus we would be required to check.
2. Linguistic competence like correct spelling and writing grammatically correct sentences.
3. Ensuring whether all the interface elements have come out in the target language or not.
4. Target language translation not fitting into the UI elements (Menus, Sub Menus, Tool bar)
5. Reporting of bugs: All the anomalies or errors should be recorded in a file as bugs and submitted to the translation manager. A sample of the bug report form is shown below in table 2.

b) Phase II

A much more rigorous translation testing should be conducted at this phase. Any deviation from the conformance as dictated by the guidelines should immediately be reported to the translation manager.

GUI Application	String where the bug was found (ENG)	String where the bug was found (NEP)	Resolved (y/n)
gedit	Save this file	यो फाइललाई संग्रह गर…	
gedit	Columns to the left	स्तम्भ…..	

Table 2. Sample of the bug report form

List of Bugs
Given below is a list of bugs or errors, that the targeted or the translated text may have. It will be a good idea to record these types of bugs in a file and distribute it to the translators in the testing phase.

1) Untranslated English Strings in UI elements
2) Non-Fitting Translated Strings in UI elements
3) Inconsistent meaning of a term
4) Spelling Errors in Strings
5) Forceful Translation of English strings into Nepali which ought to be left in English.
6) Honorific Language Usage
7) Non-Contextual Translation

Reviewing the Translation
Peer reviews turn out to be best suited for reviewing translation. This approach is effective in that one can find bugs and errors in files completed by others, also it creates healthy competition among the translators. This process, in our case took almost 20% of the translation time. Corrective actions need to be taken after finding the errors in the translation. Two translators could do the correction in the files.

One of the most important factors that can affect the reviewing phase is the faulty strings in the file. Translation files need to be free of faulty strings as much as possible otherwise translated strings may not come in the User Interface element or only partial translation may be displayed. In an extreme case, the file will crash while compiling and may take time to correct the errors.

Generally faulty strings are translated strings with some arguments missing or have errors in syntax. Such errors can be corrected without much effort. However, there may be a situation where strings are shown as faulty even when they seem to be correct from every aspect. In such a case, some time should be dedicated to looking for possible errors in the file that caused the faulty strings.

6.5 References for Further Reading

a) The Primer: Localization of Free/Open Source Software. Anousak Souphavanh and Theppitak Karoonboonyanan.
b) http://www.non.gnu.org/cvs
c) Principles of translation (http://www.completetranslation.com/principles.htm)
d) A how-to on translating GNOME applications (http://developer.gnome.org/projects/gtp/l10n-guide/)
e) Specific rules about how-to write translation
f) (http://developer.gnome.org/documents/style-guide/locale-5.html)
g) http://docs.kde.org/development/en/kdesdk/kbabel/
h) http://l10n.kde.org/tools/
i) http://www.i18nguy.com/TranslationTools.html

7 Gnome Localization

7.1 Introduction

In this Chapter, we discuss different aspects of Gnome Localization involving the Gnome Desktop Environment and its components, the localization frameworks, the issues of Gnome versions, gettext tools, official gnome site for submission of the translated files etc. References to useful links for further reading are included at the end of the chapter.

7.2 What is X Window System and Window Managers?

X Window System

The X Window System which is commonly known as X11 or X is the standard windowing system in the Unix world. It was developed in the mid-1980s at MIT as Project Athena. The original purpose of the system was to allow users of the then emerging graphic terminals to access remote graphics workstations, irrespective of the workstation's operating system or the hardware. As the source code to write X is available, it has become the standard layer for the management of graphical and input/output devices and for developing both local and remote graphical interfaces on almost all Unix, Linux and Unix-like operating systems.

The interesting feature of X windowing system is that it allows a graphical terminal user to get access to the remote resources on the network as if they were accessible locally to the user. Running a single module of the software called the X server is all that is required for this purpose. The software running on the remote machine is called the client application. X's network transparency protocols allow the separation of the display and input portions of any application from the remainder of the applications and the service is available to any number of remote users.

For detailed information on the historical and relevant background information on X Windowing System, please refer to [7.4.b].

The X Client-Server Model and Network Transparency

X, with it's unique feature of allowing applications to run on a network server and rendering the output on to a desktop machine, was very significant in the 1980s and 1990s. In the first days of X, the widely used "X terminals" were dedicated X Window hardware. The terminal accepted input, rendered output but did not perform application processing.

X works on a client-server model. X server communicates with various client programs accepting requests for graphical output (windows) and sending back user input (from keyboard, mouse, or touchscreen).

The communication protocol between server and client operates network in a transparent manner. The client and the server may run on the same machine or on different ones, possibly with different architectures and operating systems, without any incompatibility problem. They can even communicate securely over the Internet through an encrypted network session.

For Example: In order to make a remote client display to a local server, the user will typically telnet or ssh to the remote machine via the terminal window, commanding it to display to the user's machine (e.g. export DISPLAY=[user's machine]:0 on a remote machine running bash), then start the client. The client will then connect to the local server and the remote application will display to the local screen and accept input from the local input devices. Alternatively, the local machine may run a small helper or program to connect to a remote machine and start the desired application.

Working with X Window System and Window Managers

X Window, by itself, only generates borderless windows in fixed screen locations. A "window manager" is required to add borders and buttons and enable the users to resize and move the windows on screen. The default X window manager is the Tabbed Window Manager (twm). However, more than three dozen others have been used, including AfterStep, Blackbox and Enlightenment. The KDE and GNOME user interfaces for Linux use Kwin and Metacity respectively as their window managers.

When the X Window System is run, the X server manages the display, based on requests received from the window manager. Hence, the window manager is in itself an X client, which has the responsibility for managing the appearance, behavior and placement of windows on the screen.

X itself does not have any role in determining the appearance of the screen, or what users are allowed to do with windows. That is the job of the window manager. For example, some window managers roll up the window into the title bar like rolling up a window shade when double-clicked in a window's title bar. This process is referred to as shading. Other window managers maximize the window to fill the desktop area as a response to the same action.

The X server's job is to provide the low-level support to the window manager. On the other hand, window managers are responsible for icon title bars and behavior of windows for applications, handling input and mouse gestures, clicks etc. apart from the responsibility for fixing window positions.

The window could be a terminal window (called an xterm) where the user is supposed to run standard Unix commands, or it could be an X client application like the xcalc calculator, a web browser, or an Emacs session or even more complex programs.

7.3 About GNOME

GNOME stands for GNU Network Object Model Environment. The desktop environment is one of the measures to put a user friendly working environment on top of UNIX and UNIX-like operating systems such as Linux. GNOME, which is developed as a Free Software offers a reasonably complete developer toolkit and application infrastructure and in addition to that an end-user working environment. The user interface elements are similar to the windowing environments that have been in use since the early eighties with the classic Window, Icon, Mouse, Pointer (WIMP) paradigm.

For detailed information on the history and relevant background information on GNOME, please refer to [7.4.d].

7.3.1 Stable Releases of GNOME

Each component included in the GNOME project, has its own version number and release schedule. In this regard, module maintainers co-ordinate their efforts to create a full GNOME stable release roughly once every six months. The latest stable version of GNOME as per March 2006 was 2.14. For a detailed list of stable releases of GNOME starting from version 1.0, please refer to [7.4.d].

7.3.2 GNOME Components

Given below is a general overview of the different components of GNOME.

a) Display Manager

The GNOME Display Manager(GDM) is a sophisticated **XDM** replacement.
It is an entirely new implementation of XDMCP (the X Display Manager Control Protocol) and associated functionality. It consists of four separate parts:

1. small daemon
2. graphical login program
3. host chooser
4. configurator

The GDM implements a rich feature set required for managing local and remote displays. With this manager, one can login to the remote computer and use the GNOME Desktop Window as if running in the local computer.

b) File System

GNOME interacts well with the the Linux File System. Among the popular file types and operations that it handles include VFS, Mime types, Metadata etc. Files are managed and stored in different formats and different file systems like ext2 ,reiserfs, fat32 etc. in GNOME.

c) Control Center

GNOME uses the Control Center to customize the user's desktop environment according to it's preferences. The user may set his/her preferences of the GNOME environment making use of some

configuration dialog boxes. For instance, the desktop's background color, mime-type handling and setting the mouse properties are all handled by the Control Center. You also may add your own configuration dialogs.

d) Panel

The Panel refers to a generic term used to describe a particular control interface between the user and the desktop environment. Typical example of a panel would be a bar at the bottom of the screen, with a menu from which users can launch applications (the GNOME Menu), a button bar with buttons representing launch targets, as well as running applications. A clock or an email notification program also may serve as panel applets.

Multiple panels also exist in the form of a full-size panel across the bottom of the screen and a smaller panel running down on the right side of the screen. In addition, panels can also be set to auto-hide, or can be manually slid in and out of view with a button on each end.

The panel works in communication with the Session Manager. Typical cases include notifying about the session managed applications to shut down as the users log out and shut down GNOME.

All changes need to be reacted by the panel dynamically. For example, if the GNOME Menu has been changed by the user taking the help of a menu editor, the panel needs to be notified in order to properly re-construct its GNOME Menu instantly without the need to completely restart the panel to get the changes in the GNOME Menu.

e) Desktop Icons and File Manager

A comfortable and an appealing interface is a very important part of a complete desktop environment. This is handled by Nautilus, which provides the desktop icons and file system windows for GNOME. Common operations such as Open/Delete/Copy/Move can be performed by using the appropriate mouse button. Similarly, changing the desktop icons and permission of files is also possible. A variety of Views (tree, icon and listview) as well as file searching and selection are some of the other features available in the GNOME file system windows. Drag and Drop is also fully supported.

f) Session Management

GNOME adopts session management to handle logging in and logging out. The states of programs are saved and restored providing a mechanism for the user to take "snapshots" of their desktop. In addition, the window behavior and customized menus are also maintained.

g) Window Managers

Interaction between the X server and its clients is redirected through the window manager. For instance, whenever an attempt to show a new window is made, the query or request made is redirected to the window manager for taking note of the initial position of the window. Apart from this, the Window Manager also handles input and mouse gestures, clicks etc. For GNOME, Metacity is the default window manager. Enlightment and Sawfish used to be the previous window managers for GNOME before Metacity was implemented.

7.3.3 Localization Framework in GNOME

English is the language usually used for programming and documentation. A common language like English is quite handy for communication between developers, maintainers and users around the globe. At the same time, a considerable portion of the population is less comfortable with English and would prefer to work in their respective native language. This has been made possible by the Translation Project. For this a translation framework is required for translating the software, manuals and documentations in English in the local language.

GNU gettext Translation Framework

The translation framework most commonly used in FOSS is GNU gettext [7.4.h]. GNU gettext is an important tool or framework for localization, as it is an asset on which we may build many other steps. It is an integrated set of tools and documentation for programmers, translators for making world ready software which can be localized and translated to local languages. Specifically, the GNU gettext utilities are a set of tools that provide a framework within which other free packages may produce multi-lingual messages. These tools include

- 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 message in various ways the sets of translatable strings, or already translated strings.

GNU GetText is designed to minimize the impact of internationalization on program sources,hence keeping this impact as small and hardly noticeable as possible. Internationalization has better chances of succeeding if it is minimal , or at least, appears to be so, when observing program sources.

The Translation Project also uses the GNU gettext distribution as a vehicle for documenting its structure and methods. This goes beyond the strict technicalities of documenting the GNU gettext properly. By doing so, translators will find in a single place, as far as possible, all they need to know for properly doing their translating work. Also, this supplemental documentation might also help programmers, and even curious users, in understanding how GNU gettext is related to the remainder of the Translation Project, and consequently, have a glimpse of the big picture

Technical aspects of GNU gettext
Messages in the program's source code are put in a short macro that call a gettext function to retrieve the translated version. At program initialization, the hashed message database corresponding to LC_MESSAGES locale category is loaded. Then, all messages covered by the macros are translated by quick glances at the program execution.

Therefore, the task of translation is to build the message translation database for a particular language and get it installed in appropriate place for the locale. With that preparation, the gettext programs are automatically translated as per locale setting without a touch in source code [7.4.h].

GNU gettext also provides tools for creating the message database. There are three kinds of files in the process:

- **POT (Portability Object Template) file.** This is a file in human readable form to be used as template for the PO file where only the english msgid string are available.

- **PO (Portability Object) file.** This is a file in human-readable form for the translators to work with. It is named so because of its plain-text nature which is portable to other platforms.

- **MO (Machine Object) file.** This is a hashed database for machine to read. It is the final format to be loaded by the gettext program. Note that there are other translation frameworks in commercial Unixes, and their MO files are not compatible. You may also find some GMO files as immediate output

gettext Tools
“gettext” is a package which contains tools for extracting strings from source files into translatable format, turning those strings into a list which can be translated, updated and later converted into a format, the computer can use. Someone from your localization team (as many as possible really) needs to be familiar with these tools.
The following commands or utility tools are available with gettext Framework:
msggrep
The `msggrep' program extracts all messages of a translation catalog that match a given pattern or belong to some given source files.
msgunfmt
Converts binary mo file to human readable po file format
msgcat
Builds message catalogue combining many message (PO) files. Merges PO files to retrive unique messages
msgconv
Converts character encoding of message in PO file like latin1 to utf-8
msginit

Initializes a message catalogue file. Creates a new PO file, initializing the meta information with values from the user's environment.

msguniq

Unifies duplicate translations in a translation catalog. Finds duplicate translations of the same message ID.

msgcmp

Compares two Uniforum style message catalogue PO files to check that both contain the same set of msgid strings.

msgfmt

Generates binary message catalog from textual translation description. Compiles message catalogue in binary format to use with application

msgmerge

Merges message catalogue PO file with the new and updated message template POT file.

7.3.4 Localizable Components of GNOME

The GNOME Desktop Environment comprises of developer libraries and set of applications. In order to have the GNOME Desktop Environment fully localized, translation of the menus, desktop files and mime-type files, alone is not sufficient. In addition, some application specific files as well as the core libraries of GNOME gtk+, glib, gconf, libgnome, libgnomegui, libbonobo, libgnomecanvas etc, all need to be translated and localized.

Among some of the main applications of GNOME are gdm (login window), gedit (text editor), evolution (mail client), metacity (window manager), nautilus (file explorer) and gnome – desktop (menus on the desktop). It also has various applications for audio/video, file viewers (including gpdf, gv etc), CD writing, desktop theme selection, printing etc. GNOME also includes office tools like gnumeric (spreadsheet), dia (diagram editor), planner etc. After the translation of a minimum number of files of gnome – desktop, menus, panel and some associated packages like nautilus and printing libraries, the GNOME desktop would start appearing in the local language.

7.3.5 GNOME versions and Localization

GNOME programs are designed in such a way that they may be localized as per the needs of a particular location. This applies for instance to printed pages in America appearing in a different size from printed pages in Europe; weather temperatures being displayed according to Fahrenheit and Celsius scales in different places; and the language seen in programs as per the user's native language.

Translation is a major part of the localization process. The translators take sentences in the original English, supply the appropriate translation, and add the file containing this information to the GNOME CVS repository so that the next release of the software contains the new language.

To begin with, you will need to find out what your language code is. This is a two - or three-letter code. More popular languages will tend to have two-letter codes, whereas more obscure languages will tend to get stuck with a three-letter one. For languages spoken in more than one country, a translation specific to a country will be followed by an underscore and the two-letter country code capitalized. This can further be appended with an at-sign and more qualifying information. For now, you need to know that language codes typically look like "ne" (Nepali) or "ne_NP" (Nepali used in Nepal) or "ne_IN" (Nepali used in India).

Language code is used to identify your localization, for example in the form of locales and files. Once you have your code, you can start translating. The language code will be defined in the GNOME Locale (Please refer to Chapter 2 for detailed information on Locale and Locale Development).

For the GNOME Desktop Environment to be displayed in a particular language, the locale selected has to match with the language being employed for the translation. For example, the locale ne_NP and the translation in the Nepali language in the case of Nepal.

Different versions of GNOME will have different number of strings to be translated. Generally the latest version will have more and older version lesser strings to translate. Similarly, the number of applications that have to be translated will also vary upon the versions. While undertaking the translation of any version of GNOME, one has to take note of the version number as translation work for one version may not necessarily work for another version.

7.3.6 Translation, Verification and Proof Reading

Basically, the file from GNOME that has to be translated is in the PO format(Please refer to PO format in Chapter 6 on Translation.) with *"msgid"* and *"msgstr".* The msgid will have the string in ENGLISH and msgstr will have the translation in the native language. The po file will be available for almost all the packages or application that is included in GNOME.

A major part of localizing GNOME and its applications is translating the PO files. Verification of the translated strings in terms of the meaning conveyed is vital once the translation process is complete. Besides, consistency in the translation must be checked, rechecked and verified so that a word e.g. "File" does not get translated into different forms in the *Native Language*.

Translators can use various editors for the translation work,. In our case, we chose KBabel (Please refer to the Chapter 14 under the section Translation Tools for a detailed information on KBabel).

Proof Reading is yet another vital part of translation. Spelling errors make the user interface unpleasant, unattractive and inefficient, so must be handled with great care. Preferably a linguist or someone with a sound linguistic knowledge would need to be involved in proof reading. Consulting individual words in strings from hundred of files is tedious and hectic. For this reason, freely available applications handling the Unicode input regular expressions might aid in finding and replacing one or more spelling errors in multiple files.

Once verified and proof-read, the translation must be tested in the real applications and checked if the formatted strings show up correctly in terms of the essential grammar structure. For instance, Nepali follows the SOV (Subject Object Verb) sentence pattern as opposed to ENGLISH which follows the SVO (Subject Verb Object) sentence pattern. For this purpose, it is advisable to arrange for opening two applications, one using the English user interface and the other with the user interface in the native language. A comparative look would determine the changes to be made.

7.3.7 Submitting Translated Files to GNOME Mainstream

Once the po file is translated, verified and tested in the application itself, the file has to be submitted in the GNOME CVS so that the new release of GNOME will have the translated strings incorporated.

Following are the steps required for submitting the translated files to GNOME.

1) The following two CVS commands download the two files “ChangeLog” and “ne.po” in the local machine from the GNOME central CVS.

   cvs checkout modulename/po/ChangeLog
   cvs checkout modulename/po/ne.po

2) Assuming that you have translated the ne.po file. Next you need to add comments in ChangeLog
   Name <email>

   *ne.po: Updated Nepali Translation
   Here ne.po referes to Updated Nepali Translation po File

   For example,
   2006-01-01 Pawan Chitrakar <pchitrakar@gmail.com>
   * ne.po: Updated Nepali Translation

   Please note the format of the comments
   1st line ------date (YYYY-mm-yy) two spaces Name two spaces <email>
   2nd line------Blank
   3rd line------{TAB} * space filename: message

3) Overwrite the old file in the central GNOME CVS with the new and updated ne.po file (The file which has been translated locally.)

4) Commit the changes
   cvs commit -m " Updated Nepali Translation"

7.3.8 GNOME Localization Status Page

The Gnome status for the translation can be viewed in the gnome localization status page. The url address is http://l10n-status.gnome.org. On this page, you can browse by language name (language code) .
This covers two major section:

a) Developer Library Section
b) Desktop Section

The status page provides information on the translated strings of the files.

For Example, in the Desktop section for Nepali Language, the translation status looks as in the figure shown below.

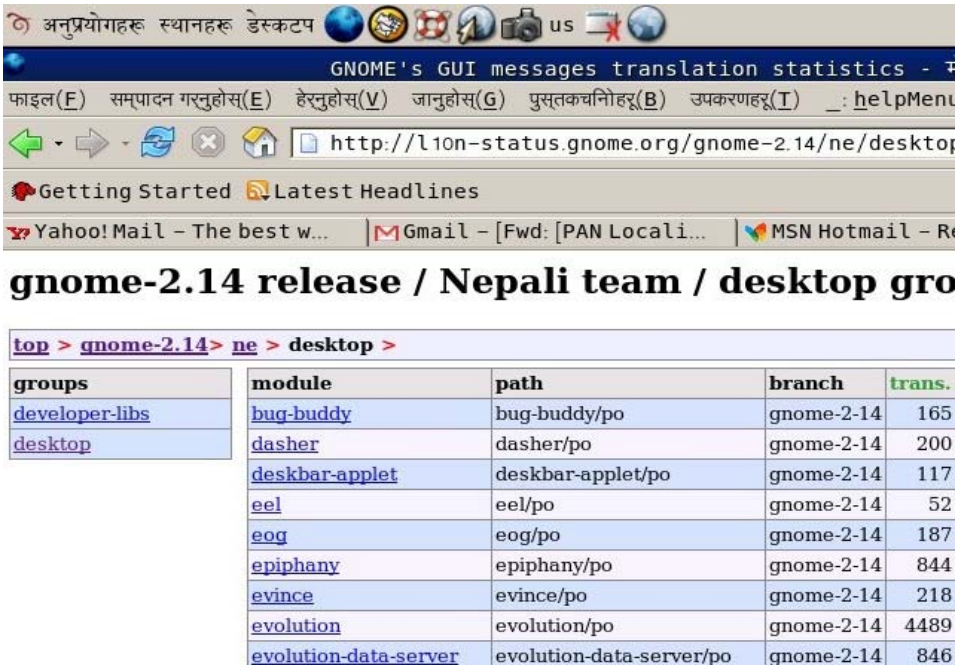

Figure 5. GNOME localization status page

7.3.9 Viewing GNOME Desktop in the Native Language

After translation is complete and locale is set to the native language, the GNOME Desktop interface will show up in native langauge This provides the user with the desktop and other applications in their native language. A screenshot of the localized GNOME Desktop in the Nepali Language is shown below.

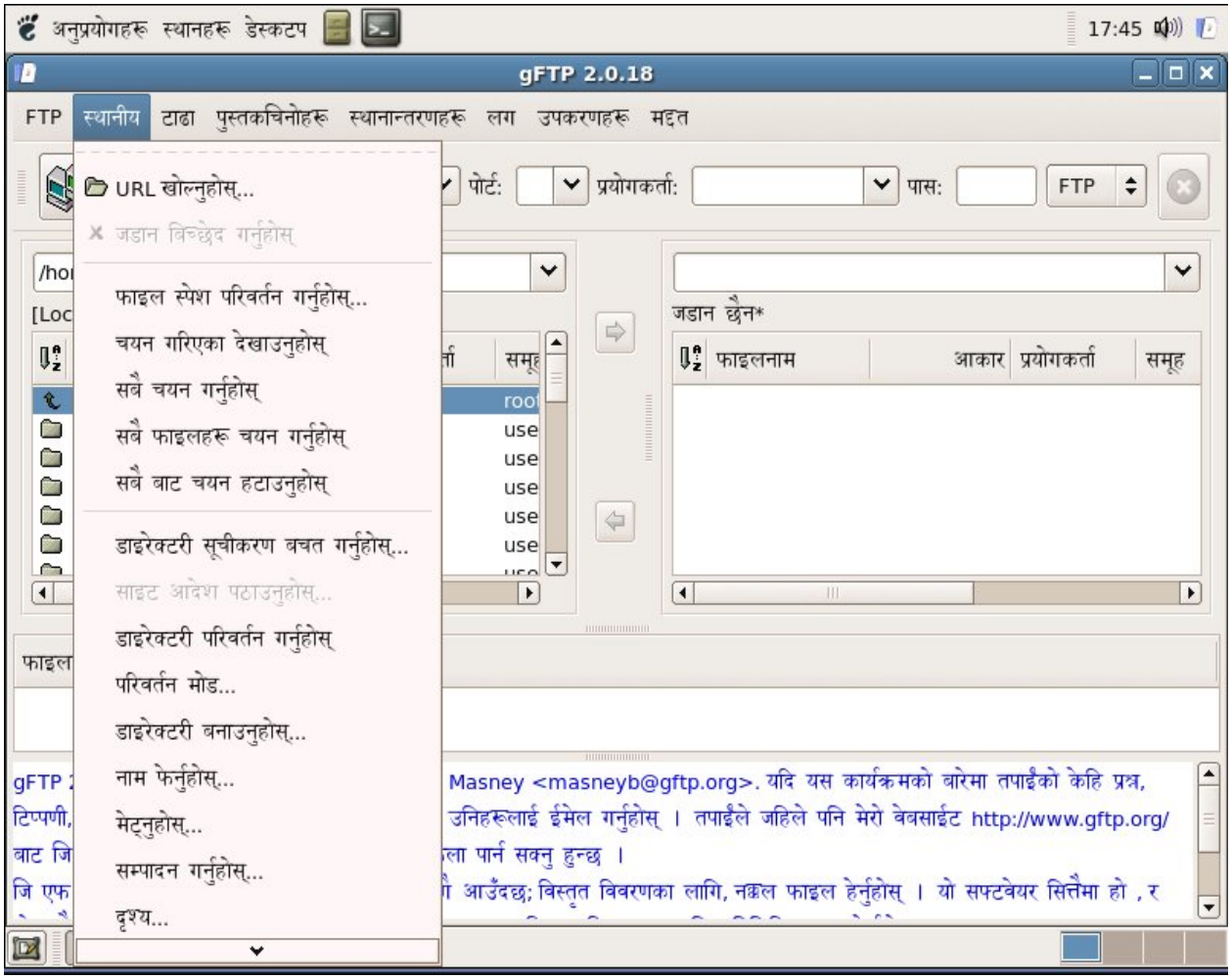

Figure 6. Localized Gnome desktop in Nepali

7.4 References for Further Reading

a) Linux in a Nutshell, fourth Edition. A Desktop quick reference. Ellen Siever, Stephen Figgins, Aaron Weber.
b) http://www.answers.com/topic/history-of-the-graphical-user-interface
c) http://en.wikipedia.org/wiki/X11
d) http://en.wikipedia.org/wiki/GNOME_desktop
e) http://www.gnome.org
f) http://www.fifi.org/cgi-bin/man2html/usr/share/man/man1/gnome.1.gz
g) http://l10n-status.gnome.org/
h) The Primer: Localization of Free/Open Source Software. Anousak Souphavanh and Theppitak Karoonboonyanan.

8 Mozilla Suite Localization

8.1 Introduction

In this Chapter, we begin with a short introduction about Mozilla. Next we will talk about Mozilla localization basics and the Mozilla Localization framework. A detailed discussion on the Mozilla Suite Localization Steps followed by an overview of the available translation tools. Similarly, the issue regarding complex text and Mozilla is also dealt with in this chapter. Other information on Mozilla in this chapater includes “Building Mozilla Suite from Source”, “Mozilla Plug-ins” and “Known Issues”., References for further reading are listed at the end of the chapter.

8.2 About Mozilla

Mozilla is a free/open source cross-platform Internet suite. Mozilla suite bundles Internet browser, Mail client, Composer, Address book and IRC chat client. Its development was initiated by Netscape Communications Corporation. Mozilla Suite works on multiple platforms like Linux/Unix, Microsoft Windows and Mac OS.

Most of the Mozilla Code has been written in C++. Other language used include Javascript, XUL, CSS, HTML etc.

Apart from being an Internet Suite, Mozilla is also a framework that allows developers to create cross-platform applications.

Evolution History

The information on the evolution history of Mozilla covered below in this section has been based on [9.4.c].

Mozilla as a project, was the continuation of the Netscape Communicator as an open project. In January 23, 1998, Netscape made an announcement that it would give away the source code for its Netscape Communicator suite. This announcement came as a huge surprise even to the Open Source community, never before had a major software company opened up its proprietary code. Netscape, however, couldn't release this code under GNU GPL license because the source code also included many third-party components and the owner of these components didn't want to give away their code. It was decided that a new license would be written to release this source code.

On 31 March 1998, Netscape finally released the source and named it Mozilla. They released it under a newly made Mozilla Public License(MPL). The Netscape Public License (NPL), under which the commercial version of Netscape, Netscape Communicator is released, is almost identical to MPL, except that the NPL includes amendments granting Netscape some additional rights.

All the source code for Mozilla is available under the Mozilla Public Licenses, which are accepted as free software licenses by the Free Software Foundation.

8.3 Mozilla Product Localization

8.3.1 Basic information

The Mozilla user interface is composed of several XML files. These pages are rendered by the layout engine at runtime [9.4.a]. Mozilla user interface is contained in resource files and are kept separate from the Mozilla core binary. Mozilla uses XUL language, pronounced “zool” (XML-based User-interface Language) for storing these UI layout. The XUL file produces the User Interface Layout, while strings to be displayed on it come from the separately placed .dtd and .properties file.

8.3.2 Localization framework

The following directory structure will help you to get a general idea of how different resource files work together in mozilla to give localized interface. It shows the format/structure generally followed when a new component (xfly) is added to mozilla. This is the minimum structure that has to be maintained in order to add a new component and be able to localize it.

An Example of Directory Structure of mozilla component is as shown below [9.4.c]:

```
MOZILLA/CHROME/
        XFLY/
        CONTENT/
                        XFLY.XUL
                        XFLY.JS
                        CONTENTS.RDF
       LOCALE/
                EN-US/
                XFLY.DTD
                CONTENTS.RDF
        NE-NP/
                        XFLY.DTD
                        CONTENTS.RDF
        SKIN/
                XFLY.CSS
                CONTENTS.RDF
```

XFLY.XUL: In the directory structure above, the xfly.xul file is the most important file. This file creates the actual application's widget Let us assume that xfly as a xml based hello world program which produces a small box and displays "Hello World" in it.
(Note: xul: pronounced zool is XML-based User-interface Language):

XFLY.JS : Javascript creates the functionality for a Mozilla-based application.

XFLY.CSS: Cascading Style Sheet formats the look and feel.

XFLY.DTD: The .dtd file holds the string which is displayed in the user interface. These strings are the ones which are to be localized while localizing. In the example above, there are two instance of xfly.dtd file. The xfly.dtd file inside the folder en-US holds the English string while the one inside ne-NP contains the translated string.

Now, when following command is executed,
#./mozilla -chrome chrome://xfly/contents -uilocale **en-US**

The xfly.xul file gets the strings(i.e english) from the xfly.dtd file inside the locale/en-US folder and displays the content on the widget and when executing,

#./mozilla -chrome chrome://xfly/contents -uilocale **ne-NP,**

the xfly.xul file gets the strings(i.e. Translated ones) from the xfly.dtd file from the locale/ne-NP folder and displays on the widget. Hence the interface is localized.

8.3.3 Mozilla Suite Localization Steps

Registration
Before considering thelocalization of any application, one should always check if somebody else has already started the localization work. If not, then you may start off by registering yourself in order to prevent duplication of the same work.

Check to see if somebody has already registered for localization of the intended product

For Mozilla Application Suite, you need to check,
http://www.mozilla.org/projects/l10n/mlp_status.html

For Firefox,
http://wiki.mozilla.org/L10n:Localization_Teams

For Thunderbird, http://www.mozilla.org/projects/l10n/mlp_otherproj.html#thunderbird

Registration and Localization Process for Mozilla Suite 1.7 for locale code ne-NP is shown below:

Registration Requirements

a) Your locale code: The locale code is usually assigned in ab-CD format. The first two characters(ab) represent the language and the second two characters(CD) represent the Country Name. For Nepali language spoken in Nepal, it would become ne-NP. For French language spoken in Canada it is fr-CA.

b) A valid email address: An email address is needed to subscribe to the mailing list.

Registration Steps

a) Go to page http://www.mozilla.org/projects/l10n/registration.html and subscribe to the mailing-list dev-l10n@lists.mozilla.org, or to its newsgroup mirror mozilla.dev.l10n.
b) File a bug in Bugzilla

From the above page, file a bug in bugzilla. Details should be as follows:

Product: "Mozilla Localizations", component : "Registration & Management".

Build Identifier: This identifies the exact version of product you are using. This is the line beginning "Mozilla/5.0" in Help | About.

Details : Provide the following details.

* the product you want to localize (e.g: Mozilla Suite)
* the language + country code (e.g. : ne-NP)
* your contact details (name + e-mail, web-page if available). (e.g: Basanta Shrestha, basanta@mpp.org.np, www.mpp.org.np)

Downloading Mozilla Build

The latest official build for Mozilla Suite can be downloaded from the following location:
http://www.mozilla.org/releases/

Alternatively, you may download the latest mozilla source and compile it yourself. But this is only recommended if you explicitly want some option to be enabled in the build. A very good example for this is a case where mozilla has to be built with CTL(Complex Text Layout) enabled, so that complex text like Devanagari can be rendered. Building mozilla from source will be discussed later in "Building Mozilla Suite from Source". Latest mozilla source can also be downloaded from the above location.

Working on Files

Chrome Files

In Mozilla, most UI resources are located in the chrome/ folder of the install directory. The original localization files are packaged as:

1. en-US.jar: contains nearly all the UI language resources.
2. en-win.jar: contains all the Windows specific UI resources.
3. en-unix.jar :contains all the Unix specific UI resources and
4. en-mac.jar: contains all the Mac OS specific UI resources.
5. US.jar: contains all the regional contents (URLs) resources.

Localizing mozilla technically means adapting the above mentioned 5 files for your desired locale, based on the above 5 files for English. That is to say, we have to extract and translate the translatable resources of en-US.jar and package it back with a new name (ab-CD.jar). For example, for Nepali language spoken in Nepal, one will have to prepare following files: ne-NP.jar, ne-win.jar, en-unix.jar, ne-mac.jar, NP.jar .

Listed below are the file types that we will come across while unpacking the above mentioned jar files. These are the files that actually hold the localizable strings and are needed to be translated into required language.

.dtd Files

Text files UTF-8 encoded
A DTD file contains a list of entities that need to be localized. The DTD files have most of the localizable

strings in Mozilla suite.
A part of a .dtd file is shown below:

```
----------------------------------------
<!ENTITY openCmd.label "Open Web Location...">
<!-- LOCALIZATION NOTE  throbber.url: DONT_TRANSLATE -->
<!ENTITY throbber.url "http://www.mozilla.org">
----------------------------------------
```

In the sample above, "openCmd.label" is the entity and its value, "Open Web Location...", is what needs to be localized/translated.

Where translation need not be done, the author of the file has added a comment “DONT_TRANSLATE”as shown in 2nd line of the above sample.

Note that these documents are UTF-8 encoded. Hence, you'll either need a text editor which is able to save the file in UTF-8 once modified, or you will have to convert it back to UTF-8 afterwards. Actual translations are usually done using appropriate tools like kBabel and not by directly editing the file in text editors.

.properties Files
These are text files with “escaped Unicode” encoded.
The .properties Files contain strings that are accessed by Javascript, C++, and possibly other scripting or component files.
A part of .properties file is show below.

```
----------------------------------------------
openButtonLabel=Open
chooseFileDialogTitle=Choose File
existingNavigatorWindow=Existing Navigator window
----------------------------------------------
```

The string on the right side of the equal symbol is to be translated. Again, localization notes and comments are often useful to explain what the string's usage in the UI is or a particular behavior the localizer should follow working on them:

Some Files contain symbols like %s or $1%s.

```
---------------------------------------
InstallFile=Installing: %s
ReplaceFile=Replacing: %s
---------------------------------------
```

They will be substituted at runtime by relevant words. Place them in your strings as they make sense in the whole sentence, typing them exactly as found originally.

.rdf (Resource Description Framework) Files
Text files, UTF-8 encoded, if not otherwise specified.

Each folder within chrome/ab-CD contains a copy of contents.rdf file. This acts as a manifest file which sits within a package and interacts with Mozilla's chrome directory. There is not much to change in this file, just change the chrome name en-US to ab-CD.

HTML and XHTML
UTF-8 encoded
Help contents are usually in .html or xhtml format.

One can directly open .dtd/.properties file and start translating using any text editor, capable of saving contents in UTF-8 format. But this is only good for testing and getting ideas on mozilla localization. For actual localization, you will need to use dedicated tools like Kbabel or Mozilla Translator.

Translation using MozPOTools
Translating using MozPOTools involves a very clever technique of converting mozilla style file format (.dtd

& .poperties) to GNU's gettext style po files for translation. Once the files have been converted to .po format, translators have the freedom to use any translation tools they prefer.

MozPOTools (collection of many commands including moz2po, po2moz, pomerge, pocount) are available in the package called translate-toolkit.

Using common tools like Kbabel is a bit more complicated than using the dedicated Mozilla Translator but the latter provides the following benefits over using Mozilla Translator

i) Work can be divided among translators
ii) Gives an overall idea of jar file XPI.

Profile Default
Profile Default includes user profile template files which are used when a new profile is created. It resides inside the defaults/profile/[country-code]/ folder. The localization is optional.

Copying/Registering the new packages
Copy all newly prepared set of .jar files into the chrome folder of mozilla installation. Now make mozilla chrome aware of this. This can be done manually by adding the following line in the file

mozilla/chrome/installed-chrome.txt
(Note: This is done automatically by install.js file when installing XPI.)

locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/global/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/necko/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/communicator/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/editor/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/mozldap/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/pipnss/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/pippki/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/navigator/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/cookie/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/wallet/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/content-packs/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/help/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/p3p/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/autoconfig/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/messenger/
locale,install,url,jar:resource:/chrome/ne-NP.jar!/locale/ne-NP/messenger-smime/
locale,install,url,jar:resource:/chrome/ne-unix.jar!/locale/ne-NP/global-platform/
locale,install,url,jar:resource:/chrome/ne-unix.jar!/locale/ne-NP/navigator-platform/
locale,install,url,jar:resource:/chrome/ne-unix.jar!/locale/ne-NP/communicator-platform/
locale,install,url,jar:resource:/chrome/NP.jar!/locale/NP/editor-region/
locale,install,url,jar:resource:/chrome/NP.jar!/locale/NP/navigator-region/
locale,install,url,jar:resource:/chrome/NP.jar!/locale/NP/communicator-region/
locale,install,url,jar:resource:/chrome/NP.jar!/locale/NP/global-region/
locale,install,url,jar:resource:/chrome/NP.jar!/locale/NP/messenger-region/
Now, run the following command and check the localized Interface.
#./mozilla -uilocale ne-NP

Packaging XPI
XPI Definition

XPI stands for cross platform installer and is pronounced as “zippy”[9.4.d]. XPI is a technology used by Mozilla products for installing extensions to add functionality. XPI is nothing but an archive of files. Along with other .jar files, a XPI typically contains a Javascript (install.js) file which extracts and copies files to specified locations.

In more recent XPIs, the install script has been replaced by a chrome manifest and a RDF file (install.rdf).
Creating XPI

Mozilla Translator can create language pack XPI itself. Those using MozPOTools, can archive all the .jar files along with install.js file using jar command to generate XPI. To get install.js file, download and extract "langenus" file, downloadable from
http://ftp.mozilla.org/pub/mozilla.org/mozilla/releases/mozilla1.7/windows-XPI/

(Note: replace mozilla1.7 with the appropriate version)

Maintain the following folder structure and then create archive to generate XPI.
/xpi/bin/chrome/ne-NP.jar
/xpi/bin/chrome/ne-win.jarh
/xpi/bin/chrome/ne-unix.jar
/xpi/bin/chrome/ne-mac.jar
/xpi/bin/chrome/NP.jar
/xpi/install.js

Now from the /xpi folder, run following command,
#jar cvf ne-NP.xpi .

Testing the XPI
After you have packed the jar files and made XPI, it is very important that you check if the XPI is installable. Sometimes during the process of translation and file format conversion, permission of files get altered and XPI eventually fails to get installed.

Often, the XPI does not get installed when the user doesn't have enough permission to write in the chrome folder -in such cases you will have to execute mozilla as root and install XPI.

To install XPI, just open the XPI file in the browser. If XPI is ok, it will ask to restart the browser.
In case of error in translation, mozilla prompts you with the file name that contains the error. There usually is .xul file with the same file name and you can open that particular file in the browser and check the error.

A .dtd file is used by a xul file with the same name. (E.g. abSelectAddressesDialog.xul uses the strings placed in abSelectAddressesDialog.dtd).

To open this file, type the following url in the address bar in the mozilla browser.
chrome://messenger/content/addressbook/abSelectAddressesDialog.xul

Submitting
Once the XPI has been made, mozilla.org can make your work publicly available from their ftp site http://ftp.mozilla.org/pub/mozilla.org/mozilla/l10n/lang

For this, put the XPI file somewhere from where mozilla people can download and send an email to the mailing list mlp-staff@mozilla.org about this. Your XPI will be uploaded to mozilla.org.

8.3.4 Translation Tools

Available Translation Tools
Some common tools used for mozilla localization are listed below [9.4.e]. Among these tools, Mozilla Translator and MozPOTools are discussed in detail.

- MozExpTool
- L10NZilla
- Mozilla Translator
- MozPOTools

MozExpTool
MozExpTool is a Visual C++ based project. It is designed to address the localization and leveraging the need of languages in single-byte charsets.

MozExpTool allows you to create text files (we call them glossaries) from dtd/xul files which can be given to

translators . When the translators have finished their translations, you can re-import them into the dtd/xul files.

L10NZilla
L10NZilla is a tool based on MySQL, Java and PHP. It allowsthe user to perform localization work using a browser. It offers the possibility of having a centralized maintenance of your translated resources.

Mozilla Translator
Mozilla Translator is a localization tool for Mozilla and other XUL based Application. It is based on Java. You can download the latest Mozilla Translator from http://sourceforge.net/projects/moztrans/. The latest version of Mozilla Translator during the time of writing this guide is 5.03.

Requirement
JAVA SDK
First, locate where the .jar files for English are. They will most probably be in the /usr/lib/mozilla/chrome/ folder. For making jar files of your language, mozilla translator uses these .jar files of english (en-US.jar, en-win.jar, en-unix.jar, en-mac.jar and US.jar) as templates. Take a note of where these files are located.

Run mozilla translator
java -jar mt502.jar

Goto File > Manage Products > Add
Fill in the fields as shown in this screen shot.

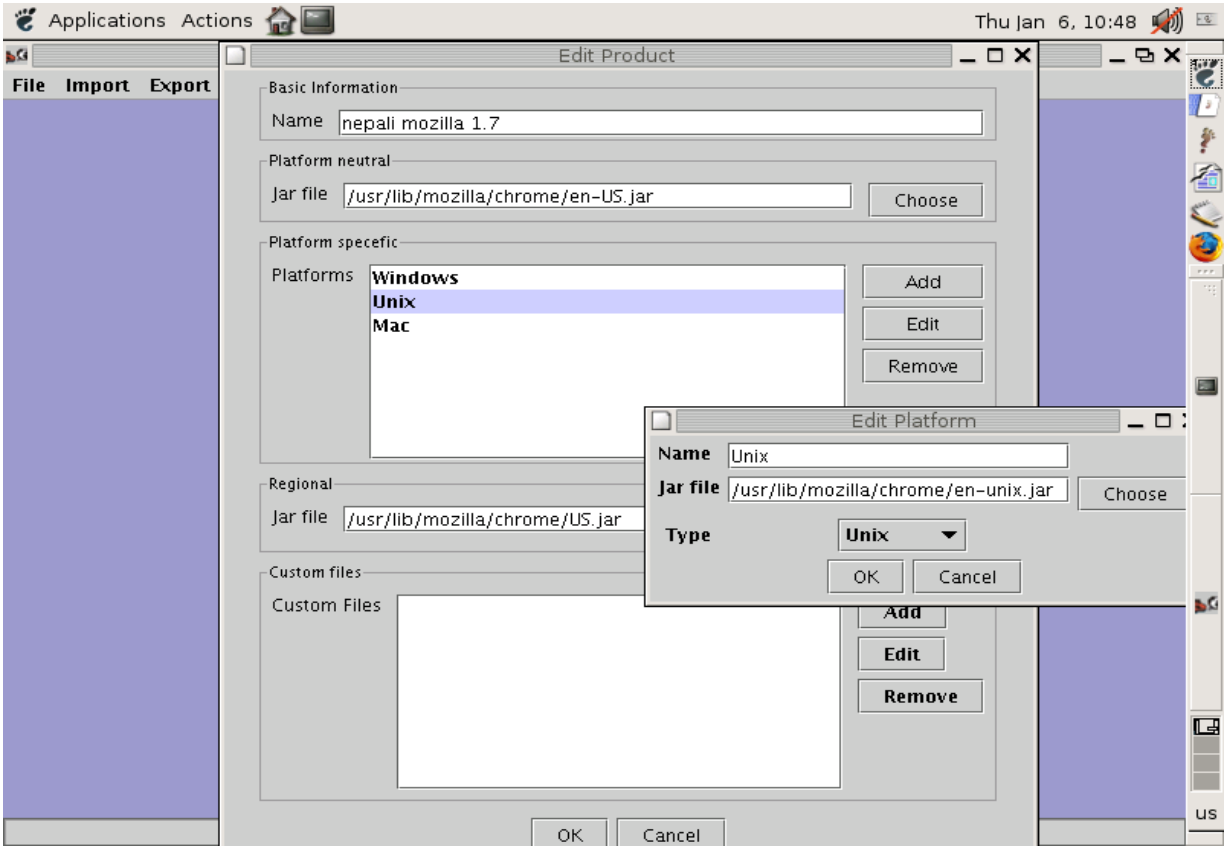

Figure 7. Screenshot of the Mozilla Translator

When all the files have been added, the final screen should look like this.

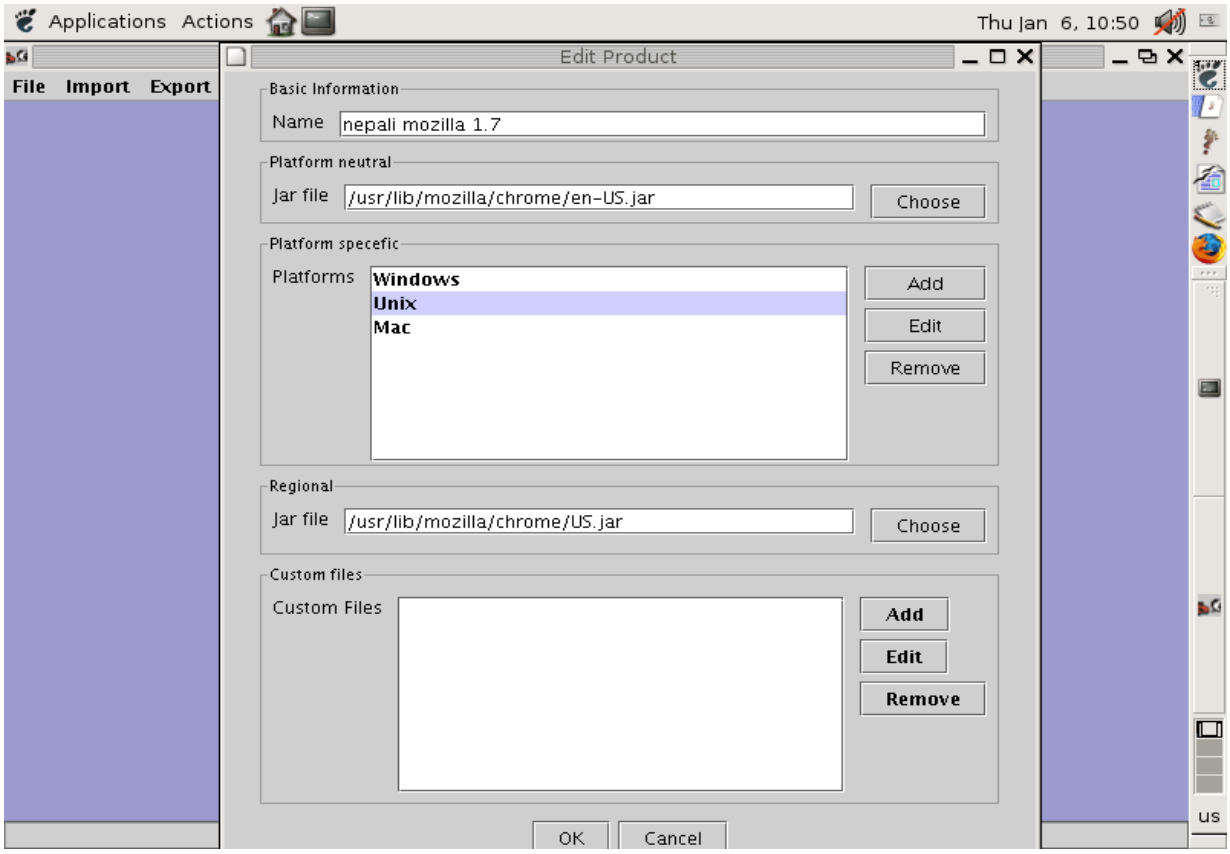

Figure 8. Screenshot of the Mozilla Translator

Go to File > Update Products .
Chrome View screen is displayed. Select the columns as shown in the following screen shot.

Click on ok.
You will get following screen.

Now you are ready to do the translation. There is an option in the Mozilla translator that will list all the redundant strings together. Hence you can simply translate one and copy paste the rest. For this goto Edit > Redundant Strings and click on “ok” on the next dialog box. You will get the following screen.

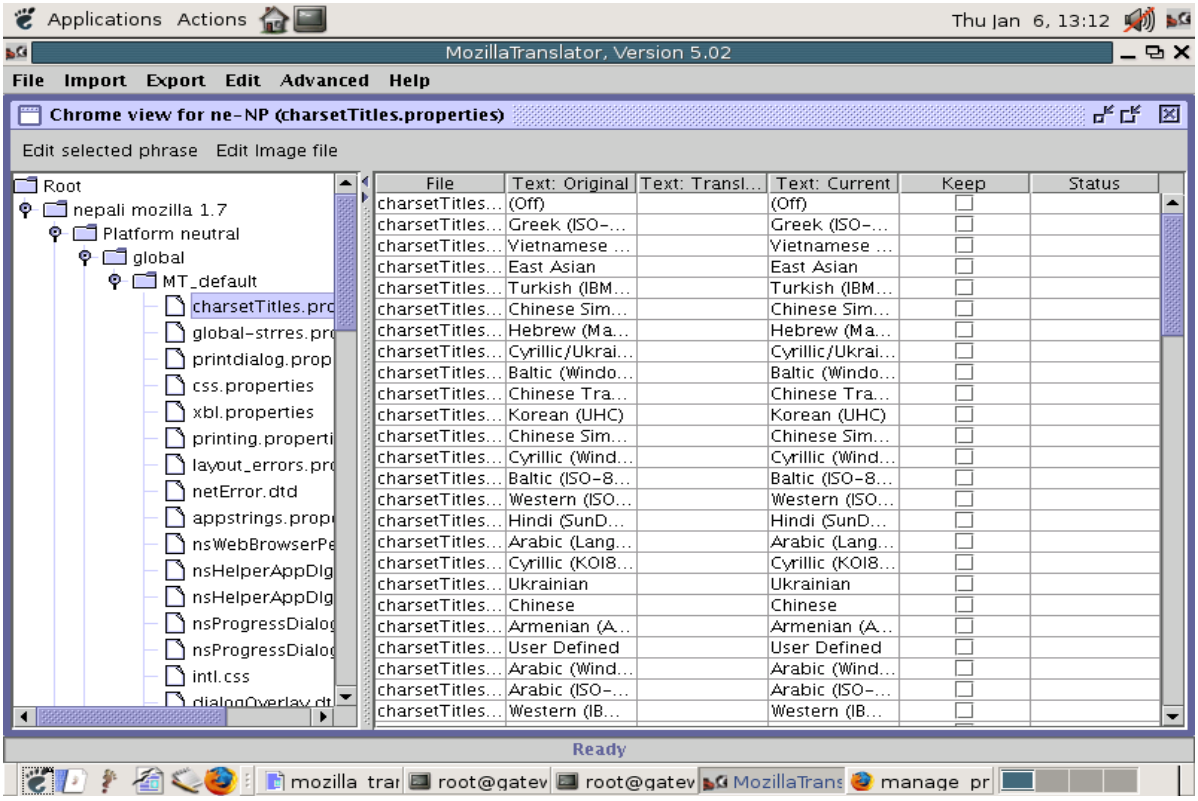

Figure 9. Screenshot of the Mozilla Translator

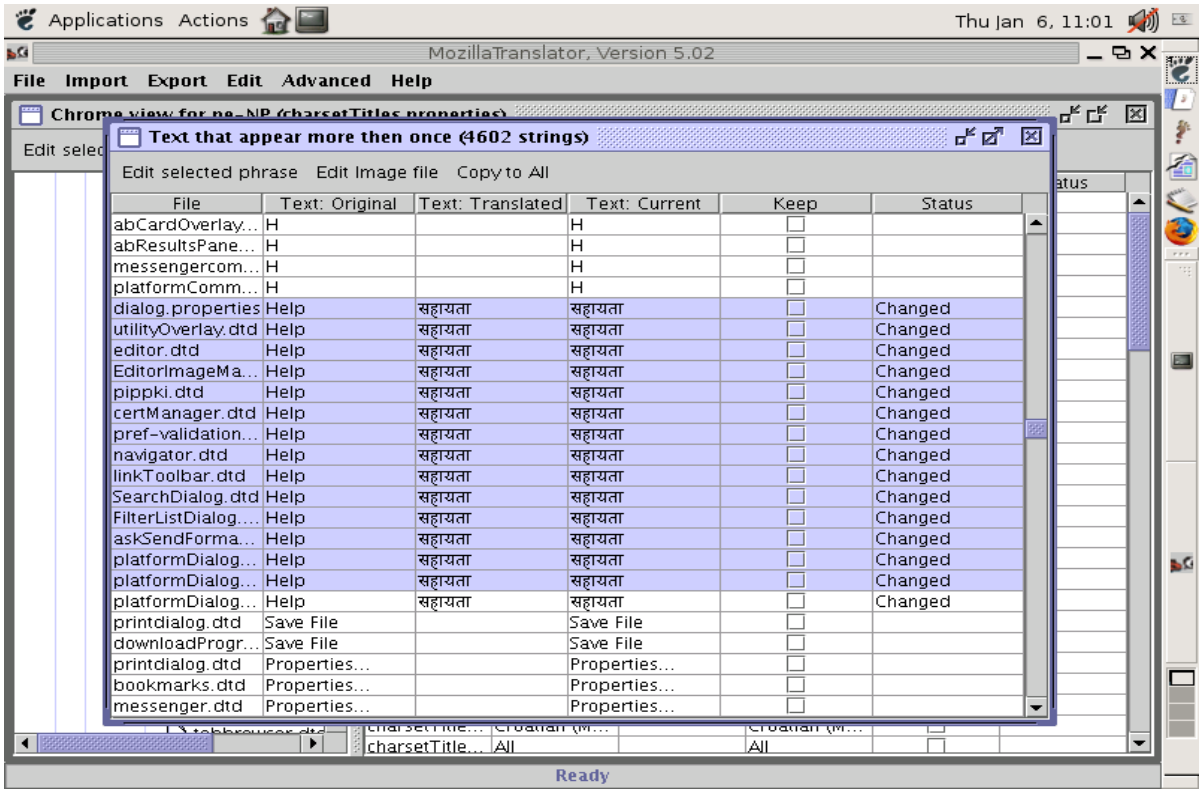

Figure 10. Screenshot of the Mozilla Translator

Now goto File > Update once again.
In following the steps above, if some times you don't get the screen as expected try “Update Product”.

Now we are ready to produce jar file of our own (eg. ne-NP.jar, ne-unix.jar etc.) . For this go to Export > Jar File. Fill the values as shown in the screen shot. Be careful while giving the version number. If the version is not correct, there will be problem while installing the XPI in the browser.

If you are making XPI for
1.7 -> put 1.7
1.7.x -> put 1.7
1.8 -> put 1.8
1.8a1->put 1.8

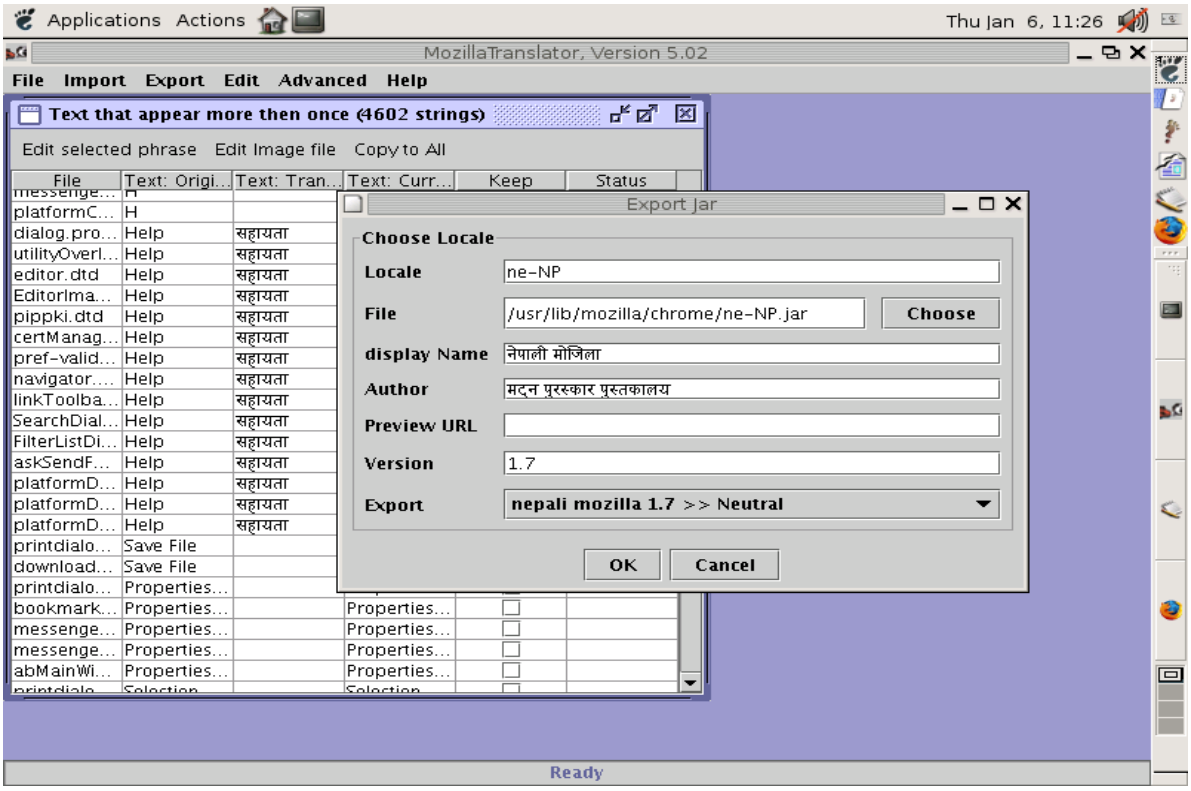

Figure 11. Screenshot of the Mozilla Translator

Check to see that the ne-NP.jar file has been created.

You have to repeat the above process for making rest of the files by appropriately selecting in the “Export” Field. ne-unix.jar is shown in the screen shot below.

Figure 12. Screenshot of the Mozilla Translator

Click on “ok”. Check to see that ne-unix.jar has been created. Repeat the above steps for ne-win.jar, ne-mac.jar and NP.jar.

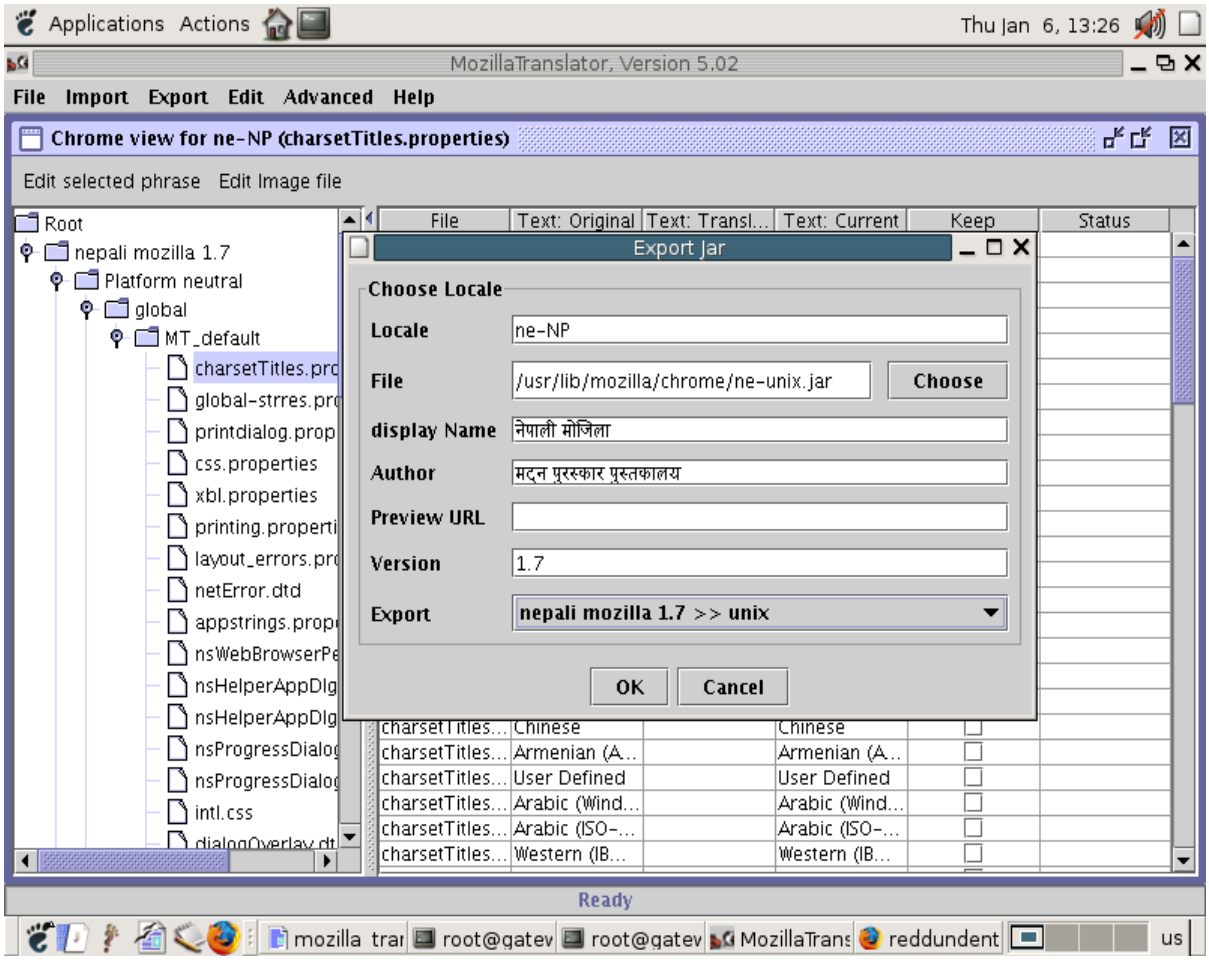

Figure 13. Screenshot of the Mozilla Translator

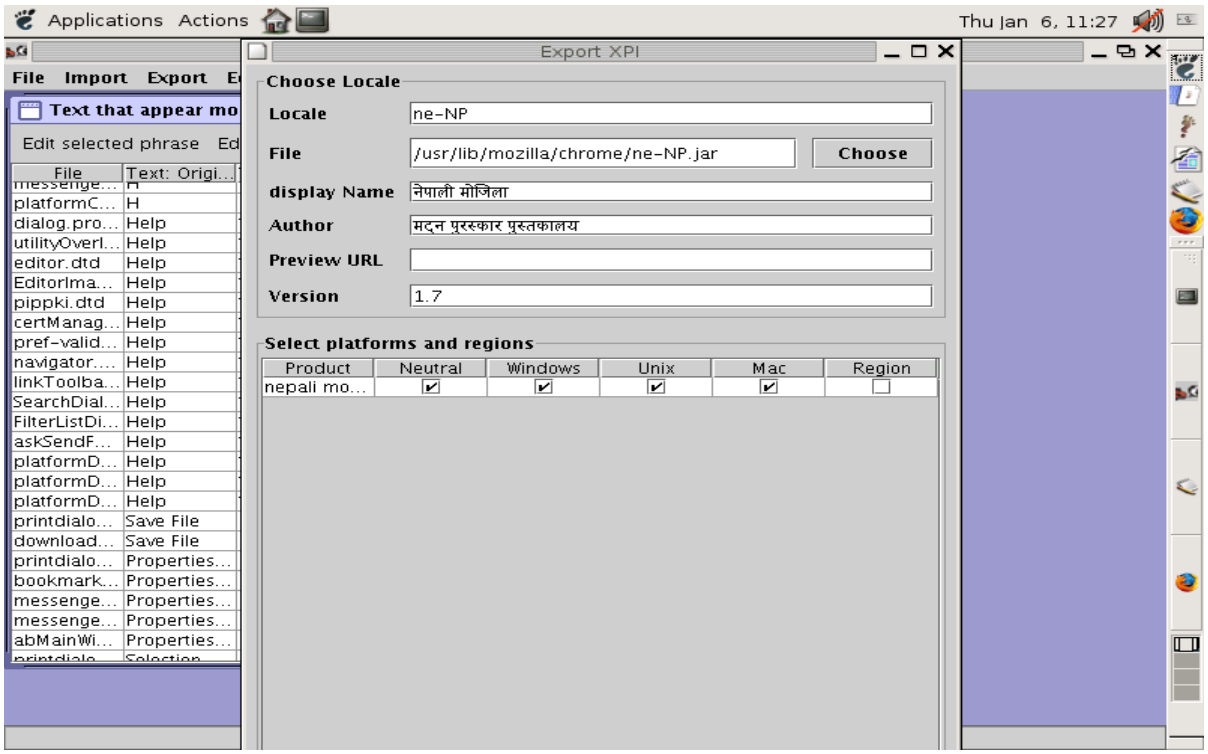

Figure 14. Screenshot of the Mozilla Translator

Now when all 5 jar files have been created, you have to make a XPI to be able to install through the browser. XPI file is the combination of jar file and a install.js (java script) file.
(Note:ne-NP.xpi =ne-NP.jar+ne-unix.jar+ ne-mac.jar+ne-win.jar+NP.jar + install.js).

Go to export > XPI install. Fill the fields as show in the following screen shot.

Click on “ok”. Now check and see if the ne-NP.xpi file has been created.

Now, Install the XPI into the browser.

Open the browser. Go to file > open file
choose the ne-NP.XPI
if the XPI gets installed successfully, you will get the following screen.

Now close the browser and open it again. The translated interfaces will appear as follows.

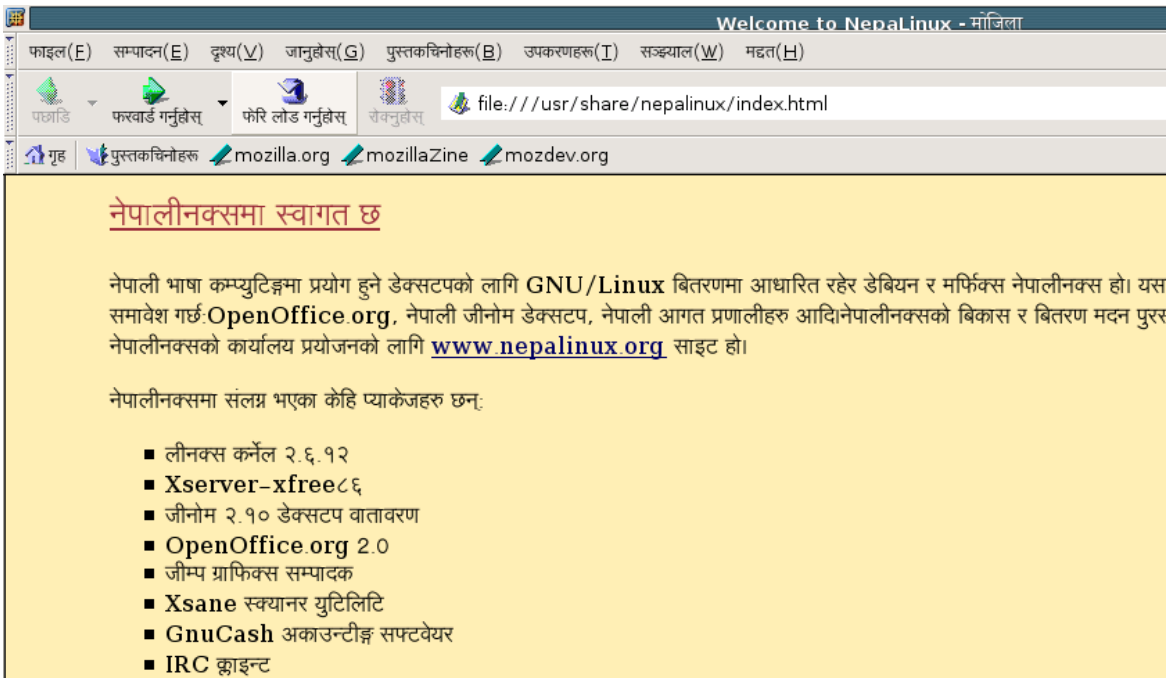

Figure 15. Localized Mozilla Nepali Browser

Creating Glossary File for further amendments
Translation is a tedious job and mistakes are almost unavoidable. Mozilla Translator provides you with a special feature of saving your work in a file called Glossary.zip and retrieving the work later when necessary so that modifications can be made easily.

For this, you need to save the Glossary in Mozilla Translator. Backup the following two files which are located in the same directory as your mt40x.jar file.

1. Glossary.zip
2. mozillatranslator.properties.

Later, to retrieve the translation in Mozilla Translator, copy the two above files in the folder where mt40x.jar exists and make a new folder "cache" inside that folder. Execute mozilla translator as usual.
java -jar mt502.jar
Now make the changes and make a new Glossary.zip for further changes.

MozPOTools
Localizable resource files (.dtd, & .properties) are converted into .po files and translated using common translation tools e.g. Kbabel. moz2po command is used for converting .dtd and .properties files to .po format and po2moz is used to convert it back to dtd & properties format. These commands are provided by package "translate-toolkit".

Installing Translate Toolkit
Download translate toolkit from
http://translate.sourceforge.net/releases/
#tar xzf translate-toolkit-0.8rc5.tar.gz
#cd translate-toolkit-0.8rc5
#su -c ./setup.py install
Apart from the two above mentioned commands, it also provides number of other useful commands like pocount, pomerge etc.

Gathering Localizable Resources

The following commands need to be followed for gathering the localizable resources.
#mkdir /localization
#cp /usr/lib/mozilla/chrome/en-US.jar /localization
#cp /usr/lib/mozilla/chrome/en-unix.jar /localization
#cp /usr/lib/mozilla/chrome/en-mac.jar /localization
#cp /usr/lib/mozilla/chrome/en-win.jar /localization
#cp /usr/lib/mozilla/chrome/US.jar /localization

Converting Files to PO format (moz2po)
The following commands need to be followed for converting Files to PO format.
cd /localization
unzip en-US.jar
mv locale locale_orig
moz2po -i locale_orig -o locale_po

Translation
All the localizable resource files .dtd and .properties are converted to .dtd.po and .properties.po respectively. Import the folder locale_po in Kbabel or other tools and start the translation.

Converting PO files back to Original format (po2moz)
The following command is used to convert PO files back to Original format.
po2moz -i locale_po -o locale -t locale_orig -l ne-NP
(Note: Here, locale_po is the folder where all the translation has been done, locale is the output folder which will be created, locale_orig is used as template folder and ne-NP is language code)

Packing the Work (Generating .jar)
Rename the folder name from locale/en-US to locale/ne-NP and make jar file
mv locale/en-US locale/ne-NP
jar cvf ne-NP.jar locale

(The above steps have to be repeated for all the en-unix.jar, en-mac.jar, en-win.jar and US.jar files. Before continuing with the other 4 jar files, you might want to take a backup of locale_po folder in case you need to make some changes in the translation.)

Building XPI
The following commands need to be run to build XPI.

mkdir -p /XPI/bin/chrome/
cp /localization/ne-NP.jar /XPI/bin/chrome/
cp /localization/ne-unix.jar /XPI/bin/chrome/
cp /localization/ne-mac.jar /XPI/bin/chrome/
cp /localization/ne-win.jar /XPI/bin/chrome/
cp /localization/NP.jar /XPI/bin/chrome/
cp /download/install.js /XPI
[Refer to “Creating XPI” in previous section to download appropriate version of install.js file]

cd /XPI
jar cvf ne-NP.XPI **.**

[Please note the trailing . in above command.]
The language pack XPI is ready now. To check, open the file in Mozilla and restart the browser.Examine the localized Interface

8.4 Complex Text and Mozilla

Complex Text Language
Complex Text Layout languages or CTL languages are languages whose writing systems require complex transformations of text. Devanagari, Thai, Hebrew are example of the languages which require complex text. For further information on Complex Text Languages refer to [9.4.l]

Gecko Layout Engine
Gecko is an open source web browser layout engine mostly used by Mozilla and Netscape browsers. For further information on Gecko Layout Engine, please refer to [9.4.m].
Gecko, NGLayout, XPFE, Rapter are the key words that we generally come across while talking about Layout Engines used by mozilla. All these words generally refer to the Layout Engine or Gecko. Gecko is the word generally used by Netscape.

Here are a few things that gecko provides [9.4.f] :

a) Layout engine and complementary browser components
b) Parsing for various document types (HTML, XML, CSS)
c) Advanced rendering capabilities to browser
d) Image rendering library
e) Networking library
f) Font library
g) Security library
h) Cache management system

Complex Text Rendering in Mozilla
Mozilla official builds are compiled without enabling CTL support. Complex text like Devanagari can only be rendered when mozilla is compiled with CTL support. Enabling CTL support in mozilla builds can be done in number of ways such as:
a) enabling pango shaper to render the text or
b) using ICU and mozilla XFT or
c) extending the CTL functionality to XFT back end itself

Here in this guide, a patch made by Mr. Jungshik Shin has been used to extend CTL functionality to XFT back end.

8.5 Building Mozilla Suite from Source

Build Requirements [9.4.f]
1) A recent POSIX Shell;
2) gcc 3.2 or higher;
3) g++ 3.2 or higher;
4) Perl 5.6 or higher;
5) CVS 1.11 or higher (needed for CVS build only);
6) gtk2 or gtk (1.2.0);
7) Gtk-devel;
8) libXt-devel for X11/Intrinsic.h, X11/Shell.h;
9) libIDL 0.6.3 or higher;
10) libIDL-devel 0.6.3 or higher;
11) libORBit;
12) libORBit-devel;
13) zip 2.3 (or higher);
14) freetype 2.1.0 (or higher);
15) fontconfig;
16) pkgconfig 0.9.0 (or higher).

Mozilla Source Download
The Mozilla Source may be downloaded from the following link.
http://ftp.mozilla.org/pub/mozilla.org/mozilla/releases/mozilla1.7/src

Patch Download
Download the patch from the following location. This patch from Jungshik Shin re-uses the shapers to extend CTL functionality to XFT backed.
http://bugzilla.mozilla.org/attachment.cgi?id=152102

Note that later versions of mozilla might not need to be patched.

Applying the Patch
The following commands have to be run to apply the patch.

tar jxvf mozilla-source-1.7.tar.bz2
cd mozilla
rename the patch to ctl.patch and copy it into the mozilla folder.
mv attachment.cgi?id=152102 ctl.patch
#patch -Np0 -i ctl.patch

The following 2 files are patched.

a) mozilla/configure.in
b) mozilla/gfx/src/gtk/nsFontMatricsXft.cpp

Once the patching is successful you are ready for compilation.

Generating build environment
export MOZILLA_OFFICIAL=1
export BUILD_OFFICIAL=1
export MOZ_INTERNAL_LIBART_LGPL= 1

Preparation for compilation
Prepare for compilation with following options
./configure --enable-mathml --enable-crypto --enable-xft --enable-default-toolkit=gtk2 --disable-ctl --enable-optimize=-O2 --disable-debug --disable-tests --without-system-nspr --without-system-zlib --without-system-jpeg --without-system-png --without-system-mng --disable-xprint --disable-freetype --enable-pango

Compilation
Compilation involves the following:
#make
After build is successful, execute and check the rendering by visiting site.
#mozilla/dist/bin/./mozilla

If the rendering is satisfactory, go ahead with the preparation of the mozilla-installer as mentioned in the following section.

Generating the Mozilla Installer
The following commands have to be run for generating the Mozilla Installer.
cd mozilla/XPInstall/packager/unix
perl deliver.pl
you will get erroran complaining about not finding the mozilla-installer.bin file.
To eliminate this error, download a mozilla-installer from mozilla.com, unzip it and copy this particular file "mozilla-installer.bin" to the following folder
mozilla/XPInstall/wizard/unix/src2/
Run the perl script again. The mozilla-installer.tar.gz will be available in the following location:
mozilla/installer/sea/

Generating mozilla.tar.gz
The following commands need to be run in order to generate mozilla.tar.gz.
cd mozilla/XPInstall/packager
make

The mozilla tar.gz “mozilla-i686-pc-linux-gnu.tar.gz” is created in the “mozilla/dist” folder.

8.6 Mozilla Plug-ins

Adding Flash Plug-in
Download "install_flash_player_7_linux.tar.gz" from macromedia.com
Unpack it. “install_flash_player_7_linux” is created.
Navigate to this directory and from the command line type ./flashplayer-installer. Follow the process and provide the installation path for Mozilla.

To verify that the plug-in has been installed, restart Mozilla and choose Help > About Plug-ins from the browser menu.

Adding Java VM Plug-in
Install JRE or JDK
Make a link as follows
#ln -s /usr/lib/PATH TO JAVA FOLDER/jre/plugin/i386/ns610-gcc32/libjavaplugin_oji.so /usr/local/mozilla/plug-ins/

To verify, restart Mozilla and choose Help > About Plug-ins from the browser menu. You can also verify by opening up the following site “http://java.sun.com/applets/other/TumblingDuke/index.html”.

8.7 Some known issues to be addressed

Display problems
Mozilla built using the above CTL patch solved the rendering problem, but still there are a few small problems like:

a) inconsistent spacing between the characters
b) cursor movement while doing text selection

Printing problems
While printing Devanagari characters from mozilla, the text is not rendered as it is seen on the screen. This problem still remains to be solved in mozilla and firefox during the time of writing this guide.

8.8 References for Further Reading

Since we will be discussing Mozilla FireFox Localization in the next Chapter and, as Mozilla Suite and Mozilla FireFox are almost similar, references for further reading for Chapters 8 and 9 is provided collectively at the end of Chapter 9.

9 Mozilla FireFox Localization

9.1 Introduction

In this Chapter, we will discuss Mozilla FireFox Localization. We have decided to dedicate a separate chapter on Mozilla FireFox Localization, as the previous Chapter on Mozilla Suite Localization does not include just a browser but much more than that, which covers the Mozilla Web Composer, Mozilla mail client, Mozilla browser etc. At the end of this Chapter, references for further reading are provided applicable both for Chapter 8 and Chapter 9.

About FireFox

Firefox is a free, open-source web browser. Firefox is becoming increasingly popular because of its availability for all popular platforms like Linux, Mac OS X and MS windows. Besides being cross platform, it is small, fast and easy to use.

The main advantages offered by Firefox over conventional browsers are:

a) Tabbed browsing
b) Pop-up blocker
c) Live bookmarks
d) Extension mechanism for adding functionality

Evolution History

Firefox was developed by the Mozilla Corporation. Because of the popularity of firefox, firefox development has become the main focus of the Mozilla Foundation. Firefox is developed as a division of browser-only from Mozilla Application Suite. But, please note that Firefox cannot be considered as standalone mozilla browser. The user interface in Firefox differs from mozilla browser in many ways. For example, customizable toolbars are the features that mozilla browser does not have [9.4.h].

Firefox has become one of the most downloaded free and open source application. Please refer to http://en.wikipedia.org/wiki/Mozilla_Firefox for further information.

Differences between FireFox and Mozilla

Mozilla Application Suite (also known as SeaMonkey) and Mozilla Firefox are both products of the Mozilla Corporation. The prime difference between the two is that Firefox is just a browser whereas Mozilla Suite is the collection of the following 5 components;

1) Navigator -> Browser
2) Mail & Newsgroups ->Mail Client
3) Composer ->Web Page Designer
4) Address Book -> Address Manager
5) IRC Chat -> Internet Relay Chat Client.

9.2 FireFox Localization

Firefox Localization is similar to Mozilla suite localization as firefox also use the same localizable resource file format as Mozilla Suite - that is .dtd and .properties. So localization of Firefox can be done using mozilla Translator as well as MozPOTools.

Building a Localized Firefox Build using CVS

By default, Firefox source tree pulls and builds only the English(en-US) localized files. To build Firefox in another language, you must pull locales of your language from CVS and pass the special option while compiling. If your locale is not available in CVS, then you can manually create your locale directory structure matching the locale directory structure of the other language.

Pulling FireFox Source Code

Firefox 1.5 (code name Deer Park) and locale code “nl” has been taken as example. Always use password. “anonymous”.

```
#mkdir /deerpark/
#cd /deerpark
#cvs -d :pserver:anonymous@cvs-mirror.mozilla.org:/cvsroot login
#cvs -d :pserver:anonymous@cvs-mirror.mozilla.org:/cvsroot co -r
FIREFOX_1_5_RELEASE mozilla/client.mk
#cd mozilla
#make -f client.mk checkout MOZ_CO_PROJECT=browser
```

(Note : Check http://developer.mozilla.org/en/docs/CVS_Tags to find out which branch should be selected.)

Pulling Locale
The following operations in central CVS server of Mozilla have to be performed to pull the locale:

```
#cvs logout
#cd ..
#cvs -d :pserver:anonymous@cvs-mirror.mozilla.org:/l10n login
#cvs -d :pserver:anonymous@cvs-mirror.mozilla.org:/l10n co -r FIREFOX_1_5_RELEASE l10n/nl
```

Creating a new locale
If no one else has initiated the localization for your language, it is likely that there is no l10n folder for your language in the central CVS server. In that case, you will have to make your l10n structure manually. Make folders in different levels so that it is an exact matche to the one you downloaded (here l1on/nl).

If your locale is available in CVS then you can skip this section.
After the folders have been created, copy the contents into the newly created folder as follows. Copy everything under /deerpark/mozilla/browser/locales/en-US/ into /deerpark/l10n/ab-CD/browser/.

Populate other folders in the same way respectively.

mozilla/browser/locales/en-US/

```
/deerpark/mozilla/dom/locales/en-US/
/deerpark/mozilla/editor/ui/locales/en-US/
/deerpark/mozilla/extensions/reporter/locales/en-US/
/deerpark/mozilla/mail/locales/en-US/
/deerpark/mozilla/netwerk/locales/en-US/
/deerpark/mozilla/other-licenses/branding/firefox/locales/en-US
/deerpark/mozilla/security/manager/locales/en-US/
/deerpark/mozilla/toolkit/locales/en-US/
```

l10n/ab-CD/browser/

```
/deerpark/l10n/ab-CD/dom/
/deerpark/l10n/ab-CD/editor/ui/
/deerpark/l10n/ab-CD/extensions/reporter/
/deerpark/l10n/ab-CD/mail/
/deerpark/l10n/ab-CD/netwerk/
/deerpark/l10n/ab-CD/other-licenses/branding/firefox/
/deerpark/l10n/ab-CD/security/manager/
/deerpark/l10n/ab-CD/toolkit/
```

Now translate everything under /deerpark/mozilla/l10n/ab-CD/. Almost all localizable resource files are located inside the chrome folder. Translation can be done folder by folder. The process using MozPOTools is shown below.

```
#cd /deerpark/l10n/
#mv ab-CD ab-CD_orig
#moz2po -i ab-CD_orig -o ab-CD_po
```

In the command above, ab-CD_orig is the original folder under l10n folder and ab-CD_po is the new folder that will be created.

After the command has been successfully executed, check to see if .dtd and .properties files have been converted to .dtd.po and .properties.po respectively. Now, open the ab-CD_po folder in translation tools like kbabel and start translating. If you just want to see the translation process, open any po file in editors like gedit and translate a few strings.
After the translation is done, convert the po files back to mozilla native format as follows:

#po2moz -i ab-CD_po -o ab-CD -t ab-CD_orig -l ab-CD
Here ab-CD is the new folder to be created. ab-CD_orig folder works as A template folder, -l ab-CD indicates locale code.

Preparation for building
Create a file named mozconfig-firefox as follows and copy it into the mozilla folder
---
. $topsrcdir/browser/config/mozconfig
mk_add_options MOZ_OBJDIR=@topsrcdir@/ffbuild
ac_add_options --prefix=/usr
ac_add_options '--mandir=${prefix}/share/man'
ac_add_options '--infodir=${prefix}/share/info'
ac_add_options --enable-default-toolkit=gtk2
ac_add_options --with-default-mozilla-five-home=/usr/lib/firefox
ac_add_options --enable-pango
ac_add_options --with-user-appdir=.mozilla
ac_add_options --with-system-png=/usr
ac_add_options --with-system-jpeg=/usr
ac_add_options --disable-mailnews
ac_add_options --disable-composer
ac_add_options --disable-ldap
ac_add_options --enable-postscript
ac_add_options --disable-installer
ac_add_options --enable-xprint
ac_add_options --enable-crypto
ac_add_options --enable-strip-libs
ac_add_options --enable-canvas
ac_add_options --enable-svg
ac_add_options --enable-svg-renderer=cairo
ac_add_options --enable-system-cairo
ac_add_options --enable-mathml
ac_add_options --disable-tests
ac_add_options --disable-gtktest
ac_add_options --disable-debug
ac_add_options --enable-xft
ac_add_options '--enable-optimize=-pipe\\ -w\\ -O2'
ac_add_options --with-system-zlib=/usr
ac_add_options --without-system-nspr
ac_add_options --enable-xinerama
ac_add_options --enable-extensions=default
ac_add_options --disable-pedantic
ac_add_options --disable-long-long-warning
ac_add_options --enable-single-profile
ac_add_options --disable-profilesharing
ac_add_options --enable-gnomevfs
ac_add_options --disable-installer
ac_add_options --disable-updater
ac_add_options --enable-chrome-format=flat
ac_add_options –enable-ui-locale=nl

Option, **. $topsrcdir/browser/config/mozconfig** sources the parameters given in the file /deerpark/mozilla/browser/config/mozconfig.

mk_add_options MOZ_OBJDIR=@topsrcdir@/ffbuild, enables you to build firefox with an objdir. This option facilitates in building firefox in a directory (/deerpark/mozilla/ffbuild) other than the source directory itself, thus, preventing the source folder from contamination.

ac_add_options –enable-pango forces firefox to use pango engine for advanced font rendering

ac_add_options –enable-ui-locale=nl builds localized firefox build.

To pass this option to build a localized Firefox, you must have your l10n CVS folder next to the mozilla folder. E.g. /deerpark/mozilla/ and /deerpark/l10n/nl

Now we are ready to build a localized firefox build. Set environment as follows:
#export MOZCONFIG=/deerpark/mozilla/mozconfig-firefox
[Note: Always provide full path here]

Building
Run the following command for building.
#make -f client.mk build_all

Executing/Checking build option
When the build is successful, execute firefox from objdir. In our case,
/deerpark/mozilla/ffbuild/dist/bin/firefox.

Check the exact version of firefox from Help menu. Type about:buildconfig in the address bar to examine compilation option. And also check if thecomplex text has been rendered correctly by visiting some bilingual sites.

Generating a tarball of localized firefox build
Now, in order to obtain a tarball of the build run the following commands.
#cd /deerpark/mozilla/ffbuild/XPInstall/packager/
#make

The tarball firefox-1.5.nl.linux-i686.tar.gz will be created under /deerpark/mozilla/ffbuild/dist folder.

Generating an installable language pack (XPI)
It is not always necessary to compile firefox to localize it. One can also create language pack xpi only and localize firefox. Po2moz is the proper way to generate language pack xpi but our attempt to do so was unsuccessful. Instead xpis for locales may be used, which are available in mozilla or created automatically and put in mozilla.org.

A manual process of generating xpi locally is shown here. This will give an insight into xpi structure and its working.

In the example below, a new xpi, ne-NP.xpi will be generated with the help of already available xpis, en-US.xpi and nl.xpi.

en-US.xpi will be used as template for extracting localizable resource while nl.xpi will to used to create xpi directory structure.

Follow the exact steps below. Lines with # are the actual command to be issued.
#mkdir /enxpi
#mkdir /nlxpi
#cd /enxpi
#download en-US.xpi
from http://ftp.mozilla.org/pub/mozilla.org/firefox/releases/1.5/linux-i686/xpi/en-US.xpi
#cd /nlxpi
download nl.xpi
from http://ftp.mozilla.org/pub/mozilla.org/firefox/releases/1.5/linux-i686/xpi/nl.xpi
#cd /enxpi
#unzip en-US.xpi

#cd bin/chrome
#unzip en-US.jar
#locale folder will be created
#mv locale/en-US/ locale/ne-NP
now translate everything under the folder locale.
#jar cvf ne-NP.jar locale
#cd /nlxpi
#unzip nl.xpi
#rm nl.xpi
#cd chrome
#rm nl.jar
#cp /enxpi/bin/chrome/ne-NP.jar .
#cd ..
Now, open the file chrome.manifest and replace all entries of nl and nl.jar with ne-NP and ne-NP.jar respectively.Open the file install.rdf. Change em:id, em:name and em:contributer entry to match your specification. DO NOT edit anything under em:targetApplication.Now generate ne-NP.xpi as follows.

jar cvf ne-NP.xpi .

Installing XPI
Execute firefox and open this newly created ne-NP.xpi file and install it Restart firefox to see the localized interface. You might have to execute Firefox with "./firefox -uilocale ne-NP" sometimes.

9.3 Complex Text and Mozilla FireFox

Mozilla and mozilla firefox use their own rendering engine for font rendering. Firefox built with CTL support is able to render indic scripts. Firefox binaries supplied by fedora core 4 and Ubuntu Linux are compiled with CTL. So executing firefox with environment MOZ_ENABLE_PANGO=1 in these distros will render indic script. Firefox official builds will have CTL integrated by default only around the beginning of 2007 [9.4.j].

Pango is not turned on by default in firefox official builds because Latin texts are rendered better by mozilla's default rendering engine rather than pango. Besides, enabling pango also increases the application's executing time.

Complex Text Rendering in Firefox is much more satisfactory than in Mozilla Suite.

9.4 Tools Available for the Localization of Mozilla FireFox

Localization of Mozilla based products is a specific task and cannot be managed with common localization tools and scripts. There are a number of tools/scripts available for the Localization of Mozilla based products.

Mozilla Translator
Mozilla Translator is the easiest tool for localization of mozilla suite and thesame may be used for localization of Mozilla Firefox.But due to huge changes in the locale structure in firefox, Mozilla Translator is becoming increasingly less appropriate because of the extra work that has to be done manually.

Build Requirements [9.4.f]

a) A recent POSIX Shell
b) gcc 3.2 or hgher
c) g++ 3.2 or higher
d) Perl 5.6 or higher
e) CVS 1.11 or higher (needed for CVS build only)
f) gtk2 or gtk (1.2.0) QT and plain xlib configure options are available, but they are not well tested or supported
g) libXt-devel for X11/Intrinsic.h, X11/Shell.h
h) libIDL 0.6.3 or higher
i) libIDL-devel 0.6.3 or higher
j) libORBit
k) libORBit-devel
l) zip 2.3 (or higher)

m) freetype 2.1.0 (or higher)
n) fontconfig
o) pkgconfig 0.9.0 (or higher)

MozPOTools
MozPOTools is yet another tool that you can always use for any mozilla product localization. You have to convert mozilla's native resource file format to GNU PO format and use the available translation tool like KBabel. When the translation is done, change the resource file format back to mozilla's native .rdf and .properties format.

Command moz2po is used to change the format to po format and po2moz is used to change it back to .rdf format. Commands like pomerge and pocount are useful commands that come along with others when installing translate-toolkit.

MozLCDB
MozLCDB is a Tool made for localization of various Mozilla (Gecko) based products. Besides Firefox, this tool can conveniently be used for localization of Thunderbird, Mozilla and Netscape. This tool can be used for both CVS and downloaded ZIP language packs [9.4.k].

9.5 References for Further Reading

a) http://www.mozilla.org/projects/l10n/mlp_chrome.html
b) http://l10nzilla.mozdev.org/
c) http://docs.mandragor.org/files/Misc/Mozilla_applications_en/
d) mozile.mozdev.org/0.6/docs/WWW/mozileDevelopment.html
e) http://www.mozilla.org/projects/l10n/mlp_tools.html
f) http://developer.mozilla.org/en/docs/Linux_Build_Prerequisites
g) http://developer.mozilla.org/en/docs/Gecko_FAQ
h) http://www.mozilla.org/support/firefox/faq
i) http://www.mozilla.org/projects/firefox/l10n/index.html
j) http://blacksapphire.com/firefox-rtl/
k) http://www.moztw.org/tools/mozlcdb/
l) http://en.wikipedia.org/wiki/Complex_Text_Layout_languages
m) http://en.wikipedia.org/wiki/Gecko_(layout_engine

10 OpenOffice.Org Localization

10.1 Introduction

In this Chapter, we will talk about OpenOffice.Org Localization(OOo). We begin with a short introduction about OpenOffice.Org followed by the procedures required for OpenOffice.Org Localization. We then move to the development of the OpenOffice.Org localeand the collation sequence . After giving a general overview of the translation process. We, will also talk about the Spell Checker and Thesaurus Development under the OpenOffice.Org framework. General procedures for building localized OpenOffice.Org in Debian GNU/Linux-based systems are also discussed. References to important links for further reading are provided at the end of the Chapter.

About OpenOffice.Org

OpenOffice.org (OOo) is an office productivity suite comprising of a word processor (Writer), a spreadsheet application (Calc), presentation software (Impress), etc. It supports a variety of file formats and not only does it work with OpenDocument standard from OASIS, it also works with file formats like that of Microsoft Office [10.8.f].

OpenOffice.org is available in a number of languages, with the number increasing with the support of the community. It runs on a number of operating systems such as GNU/Linux, Solaris, Mac OS X, Windows, etc.

There are two major versions in OpenOffice.org till now:

- OpenOffice.org 1.1.x (645er series)
- OpenOffice.org 2.x (680er series)

The build instructions and the localization steps for OpenOffice.org 1.1.x and 2.x are different. This guide discusses how to go about localizing and building the 2.x series. The developmental versions in between the major and minor versions are referred to as 'milestones' and are released frequently. Eg. SRC680_m175.

The official website for OpenOffice.org project is www.openoffice.org and for more information about OpenOffice.org, please refer to [10.8.f].

Historical Background

Sun Microsystems acquired StarDivision, the original author of the StarOffice suite, in 1999 and released StarOffice 5.2 in 2000. Beginning with 6.0, StarOffice software has been built using the OpenOffice.org source, APIs, file formats, and reference implementation. The development of OpenOffice.org is sponsored by Sun and that it is the primary contributor of code to OpenOffice.org. For details about the 'Historical Background', please refer to [10.8.f].

Licenses

OpenOffice.org uses **LGPL** (GNU Lesser General Public License) for the source code [10.8.g]:

For details, please visit http://www.openoffice.org/license.html

In order to contribute code to the project, you must submit the Joint Copyright Assignment (JCA) form. This form jointly assigns copyright over your work to yourself and to Sun Microsystems. Details are available on the "Contributing" page: http://contributing.openoffice.org/programming.html#jca.

10.2 Steps for OpenOffice.Org Localization

As OpenOffice.org is an Open Source project, much of the work depends on the contributions of the volunteers from all around the world. The contributions need not be only in the form of development of source code, but can be in the form of localization and others. So in order to contribute, first register in the OpenOffice.org official site and then join the dev@l10n.openoffice.org mailing list.

The next step is to check whether your language is listed in the http://l10n.openoffice.org/languages.html. If it

already exists, then you can contact the lead for that particular language and express your interest in contributing. If not, then you can write to the dev@l10n.openoffice.org list and express your interest in initiating the contribution in the form of translation**.**

Adding new language to OpenOffice.org source
Certain files, which are given below, need to be modified in order to incorporate new language into OpenOffice.org [10.8.i].

Modification of files becomes much easier and faster when patches are created, submitted and applied. Patch is the difference between the original file and the new changed file. When creating a patch, you make a copy of the original file (eg. lang.h) and change the new copied file. The 'diff' command is used between the original or the old one and the new one [10.8.j]

The syntax of the 'diff' command is:
diff -u [oldfile] [newfile] > [patchfile]

The extension of the file can be either '.diff' or '.patch'.

For instance in our case, diff –u lang.h lang_new.h > lang.h.diff, where -u is for unified diff format.

The simplest way to use the patch is to use the command:
patch -po <patchfile.diff

in the directory where the file to be patched exists and the patchfile.diff is the file to be patched.

When you are done with the file, create an issue in the Localization (L10n) project, and submit a patch for the file. The way to submit an issue for the l10n project in openoffice.org website is [10.8.i]:

- go to http://www.openoffice.org/ and login to the site (if you are already registered that is)
- then go to 'My issues' on the left navigation menubar and click on 'New'
- choose 'l10n' as the component after that one
- select the 'version', 'subcomponent' as code, 'Issue type' as PATCH, and 'Summary' as appropriate
- write a short and precise description for the issue and hit the 'Submit issue' button
- you will be asked if you want to attach a file and the type of the file. Attach the patch and submit it

This is all what you need to do while submitting a patch. The above procedure also applies to other patches, but the only thing that might vary is the component part.

Adding new language to the file i18npool/inc/i18npool/lang.h
Entries for most of the languages are already there in this file. In this file, LANGID's can be found for most of the languages. Sample of the file:

#define LANGUAGE_MONGOLIAN 0x0450
#define LANGUAGE_MONGOLIAN_MONGOLIAN 0x0850
#define LANGUAGE_NEPALI 0x0461
#define LANGUAGE_NEPALI_INDIA 0x0861
#define LANGUAGE_NORWEGIAN 0x0014

The LANGID's, which are defined by Microsoft, can be found at:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/intl/nls_238z.asp

Note: The location and name of the file tools/inc/lang.hxx has changed to i18npool/inc/i18npool/lang.h from previous OOo source.

Creating a unique back and forth mapping between the LANGID and its ISO names in the file i18npool/source/isolang/isolang.cxx

Sample for the file:

```
{ LANGUAGE_MARATHI, "mr", "IN" },
{ LANGUAGE_KONKANI, "kok", "IN" },
{ LANGUAGE_NEPALI, "ne", "NP" },
{ LANGUAGE_NEPALI_INDIA, "ne", "IN" },
{ LANGUAGE_ORIYA, "or", "IN" },
-----------------------------------------------------------------
```

Note: The location and name of the file tools/source/intntl/isolang.cxx has changed to i18npool/source/isolang/isolang.cxx from previous OOo source.

Modifying the file svx/source/dialog/langtab.src

Add an entry for your language in the file svx/source/dialog/langtab.src to see your locale in the listbox of locales. Although English_US and German Entries have to be provided, this might not be needed in the future.

Sample of the file:

```
-----------------------------------------------------------------
< "Manipuri" ; LANGUAGE_MANIPURI ; > ;
< "Marathi" ; LANGUAGE_MARATHI ; > ;
< "Nepalesisch (Nepal)" ; LANGUAGE_NEPALI ; > ;
< "Nepalesisch (Indian)" ; LANGUAGE_NEPALI_INDIA ; > ;
-----------------------------------------------------------------
-----------------------------------------------------------------
< "Manipuri" ; LANGUAGE_MANIPURI ; > ;
< "Marathi" ; LANGUAGE_MARATHI ; > ;
< "Nepali (Nepal)" ; LANGUAGE_NEPALI ; > ;
< "Nepali (India)" ; LANGUAGE_NEPALI_INDIA ; > ;
-----------------------------------------------------------------
```

Defining default fonts for your language

In order to define default fonts for your language, you need to modify a file called officecfg/registry/data/org/openoffice/VCL.xcu [10.8.i]. In this file, you need to define the fonts which, by default, will be used on the user interface of the localized applications such as wordprocessor, spreadsheet, presentation, etc. Not only the user interface, but also the default font to be used in the document can be defined in the file.

Definition of the font default table for any language requires the modification of the file officecfg/registry/data/org/openoffice/VCL.xcu.

A node or a block similar to the following should be added for the language you would like to define the default fonts:

```
<node oor:name="ne" oor:op="replace">
<prop oor:name="UI_SANS" oor:op="replace" oor:type="xs:string">
<value>Lohit Nepali;Kalimati;Samanata;Sans</value>
</prop>
<prop oor:name="CTL_DISPLAY" oor:op="replace" oor:type="xs:string">
<value>Lohit Nepali;Kalimati;Samanata;Sans</value>
</prop>
<prop oor:name="CTL_HEADING" oor:op="replace" oor:type="xs:string">
<value>Lohit Nepali;Kalimati;Samanata;Sans</value>
</prop>
<prop oor:name="CTL_PRESENTATION" oor:op="replace" oor:type="xs:string">
<value>Lohit Nepali;Kalimati;Samanata;Sans</value>
</prop>
<prop oor:name="CTL_SPREADSHEET" oor:op="replace" oor:type="xs:string">
<value>Lohit Nepali;Kalimati;Samanata;Sans</value>
</prop>
<prop oor:name="CTL_TEXT" oor:op="replace" oor:type="xs:string">
<value>Lohit Nepali;Kalimati;Samanata;Sans</value>
```

</prop>
</node>

Points to remember while defining the default fonts for your language [10.8.i]:

- the fonts have to be separated by semi-colon (;) character
- the internal name of the fonts should be used - the one that appears on the font menu when selecting the fonts in OpenOffice.Org or any other program and not the name of the file that contains the font
- though their names might have spaces, this is not a problem, just include them
- while entering different fonts, do not put spaces before or after the ';' signs, nor at the beginning or the end of the font list

Making the required changes when including locale and collation files
The following files have to be modified for the inclusion of locale data [10.8.k]. Please refer to the section 'Developing OpenOffice.org Locale and Collation' for more on the development of locale and collation.

Sample of the file:

i18npool/source/localedata/localedata.cxx

{ "he_IL", lcl_DATA_OTHERS, "he" },
{ "hi_IN", lcl_DATA_OTHERS, "hi" },
{ "ne_NP", lcl_DATA_OTHERS, "ne" },
{ "kn_IN", lcl_DATA_OTHERS, "kn" },
{ "ta_IN", lcl_DATA_OTHERS, "ta" },
i18npool/source/localedata/data/makefile.mk

$(MISC)$/localedata_he_IL.cxx \
$(MISC)$/localedata_hi_IN.cxx \
**$(MISC)$/localedata_ne_NP.cxx **
$(MISC)$/localedata_hr_HR.cxx \
---
and also the following file for CTL languages with around 16 entries:
i18npool/source/localedata/data/localedata_others.map
---
getAllCalendars_ne_NP;
getAllCurrencies_ne_NP;
getAllFormats_ne_NP;
getCollationOptions_ne_NP;
getCollatorImplementation_ne_NP;
getContinuousNumberingLevels_ne_NP;
getFollowPageWords_ne_NP;
getForbiddenCharacters_ne_NP;
getIndexAlgorithm_ne_NP;
getLCInfo_ne_NP;
getLocaleItem_ne_NP;
getOutlineNumberingLevels_ne_NP;
getReservedWords_ne_NP;
getSearchOptions_ne_NP;
getTransliterations_ne_NP;
getUnicodeScripts_ne_NP;

Collation is the sorting order of strings of characters. The following files have to be modified for the inclusion of collation data [10.8.l].

This file has to be created in the source for your langauge:
i18npool/source/collator/data/xx_charset.txt
where, xx corresponds to a two-letter code for your language. In addition, it could also be a four-letter code

for a language, for instance, zh_TW for traditional Chinese charset.
Please refer to the section 'Developing OpenOffice.org Locale and Collation' for more on the development of collation.

The second step is to modify the file i18npool/source/collator/data/collator_data.map
A line similar to the following in bold should be added for your language:

```
-------------------------------------------------------------------
   get_km_charset;
   get_ne_charset;
   get_dz_charset;
-------------------------------------------------------------------
```

The final step is to modify the locale file for your language/country by making sure that the LC_COLLATION section of the file looks like:

```
-------------------------------------------------------------------
  <LC_COLLATION>
    <Collator default="true" unoid="charset" />
    <CollationOptions>
      <TransliterationModules>IGNORE_CASE</TransliterationModules>
    </CollationOptions>
  </LC_COLLATION>
-------------------------------------------------------------------
```

All the locale files are located in the i18npool/source/localedata/data/ directory.

Including your language in the build environment
In order to include your language in the build environment, the language code as defined in http://l10n.openoffice.org/languages.html must be included in the list specified in file solenv/inc/postset.mk [10.8.i]. Sample of the file:

```
-------------------------------------------------------------------
completelangiso=af ar bg ca cs cy da de el en-US eo es et eu fi fr gl he hi-IN hu it ja kn-IN ko lt ms nb ne nl
nn ns pl pt pt-BR ru sk sl sv th tn tr zh-CN zh-TW zu
-------------------------------------------------------------------
```

Adding Support for using native numbers
For a script that is new to to OpenOffice.org, things such as Unicode code-points of the digits in the script, separation characters etc have to be included so the numbers in that script can be used in Calc, or for numbered lists.

The following files need to be modified in order to add support for native numbers, separation characters, etc. in OpenOffice.org[10.8.i]:

- i18npool/source/nativenumber/data/numberchar.h
- i18npool/source/nativenumber/nativenumbersupplier.cxx

In the first file, changes needed are something like this:

```
-------------------------------------------------------
static const sal_Int16 NumberChar_he          = 28;
static const sal_Int16 NumberChar_ne     = 29;
static const sal_Int16 NumberChar_dz          = 30;
-------------------------------------------------------
-------------------------------------------------------
 { 0x0020, 0x05D0, 0x05D1, 0x05D2, 0x05D3, 0x05D4, 0x05D5, 0x05D6, 0x05D7, 0x05D8 }, // Hebrew
 { 0x0966, 0x0967, 0x0968, 0x0969, 0x096A, 0x096B, 0x096C, 0x096D, 0x096E, 0x096F }, // Nepali
 { 0x0F20, 0x0F21, 0x0F22, 0x0F23, 0x0F24, 0x0F25, 0x0F26, 0x0F27, 0x0F28, 0x0F29 }, // Dzongkha
-------------------------------------------------------
for number character.
-------------------------------------------------------
   0x0000, // Hebrew
   0x0000, // Nepali
   0x0000, // Dzongkha
-------------------------------------------------------
```

for decimal character.

0x0000, // Hebrew
0x0000, // Nepali
0x0000, // Dzongkha
---
for minus character.

0x0000, // Hebrew
0x0000, // Nepali
0x0000, // Dzongkha
---
for separator character.
In the second file, you need to include your script in both the natnum1Locales[] and natnum1[] arrays.

---
static const sal_Char *natnum1Locales[] = { "zh_CN", "zh_TW", "ja", "ko", "he", "ar", "th", "hi", "or", "mr", "bn", "pa", "gu", "ta", "te", "kn", "ml", "lo", "bo", "my", "km", "mn", "**ne**","dz" };

NumberChar_mn, **NumberChar_ne**, NumberChar_dz

It is suggested to send in a single patch that covers changes to these two files.
If this is already done, then you can - for example - format cells in Calc by using:
[NatNum1]

Adding support for numbering in local language numbers and letters
The number lists in OpenOffice.org Writer can be used in local language numbers instead of Latin numbers. This defined styles in OOo Writer can be found in Format>Bullets and Numbering->Numbering type tab. To change these styles, the necessary changes have to be done in this part of your locale file [10.8.k]:

```
<LC_NumberingLevel>
   <NumberingLevel NumType="12" Prefix=" " Suffix="."/>
   <NumberingLevel NumType="12" Prefix=" " Suffix=")"/>
   <NumberingLevel NumType="4" Prefix=" " Suffix="."/>
   <NumberingLevel NumType="4" Prefix=" " Suffix=")"/>
   <NumberingLevel NumType="34" Prefix=" " Suffix="."/>
   <NumberingLevel NumType="34" Prefix=" " Suffix=")"/>
   <NumberingLevel NumType="2" Prefix=" " Suffix="."/>
   <NumberingLevel NumType="2" Prefix=" " Suffix=")"/>
</LC_NumberingLevel>
```

If you want to use numbers in the script of your locale, you need to use NumType="12". For more information, please see the file: offapi/com/sun/star/style/NumberingType.idl.

It is a little more complicated to use the letters of the script instead of numbers (equivalent to using A,B,C,D... in English). We have to define the style first, only after which the letters included in the <IndexKey> of the locale file will be used.

Using letters requires patching a few files [10.8.i]:

In the file offapi/com/sun/star/style/NumberingType.idl, you need to add a new line with a new number, including something like this:

```
   /** Numbering in Nepali alphabet letters

       @since OOo 2.0.1
        */
   const short CHARS_NEPALI = 34;
```

---
in which the number is the next one after the last that you find in the file. For instance, the number '34' is the next number after the last one ('33' in this case) that you find in the file.

In the file i18npool/source/defaultnumberingprovider/defaultnumberingprovider.cxx,
you need to include an entry of the type

```
  case CHARS_NEPALI:
  lcl_formatChars(table_Alphabet_ne, sizeof(table_Alphabet_ne) /   sizeof(sal_Unicode), number - 1, result);
 break;
```
---
in **DefaultNumberingProvider::makeNumberingString**

and then include a line of the type:

```
      {style::NumberingType::CHARS_NEPALI,    NULL, LANG_CTL},
```
---

in the correct position in **static const Supported_NumberingType aSupportedTypes[]**

Note that the last element is LANG_CTL, defining the language as CTL (this will appear in the menus only if CTL is activated), here you can also use LANG_CJK or LANG_ALL.

Finally, you have to define the table of permitted characters (table_Alphabet_ne) in the file i18npool/inc/bullet.h with a block such as:

```
static sal_Unicode table_Alphabet_ne[] = {
      0x0915, 0x0916, 0x0917, 0x0918, 0x0919, 0x091A, 0x091B, 0x091C,
      0x091D, 0x091E, 0x091F, 0x0920, 0x0921, 0x0922, 0x0923, 0x0924,
      0x0925, 0x0926, 0x0927, 0x0928, 0x092A, 0x092B, 0x092C, 0x092D,
      0x092E, 0x092F, 0x0930, 0x0932, 0x0935, 0x0936, 0x0937, 0x0938,
      0x0939
};
```

Getting the 'Help' contents in your script
In order to have the help contents of OpenOffice.org in your script (if other than Latin), it is necessary to localize some stylesheets besides translating the help files [10.8.i].

The directory where the files for different languages are located is:
helpcontent2/source/auxiliary/

The contents of the en-US directory should be copied to a directory of your iso name and the files have to be localized. Special attention should be given to the fonts defined in the css files as they mark the script that will be used in the help. If the first font in the lists does not contain your script, then the help will not be displayed correctly.

10.3 Developing OpenOffice.Org Locale and Collation

Note: Most of this section has been based on [10.8.k] and [10.8.l].

Locale in OpenOffice.org
Programs that are localized to many languages tend to place all the data related to a language or to a country (region) in a file called a LOCALE for that culture.

OpenOffice.org requires that you place all the cultural data for your language/region in a file that has a format specific to OpenOffice.org. This file is an XML file that is plain text (utf-8 if your language requires it). It can be edited with any plain text editor.

All locale files are located in the i18npool/source/localedata/data directory. Eg. ne_NP.xml, en_US.xml, etc. The file name of the locale file in OpenOffice.Org always uses both the language code and the country code (language code in small letters and country code in capital letters), separated by an underscore (not a hyphen) and with an .xml extension.

Locale Developement
Earlier, the locale file had to be created manually with the help of a template file, the template file being en_US.xml. But this process has been made much easier with "Locale Generator", which is a web based tool to support creation of locale files. Please visit http://www.it46.se/localegen/ for the above purpose.

The first part of the locale contains Locale File Specific Information such as:

- Filename version
- OpenOffice.org DTD Version

A sample of this part:
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Locale SYSTEM 'locale.dtd'>
<Locale versionDTD="2.0" allowUpdateFromCLDR="yes" version="1.2">
```

The default values for this part need not be modified.
The locale file is divided into the following sections:

<LC_INFO>
In this section, you need to provide the Language Specific Data:

- Language Code (Available in ISO-639)
- Language Name
- Country Code (Available in ISO-3166. Capital letters)
- Country Name

Sample of this section:

```
 <LC_INFO>
   <Language>
     <LangID>en</LangID>
     <DefaultName>English</DefaultName>
   </Language>
   <Country>
     <CountryID>US</CountryID>
     <DefaultName>United States</DefaultName>
   </Country>
 </LC_INFO>
```

<LC_CTYPE>
In this section, you need to provide Separators and Quotation Marks.

Separators:

- Thousand
  - 1,000 - 1.000 - 1 000
- Decimal
  - 10.00 - 10,00
- List
  - a, b, c - a; b; c
- Date
  - 19/02/2006 - 19-02-2006 - 19.02.2006
- Time
  - 12:30 - 12.30 - 12,30
- Miliseconds

    - 1.12 secs - 1,12 secs - 1:12 secs
- Day of Week, Long name
    - When using long date format as in: Wednesday, March 12, 2023.
      The separator after Wednesday is ', '
- Day of Week, Number
    - When using long date format as in: Wednesday, March 12, 2023.
      The separator after 12 is ', '
- Month, Long name
    - Wednesday, March 12, 2023.
      The month separator after March is one space
- Year
    - When using four digits instead of two

Quotation Marks:

- Quotation Start
- Quotation End
- Double Quotation start
- Double Quotation end

You also need to provide TimeAM and TimePM and the Measurement System metric US UK.

Sample for this section:

```
 <LC_CTYPE>
   <Separators>
     <DateSeparator>/</DateSeparator>
     <ThousandSeparator>,</ThousandSeparator>
     <DecimalSeparator>.</DecimalSeparator>
     <TimeSeparator>:</TimeSeparator>
     <Time100SecSeparator>.</Time100SecSeparator>
     <ListSeparator>;</ListSeparator>
     <LongDateDayOfWeekSeparator>, </LongDateDayOfWeekSeparator>
     <LongDateDaySeparator>, </LongDateDaySeparator>
     <LongDateMonthSeparator> </LongDateMonthSeparator>
     <LongDateYearSeparator> </LongDateYearSeparator>
   </Separators>
   <Markers>
     <QuotationStart></QuotationStart>
     <QuotationEnd></QuotationEnd>
     <DoubleQuotationStart></DoubleQuotationStart>
     <DoubleQuotationEnd></DoubleQuotationEnd>
   </Markers>
   <TimeAM>AM</TimeAM>
   <TimePM>PM</TimePM>
   <MeasurementSystem>US</MeasurementSystem>
 </LC_CTYPE>
```

<LC_FORMAT>
In this section, you need to provide Date format, Currency format, MS Locale Identifier (LCID), and Percentage.

- Default order of Year, Month and Date
    - Choose the most common order of Day, Month and Year in a date
- Decimal places
    - Number of decimal (digits) used with the currency
- Currency format for positive values
    - Position of symbol, space between symbol and digits
- Currency format for negative values

    - Position of symbol and minus sign, space between symbol and digits
- MS Locale Identifier, LCID
- Percentage (short)
    - Without using decimals
    - 0%0 %
- Percentage (long)
- When using decimals

Sample for this section:

```
 <LC_FORMAT replaceFrom="[CURRENCY]" replaceTo="[$$-409]">
 <FormatElement msgid="FixedFormatskey1" default="true" type="medium" usage="FIXED_NUMBER"
formatindex="0">
 <FormatCode>General</FormatCode>
 </FormatElement>
 <FormatElement msgid="FixedFormatskey2" default="true" type="short" usage="FIXED_NUMBER"
formatindex="1">
 <FormatCode>0</FormatCode>
 </FormatElement>
          .
          .
          .
  <FormatElement msgid="PercentFormatskey1" default="true" type="short" usage="PERCENT_NUMBER"
formatindex="8">
    <FormatCode>0%</FormatCode>
  </FormatElement>
  <FormatElement msgid="PercentFormatskey2" default="true" type="long" usage="PERCENT_NUMBER"
formatindex="9">
    <FormatCode>0.00%</FormatCode>
  </FormatElement>
  <FormatElement msgid="CurrencyFormatskey1" default="true" type="short" usage="CURRENCY"
formatindex="12">
    <FormatCode>[CURRENCY]#,##0;-[CURRENCY]#,##0</FormatCode>
  </FormatElement>
  <FormatElement msgid="CurrencyFormatskey2" default="false" type="medium" usage="CURRENCY"
formatindex="13">
    <FormatCode>[CURRENCY]#,##0.00;-[CURRENCY]#,##0.00</FormatCode>
  </FormatElement>
          .
          .
          .
  <FormatElement msgid="DateFormatskey1" default="true" type="short" usage="DATE" formatindex="18">
    <FormatCode>M/D/YY</FormatCode>
  </FormatElement>
          .
          .
          .
 </LC_FORMAT>
```

<LC_COLLATION>
Please refer to the Collation section for more detailed information on this.

```
 <LC_COLLATION>
<Collator default="true" unoid="alphanumeric" />
<CollationOptions>
<TransliterationModules>IGNORE_CASE</TransliterationModules>
</CollationOptions>
 </LC_COLLATION>
```

<LC_SEARCH>

Sample of this section:

```
<LC_SEARCH>
<SearchOptions>
<TransliterationModules>IGNORE_CASE</TransliterationModules>
</SearchOptions>
 </LC_SEARCH>
```

<LC_INDEX>

In this section, you need to provide Enumeration and Scripts.

- Character range for indexes
  - A-Z Ä Ö
- Abbreviation used for 'the following page (singular)'
  - p. s. Стр.
- Abbreviation used for 'the following pages (plural)'
  - pp. ss. Стр.
- Unicode Script

Sample of this section:

```
 <LC_INDEX>
   <IndexKey phonetic="false" default="true" unoid="alphanumeric">A-Z</IndexKey>
   <UnicodeScript>0</UnicodeScript>
   <UnicodeScript>1</UnicodeScript>
   <FollowPageWord>p.</FollowPageWord>
   <FollowPageWord>pp.</FollowPageWord>
 </LC_INDEX>
```

<LC_CALENDAR>
In this section, you need to provide Days of the week, Months of the year,i.e. Gregorian Calendar Specific Information.

- Days of the week:
  - Gregorian Calendar, Short and Long names of days of the week
    - Sun : Sunday
    - Mon : Monday
    - Tue : Tuesday
    - Wed : Wednesday
    - Thu : Thursday
    - Fri : Friday
    - Sat : Saturday

- Months of the year:
  - Gregorian Calendar,abbreviated and full names of months of the year
    - Jan : January
    - Feb : February
    - Mar : March
    - Apr : April
    - May : May
    - Jun : June
    - Jul : July
    - Aug : August
    - Sep : September
    - Oct : October
    - Nov : November

        - Dec : December

- Gregorian Calendar Specific Information:
- BC : Before Christ
- Short and Long names for BC
- AD : Anno Domini
- Short and Long names for AD
- First day of the week
- In a calendar, indicate what is the first day of the week
- sun mon tue wed thu fri sat

- Minimum number of days in first week of the year
    - The number of days of a week that must reside in the beginning of a year to make a week the first week of the year
        - 1 2 3 4 5 6 7

Sample of this section:

```
 <LC_CALENDAR>
  <Calendar unoid="gregorian" default="true">
   <DaysOfWeek>
    <Day>
     <DayID>sun</DayID>
     <DefaultAbbrvName>Sun</DefaultAbbrvName>
     <DefaultFullName>Sunday</DefaultFullName>
    </Day>
    <Day>
     <DayID>mon</DayID>
     <DefaultAbbrvName>Mon</DefaultAbbrvName>
     <DefaultFullName>Monday</DefaultFullName>
    </Day>
    <Day>
     <DayID>tue</DayID>
     <DefaultAbbrvName>Tue</DefaultAbbrvName>
     <DefaultFullName>Tuesday</DefaultFullName>
    </Day>
    <Day>
     <DayID>wed</DayID>
     <DefaultAbbrvName>Wed</DefaultAbbrvName>
     <DefaultFullName>Wednesday</DefaultFullName>
    </Day>
    <Day>
     <DayID>thu</DayID>
     <DefaultAbbrvName>Thu</DefaultAbbrvName>
     <DefaultFullName>Thursday</DefaultFullName>
    </Day>
    <Day>
     <DayID>fri</DayID>
     <DefaultAbbrvName>Fri</DefaultAbbrvName>
     <DefaultFullName>Friday</DefaultFullName>
    </Day>
    <Day>
     <DayID>sat</DayID>
     <DefaultAbbrvName>Sat</DefaultAbbrvName>
     <DefaultFullName>Saturday</DefaultFullName>
    </Day>
   </DaysOfWeek>
   <MonthsOfYear>
    <Month>
```

```
    <MonthID>jan</MonthID>
    <DefaultAbbrvName>Jan</DefaultAbbrvName>
    <DefaultFullName>January</DefaultFullName>
  </Month>
  <Month>
    <MonthID>feb</MonthID>
    <DefaultAbbrvName>Feb</DefaultAbbrvName>
    <DefaultFullName>February</DefaultFullName>
  </Month>
  <Month>
    <MonthID>mar</MonthID>
    <DefaultAbbrvName>Mar</DefaultAbbrvName>
    <DefaultFullName>March</DefaultFullName>
  </Month>
  <Month>
    <MonthID>apr</MonthID>
    <DefaultAbbrvName>Apr</DefaultAbbrvName>
    <DefaultFullName>April</DefaultFullName>
  </Month>
  <Month>
    <MonthID>may</MonthID>
    <DefaultAbbrvName>May</DefaultAbbrvName>
    <DefaultFullName>May</DefaultFullName>
  </Month>
  <Month>
    <MonthID>jun</MonthID>
    <DefaultAbbrvName>Jun</DefaultAbbrvName>
    <DefaultFullName>June</DefaultFullName>
  </Month>
  <Month>
    <MonthID>jul</MonthID>
    <DefaultAbbrvName>Jul</DefaultAbbrvName>
    <DefaultFullName>July</DefaultFullName>
  </Month>
  <Month>
    <MonthID>aug</MonthID>
    <DefaultAbbrvName>Aug</DefaultAbbrvName>
    <DefaultFullName>August</DefaultFullName>
  </Month>
  <Month>
    <MonthID>sep</MonthID>
    <DefaultAbbrvName>Sep</DefaultAbbrvName>
    <DefaultFullName>September</DefaultFullName>
  </Month>
  <Month>
    <MonthID>oct</MonthID>
    <DefaultAbbrvName>Oct</DefaultAbbrvName>
    <DefaultFullName>October</DefaultFullName>
  </Month>
  <Month>
    <MonthID>nov</MonthID>
    <DefaultAbbrvName>Nov</DefaultAbbrvName>
    <DefaultFullName>November</DefaultFullName>
  </Month>
  <Month>
    <MonthID>dec</MonthID>
    <DefaultAbbrvName>Dec</DefaultAbbrvName>
    <DefaultFullName>December</DefaultFullName>
  </Month>
</MonthsOfYear>
```

```
    <Eras>
     <Era>
      <EraID>bc</EraID>
      <DefaultAbbrvName>BC</DefaultAbbrvName>
      <DefaultFullName>BC</DefaultFullName>
     </Era>
     <Era>
      <EraID>ad</EraID>
      <DefaultAbbrvName>AD</DefaultAbbrvName>
      <DefaultFullName>AD</DefaultFullName>
     </Era>
    </Eras>
    <StartDayOfWeek>
     <DayID>sun</DayID>
    </StartDayOfWeek>
    <MinimalDaysInFirstWeek>1</MinimalDaysInFirstWeek>
  </Calendar>
 </LC_CALENDAR>
```

<LC_CURRENCY>
In this section, you need to provide the Currency details for your locale.

- Bank code
    - Three letter code in capitals
- Currency symbol
- Currency name
- Localized name of currency (singular)
- Decimal places
    - Number of decimal (digits) used with the currency

Sample for this section:

```
 <LC_CURRENCY>
  <Currency default="true" usedInCompatibleFormatCodes="true">
   <CurrencyID>USD</CurrencyID>
   <CurrencySymbol>$</CurrencySymbol>
   <BankSymbol>USD</BankSymbol>
   <CurrencyName>US Dollar</CurrencyName>
   <DecimalPlaces>2</DecimalPlaces>
  </Currency>
 </LC_CURRENCY>
```

<LC_TRANSLITERATION>
This section is used to specify character conversion algorithms. Languages that use Latin characters do use upper to lowercase conversion algorithms, such as the ones mentioned in this file.

Some languages, such as Japanese, Chinese or Korean, have complicated transliteration schemes. Specific transliteration could be used, but it has not been used in other scripts before. Transliteration procedures need to be written before they are included here. If your language is not Chinese, Korean or Japanese, just leave this as it is and if it is one of them, then it has already been done.

If you really want to do transliteration, please look into:
http://l10n.openoffice.org/i18n_framework/HowToAddLocaleInI18n.html

Sample of this section:

```
 <LC_TRANSLITERATION>
  <Transliteration unoid="LOWERCASE_UPPERCASE"/>
  <Transliteration unoid="UPPERCASE_LOWERCASE"/>
```

```
  <Transliteration unoid="IGNORE_CASE"/>
 </LC_TRANSLITERATION>
```

<LC_MISC>
In this section, you need to provide the Reserved Words.

- Reserved Words
    - True
    - False
    - Quarter 1 (full name)
    - Quarter 2 (full name)
    - Quarter 3 (full name)
    - Quarter 4 (full name)
    - Quarter 1 (abbreviation)
    - Quarter 2 (abbreviation)
    - Quarter 3 (abbreviation)
    - Quarter 4 (abbreviation)
    - Above
    - Below

Sample for this section:

```
 <LC_MISC>
  <ReservedWords>
   <trueWord>true</trueWord>
   <falseWord>false</falseWord>
   <quarter1Word>1st quarter</quarter1Word>
   <quarter2Word>2nd quarter</quarter2Word>
   <quarter3Word>3rd quarter</quarter3Word>
   <quarter4Word>4th quarter</quarter4Word>
   <aboveWord>above</aboveWord>
   <belowWord>below</belowWord>
   <quarter1Abbreviation>Q1</quarter1Abbreviation>
   <quarter2Abbreviation>Q2</quarter2Abbreviation>
   <quarter3Abbreviation>Q3</quarter3Abbreviation>
   <quarter4Abbreviation>Q4</quarter4Abbreviation>
  </ReservedWords>
 </LC_MISC>
```

<LC_NumberingLevel>
In this section, you need to provide the Numbering styles for paragraphs. Please refer to 'Adding support for numbering in local language numbers and letters' section for more on this.

Sample of this section:

```
 <LC_NumberingLevel>
  <NumberingLevel NumType="4" Prefix=" " Suffix=")"/>
  <NumberingLevel NumType="4" Prefix=" " Suffix="."/>
  <NumberingLevel NumType="4" Prefix="(" Suffix=")"/>
  <NumberingLevel NumType="2" Prefix=" " Suffix="."/>
  <NumberingLevel NumType="0" Prefix=" " Suffix=")"/>
  <NumberingLevel NumType="1" Prefix=" " Suffix=")"/>
  <NumberingLevel NumType="1" Prefix="(" Suffix=")"/>
  <NumberingLevel NumType="3" Prefix=" " Suffix="."/>
 </LC_NumberingLevel>
```

If you want to use numbers in the script of your locale, you need to use NumType="12".

<LC_OutLineNumberingLevel>

In this section of the locale file, you have to define what styles will be included in OpenOffice.org Writer in Format-->Bullets and Numbering-->Numbering type tab. Each <OutlineStyle> block in the locale file defines one of the 8 squares in this page. Each block has five lines defining the first five levels of heading. The first line of a block (for example) will define how paragraphs with style 'Heading 1' will be numbered (including number style, and characters to be placed before and after the number), if some bullet characters should be used, the left margin of Heading 1 numbered paragraphs, etc. In subsequent levels (other lines), it is important to say how many levels of headings will be named in this specific number. For example, if we are defining the numbering of level 3 (Heading 3), the ParentNumbering could be 0 (in which case only one number will be shown) or could be 1 (two numbers will be shown, as in 1.1) or 2, in which case we will have numbers of the style 1.1.1.

Sample of this section:

```
 <LC_OutLineNumberingLevel>
<OutlineStyle>
<OutLineNumberingLevel Prefix=" " NumType="4" Suffix="." BulletChar="0020" BulletFontName=""
ParentNumbering="0" LeftMargin="0" SymbolTextDistance="50" FirstLineOffset="0"/>
        .
        .
        .
 <OutLineNumberingLevel Prefix=" " NumType="6" Suffix=" " BulletChar="2022"
BulletFontName="StarSymbol" ParentNumbering="0" LeftMargin="200" SymbolTextDistance="50"
FirstLineOffset="0"/>
  </OutlineStyle>
</LC_OutLineNumberingLevel>

</Locale>
```

OpenOffice.org Locale submission

When you have completed works with the locale definition file, you need to create an issue in the Localization (L10n) project, and submit a patch for the locale file.

To submit an issue

- go to http://www.openoffice.org/ and login to the site (if you are already registered that is)
- then go to 'My issues' on the left navigation menubar and click on 'New'
- choose 'l10n' as the component after that one
- select the 'version', 'subcomponent' as code, 'Issue type' as ENHANCEMENT, and 'Summary' as appropriate
- write a short and precise description for the issue and hit the 'Submit issue' button
- you will be asked if you want to attach a file and the type of the file. Attach the locale file and submit it

This is all what you need to do while submitting a locale file.

Collation mechanism in OpenOffice.Org

The following file related to the collation sequence has to be created in the source in the location [10.8.l]: i18npool/source/collator/data/xx_charset.txt

This above file contains one or more reordering sequences with a syntax of the style:

```
-------------------------------------------------------------
&n < ñ <<< Ñ < o
-------------------------------------------------------------
```

Need to add that the file should be in UTF-8 format.

To know exactly how the collation mechanism works, please refere to the following document, which explains in detail the tailoring syntax and philosophy:
http://icu.sourceforge.net/userguide/Collate_Customization.html

10.4 Translation works

Translation strings for OpenOffice.org
The translation of OpenOffice.org consists of the OpenOffice.org software itself and the help files. There are about 25,000+ messages to be translated with additional 43,000+ messages in the help files [10.8.n]. All the the standard PO (portable objects) file format.

Translation Tools
There are a variety of translation tools available for both GNU/Linux and Windows environment. Some of the widely used translation tools are:

1) Kbabel
2) POEdit

Web-based translation tools like Pootle, Rosetta, Entrans are also becoming increasingly popular as they allow people spread across different places to contribute to the translation process. For detailed information on the translation tools, please refer to Chapter 13, Tools for Localization under Translation tools.

Prerequisites for Translation
We will discuss the different prerequisites for translation below,.

Downloading the POT files and the GSI files
The first step in the translation of OpenOffice.org is the downloading of the POT files and the GSI file which can be found at [10.8.m]:

http://ftp.linux.cz/pub/localization/OpenOffice.org/devel/POT/

Here, you will find the latest tarred version of the POT files. The downloaded file consists of the .pot files and the en-US version of the GSI/SDF file, which would later be needed for the creation of the language-specific OpenOffice.org-format file called GSI/SDF file after the completion of the translation. Please note that the same GSI/SDF file will be needed later when converting the .po format files to the language-specific GSI/SDF file.

Installing Translate Toolkit
It is a toolkit to convert various translation formats (such as gettext-based .po formats, OpenOffice.org formats, and Mozilla formats). This makes it possible to stay in one format across all your localization. It consists of tools to help process and validate localizations, etc [10.8.o].

The toolkit can be downloaded from the following location:
http://translate.sourceforge.net/

In order to install it, you can use the following sequence of commands:

```
tar zxvf translate-toolkit-0.8rc5.tar.gz
cd translate-toolkit-0.8rc5
./setup.py install
```

Generating PO files from POT files
The downloaded POT file has to be converted into a PO file first in order to start the translation [10.8.p]. First untar the file:

tar zxvf OpenOffice.org-SRC680_m<MILESTONE>-POT.tar.gz

Then use the command 'pot2po' from the translate toolkit to get the PO files:
pot2po -i pot -o po

where pot is the directory containing pot files and po is the resulting directory containing po files with OpenOffice.org's own directory structure.
pot2po --help on the command line will list all the options.

It is also to be noted that the same command with different options can be used when upgrading from one milestone (or version) to another since the newer milestone can have added words for translation.

Translation activity
The next step is the translation itself. As already mentioned above, there are quite a number of translation tools available. Translation can done using any of the tools like KBabel, POEdit, etc.

After the completion of all translations, the po files should be converted to OpenOffice.org format i.e. to GSI/SDF file which is explained below.

Generating the final GSI/SDF file from po files
With the translation completed, you have everything ready to transform your PO files to GSI/SDF (OpenOffice.org format) file using the following command[10.8.p]:

po2oo -i po -o GSI_ne.sdf -t en-US.sdf -l ne

where ne is the ISO code for your language (ne if the language is Nepali), en-US.sdf is one of the GSI/SDF files used for generating POT files. po is the name of the directory with PO files and GSI_ne.sdf is the final GSI/SDF file that can be used with localize tool to merge into the source.

The final GSI/SDF file should be submitted as an issue to OpenOffice.org to be included in the official source.

10.5 Building Localized OpenOffice.org in Debian GNU/Linux-Based Systems

Building OpenOffice.org is a big challenge in the sense that it is time-consuming and that there are lots of modules inside OpenOffice.org to be compiled. Though the compilation time varies for different systems with different configurations, compiling in GNU/Linux-based systems takes more than 12 hours in a system with high resources. For any questions or suggestions/comments related to development and building, you should write to the dev@openoffice.org mailing list.

This part of the localization guide describes the process to build localized OpenOffice.org on GNU/Linux. And the GNU/Linux distribution that has been used is a Debian-based Linux system.

Note: $ooo_src will denote the directory in which the source code of OpenOffice.org is stored.

Build requirements [10.8.r]
Hardware requirements:

- Intel Pentium II (P4 recommended)
- 256 MB RAM (highly recommended)
- 4 Gb free disk space, add approximately 2 Gb to build with --with-lang=ALL option

Software requirements:

- **glibc 2.1.x or higher:** The GNU C Library - It contains the standard libraries that are used by nearly all programs on the system. This package includes shared versions of the standard C library and the standard math library, as well as many others. Timezone data is also included.
- **gcc:** The GNU C compiler - It is a fairly portable optimized compiler for C.
- **g++:** The GNU C++ compiler – It is a fairly portable optimized compiler for C++.
- The **X11 development libraries and header files** should be installed. These should be in place with most GNU/Linux distributions.
- **PAM:** Pluggable Authentication Module - It should come with most GNU/Linux distributions. The development package must be installed for your distribution.
- **JDK 1.3.1 or JDK 1.4.2:** It is to be noted that not just the JRE (Java Runtime Environment), but the SDK (Software Development Kit) is also needed. **JDK 1.5.0** is supported starting from milestone m158.
    - To install JDK 1.4.2,
    - Go to http://java.sun.com/j2se/1.4.2 and download the required package

    - Then, execute the file like this: ./j2sdk-1_4_2_02-linux-i586.bin
- If you are building with JDK 1.3.1, you need to download **crimson.jar** from http://xml.apache.org/crimson/ and **xalan.jar** and **xml-apis.jar** from http://xml.apache.org/xalan-j/index.html and add these to the compilation classpath
- **Perl 5:** Practical Extraction and Report Language - Perl is optimised for scanning arbitrary text files and system administration. It has built-in extended regular expression matching and replacement, a data-flow mechanism to improve security with setuid scripts and is extensible via modules that can interface to C libraries.
- **tcsh** – It is an enhanced but completely compatible version of the Berkeley UNIX C shell, csh. Although the build can be started in bash, all the scripts in the build system are actually csh scripts.
- **zip** and **unzip** - Archiver and De-archiver for .zip files respectively
- **General Polygon Clipper library (gpc):** A flexible and highly robust polygon set operations library for use with C applications.
- To get gpc files,
- Go to http://www.cs.man.ac.uk/aig/staff/alan/software/ and download the required file
- Unpack the tarball like this: unzip gpc232.zip
- You should have the files gpc.c and gpc.h in $ooo_src/external/gpc by doing as follows:
- cp gpc232/* ooo_SRC680_m<MILESTONE>_src/external/gpc/
- **Ant:** Apache Ant is a Java-based build tool.
    - To install Apache-Ant,
    - Go to http://ant.apache.org/ and download the required package
    - Then unzip the file like this: unzip apache-ant-1.6.2-bin.zip
- **Mozilla libraries**: Some Mozilla libraries are needed. You can choose one of the following three options:
  Build the libraries
    - Get the source from http://ftp.mozilla.org/pub/mozilla.org/mozilla/releases/mozilla1.7.5/source/, copy it into $ooo_src/moz/download and remember to configure with --enable-build-mozilla .
    - **Use pre-built libraries**
    - Get the files from http://tools.openoffice.org/moz_prebuild/680/
    - Place LINUXGCCI{inc,lib,runtile}.zip into $ooo_src/moz/zipped.

  Don't use the libraries
    - By using the --disable-mozilla switch for configure, you waive the extra functionality.
- **Perl Modules:**
    - **Archive::Zip -** It is used for packing image lists, eventually for further zipping needs
    - **XML::Parser** – It is used for expat-based parser for the new XML based build lists

To install a perl module [10.8.r],
Start the CPAN module.
$ perl -MCPAN -e shell
CPAN is the **C**omprehensive **P**erl **A**rchive **N**etwork, a large collection of Perl software and documentation. Note that CPAN is also the name of a Perl module, CPAN.pm, which is used to download and install Perl software from the CPAN archive [10.8.r].

If this is the first time you use this module you have to answer some questions for this module. Just follow the directions on your screen.

This will get you into cpan shell.
For example, if you want to install the Archive::Zip module:
cpan> install Archive::Zip

Typing help gets you some online help.
cpan> help
And typing quit quits the module.
cpan> quit

For details, please visit http://wiki.services.openoffice.org/wiki/CPAN_install

Getting the source code
There are two options to get the source code:

1. **Downloading the source code tarball**
   a) http://download.openoffice.org/2.0.0/source.html
      For example, OOo_2.0.0_src.tar.gz in case of the 2.0 stable release.
   b) Unpack the tarball as follows:
      tar zxvf OOo_2.0.0_src.tar.gz
      cd Ooo_2.0.0rc3_src
      Note: This will be $ooo_src from now on.
2. **Checking out the code from cvs as anoncvs if you don't have a username and password**
   a) cvs -d:pserver:anoncvs@anoncvs.services.openoffice.org:/cvs login
      Just press enter when prompted for the password.
   b) cd $ooo_src
      cvs -d:pserver:anoncvs@anoncvs.services.openoffice.org:/cvs co -r OpenOffice_2_0_0 OpenOffice

Setting up the build environment and generating the build tools
The configure script, which is used to check/prepare the build environment, checks all the requirements such as software, hardware and system requirements for the build and creates a configuration file called LinuxIntelEnv.Set. This configuration file is used to set all necessary build environment variables.

The environment variables CC and CXX should point to c and c++ compiler if the compiler running has a non-standard name or location.

export CC=/your/path/to/gcc
export CXX=/your/path/to/g++

To run the configure script, first go to the config_office directory and run the configure script:
$ooo_src> cd config_office
config_office> ./configure --with-lang="**ne**"

The –with-lang option is for the inclusion of Nepali language in the build, which could of course be any other language. Here, Nepali is used as an example.

A number of options can be used with the configure script. Type the following command:

config_office> ./configure --help

to display a list of options.

After running the configure script, change to tcsh shell:

$ooo_src> tcsh

The next step is to create the dmake make utility needed for the OpenOffice.org build:

$ooo_src> ./bootstrap

Now source the above-created configuration file to set all the environment variables:

$ooo_src> source LinuxIntelEnv.Set

Preparing Localization Tools: Building transex3 module first [10.8.q]
The transex3 module is required to build first in order to build the tools for merging the translations into the source code. If this module is not built first, then the build would have to be performed twice for the localized builds.

To build transex3 module,

```
$ooo_src> cd transex3
transex3> build --all
transex3> deliver
```

The required localization tools for OpenOffice.org are built after the transex3 module is built. It is a good practice to check the GSI/SDF file created for your language. It can be done with the 'gsicheck' command. The 'gsicheck' command:

- checks the syntax of tags in GSI-Files and SDF-Files
- checks for inconsistencies and malicious UTF8 encoding
- checks tags in Online Help

Commonly, gsicheck is used like this:

```
$SRC_ROOT> gsicheck -c -l "" GSI_ne.sdf
```

where,

-c is to add context to error message i.e. to print the line containing the error
-l is the ISO language code or numerical 2 digits identifier of the source language. The default is en-US. Use "" (empty string) or 'none' to disable source language dependent checks.
And of course, the last one is the filename of the language-specific GSI/SDF file.

For more options,

```
$SRC_ROOT> gsicheck --help
```

Removing existing language-specific translations from source
If the translation has not been merged into the source previously, then this step can be ignored. Only if some translation exists in the source, are you required to remove existing language-specific translations from the source code. Your need to remove the lines which contain the language-specific translations from all the localize.sdf files [10.8.q]. The localize.sdf files are those files which contain the translations for all the languages and into which the translations are merged, which is explained in the next section.

```
$ooo_src> for localize in `find . -name localize.sdf`
> do
>  grep -v "     ne      " $localize >$localize.tmp
>  mv -f $localize.tmp $localize
> done
```

Merging the latest translations to source
After the removal of all the translations for a specific language, the next step is the merging of the latest translations to source before starting the actual build [10.8.q]. The same localized tool can be used, but with different options, to extract the translations.

This is how the merging can be done:

```
$ooo_src> localize -m -l ne -f GSI_ne.sdf
```

where,

-m is the merge mode
-l is the comma separated languages
-f is the filename of the language-specific GSI/SDF file

For more options,

```
$ooo_src> gsicheck –help
```

Building a full build
With the environment created to build the entire suite, all that is required is to run dmake from the top-level directory. This will take more than 12 hours depending upon the available resources.

$ooo_src>> dmake

Similarly, the 'build --all' inside the instsetoo_native directory is equivalent to the dmake command in the root of the source directory.

cd $ooo_src
cd instetoo_native
build --all

Built packages will be in different directories in $ooo_src/instsetoo_native/unxlngi6/OpenOffice/deb/install/ depending on the built language(s). But note that the en-US language will always be built by default even if you don't specify it when configuring.

To clean up a previous build, you have to delete all the output directories:

$ooo_src> rm -rf */unxlngi6.pro

To rebuild a module or build each module individually, you will have to use the 'build' tool and then the 'deliver' tool will copy all created binaries, libraries etc. into the solver tree:

$ooo_src/(module)> build
$ooo_src/(module)> deliver

Please refer to [10.8.r] for more details on building OpenOffice.org 2.x (680er series) under GNU/Linux.

Building Localized Language Packs

As for building localized language packs for different languages, the following steps have to be performed:
$ooo_src> cd instsetoo_native/util/
$ooo_src> dmake ooolanguagepack

Built language packs will be in $ooo_src/instsetoo_native/unxlngi6.pro/OpenOffice_languagepack/deb/install/

10.6 Spell Checker in OpenOffice.Org

History
Spell checker is an important component of the OOo. Initially the available Spell checker in OOo was MySpell. Only recently from OOo 2.0.2, MySpell has been replaced by HunSpell, which is a Hungarian spell checker written by László Németh, because of its better performance and support to non-latin characters.

Character Encoding Issues
The problems of 8 bit encodings covered in this section is based on the documentation of Hunspell [10.8.a].

Myspell uses the 8-bit ASCII character encoding, which is a major deficiency when it comes to scalability. Even with languages like Hungarian which has a standard ASCII character set (ISO 8859-2), MySpell fails to allow a full implementation of Hungarian orthographic conventions. For instance, the '--' symbol (n-dash) is missing from this character set contrary to the fact that it is not only the official symbol to delimit parenthetic clauses in the language, but also can be in compound words as a special 'big' hyphen.

MySpell uses 8-bit encoding tables, but there are languages whose character sets are not covered by the standard 8-bit encoding, too. For example, Nepali and a lot of African languages have non-latin or extended latin characters.

Even in the Hungarian language with its standard ASCII character set (ISO 8859-2), problems arise when dealing with foreign words like *Ångström* or *Molière as* the characters 'Å' and 'è' are not part of ISO 8859-2.

When these words combine with inflections containing characters only in ISO 8859-2 (like elative *-bo=l*, allative *-to=l* or delative *-ro=l* with double acute), they result in words (like *Ångströmro=l* or *Molière-to=l.*) which can not be encoded using any single ASCII encoding scheme.

These problems related to 8-bit ASCII encoding have long been figured out by proponents of Unicode. Unfortunately, switching to Unicode (e.g., UTF-16 encoding) will require a great deal of code optimization and also have an impact on the efficiency of the algorithm. The Dömölki algorithm used in checking affixing conditions utilizes 256-byte character arrays, which would grow upto 64k with Unicode encoding. Since online affixing for a richly agglutinative language can easily have several hundreds of such arrays (in the case of the standard Hungarian resources , this number is ca. 300 or more since redundant storage of structurally identical affix patterns improves efficiency), switching to Unicode would incur high resource costs. Nonetheless, it is clear that trading efficiency for encoding-independence has its advantages when it comes to a truly multi-lingual application.

In this regard, recently a memory and time efficient Unicode handling has been implemented. In non-UTF-8 character encodings, Hunspell works with the original 8-bit algorithms, but this time with UTF-8 encoded dictionary and affix file. Hunspell uses a hybrid string manipulation and conditional checking to support Unicode which are as listed below:

I) Affixes and words are stored in UTF-8.
While analyzing they are handled mostly in UTF-8, and for conditional checking and suggestion they are converted to UTF-16.

II) Dömölki-algorithm is used for storing and checking 7-bit ASCII (ISO 646) conditional characters, and sorted UTF-16 lists of other Unicode characters with conditional patterns.

III) Hunspell supports only the first 65,536 character (Basic Multilingual Plane) of the Unicode Standard till date.

Different Spell Checker Formats in OpenOffice.Org
Kevin Hendriks started MySpell, written in C++, to integrate various open source spelling checkers into the OpenOffice.org build. For more information on MySpell please refer to [10.8.g].

As MySpell has been replaced by Hunspell from OOo 2.0.2, we will not go in details about MySpell.
Hunspell is a spell checker and morphological analyzer library. It has the ability to handle languages with rich morphology and complex word compounding or character encoding. Hunspell interfaces: Ispell-like terminal interface using Curses library, Ispell pipe interface, OpenOffice.org UNO module.

Hunspell's code base comes from the OOo MySpell. The main features of Hunspell spell checker and morphological analyzer are as listed below [10.8.a]:

1. It has Unicode support (first 65535 Unicode character)
2. Morphological analysis can be done (in custom item and arrangement style)
3. Max. 65,535 affix classes and twofold affix stripping (for agglutinative languages, like Azeri, Basque, Estonian, Finnish, Hungarian, Nepali, Turkish, etc.)
4. It supports complex compounding (for example, Hungarian and German)
5. It supports language specific algorithms (for example, handling Azeri and Turkish dotted i, or German sharp s)
6. It can handle conditional affixes, circumfixes, fogemorphemes, forbidden words, pseudoroots and homonyms.
7. It has been released under GPL/LGPL/MPL tri-license

Hunspell consists of language files for different language specific territory. It requires two files in order to define the language that it is spell checking. The first file is a dictionary containing words for the language, and the second is an "affix" file that defines the meaning of special flags in the dictionary. Every locale (language for a specific territory) can have files for HunSpell. These files are located together in one folder, ~openofficefolder/share/dic/ooo/. The spell checking is done using the .aff file for the language together with the .dic file. The .dic file is a list of words along with a group of characters which refer to the affixes found in the .aff file. This saves space because instead of including all forms of a word, for example , drink (drinking, drinks, drunk), the .dic file will include the word once and the references to the affixes in the .aff file allow the

construction of all the other forms. It is not enough to copy the files for a language into the folder. As there could be a many numbers of languages, there would be a considerable overhead if the dictionaries for every language are automatically loaded. Hence, only those languages listed in the dictionary.lst file are loaded. The file dictionary.lst can be edited with a simple text editor [10.8.e].

The dictionary.lst has the following format as shown below:

```
#dictionary.lst for OOo

DICT ne NP ne_NP
THES ne NP th_ne_NP
HYPH ne NP hyph_ne_NP

#end of the dictionary.lst
```

The '#' indicates the commented lines in the file. Each entry in the list has the following space delimited fields
Field 1: Entry Type "DICT" - spellchecking dictionary
"HYPH" - hyphenation dictionary
"THES" - thesaurus files

Field 2: Language code from Locale "en" or "de" or "ne" ...
Field 3: Country Code from Locale "US" or "GB" or "NP"
Field 4: Root name of file(s) for the corresponding field 1 "ne_NP" or "hyph_en_US" or "th_ne_NP (do not add extensions to the name like .dic,.aff etc.)

For Hunspell the line starting from DICT is the only line required. The other two lines are required for the hyphenation and thesaurus.

Writing the Dictionary and affix files for HunSpell
All of the examples in presented in this section have been based on the documentation of Hunspell [10.8.a] for better understanding.

Hunspell needs two files as mentioned earlier to check the spelling. A dictionary file (*.dic) contains a list of words, one per line. The first line of the dictionary (except personal dictionaries) contains the approximate word count (for optimal hash memory size). Each word may optionally be followed by a slash ("/") and one or more flags, which represent affixes or special attributes. Default flag format is a single (usually alphabetic) character. A possible dictionary file, dict1.dic, could be as shown below:

```
3
hello
try/B
work/AB [NOUN]
```

In the above dictionary file, the first line has the number "3", which gives the optimal hash memory size and the number of words, which are "hello", "try", and "work". The first word hello does not have any flag. The word "try" has one flag which is separated by a "/" back slash. The flag "B" points to a rule named "B" in the affix file. Similarly in the next line, the word "work" and it's field AB are separated by a "/". The flag "AB" points to the rule name "A" and rule name "B" in the affix file.

Hunspell also has an optional morphological description field. There is a similar optional field in the dictionary file, separated by tabulator:
word/flags morphology
We define a simple resource with morphological informations.
Dictionary file:

```
drink/X   [VERB]
```

The "[VERB]" informs us that the word "drink" is a verb.

The affix file consists of the rules that will add affixes to the words which are present in the .dic file. As mentioned earlier, the flag, which is usually a character, in the dictionary file points to these rules. To clarify the concept of the affix file consider the following example of an affix file,affix1.aff:

```
SET UTF-8
TRY esianrtolcdugmphbyfvkwzESIANRTOLCDUGMPHBYFVKWZ'

REP 2
REP f ph
REP ph f

PFX A Y 1
PFX A 0 re .

SFX B Y 2
SFX B 0 ed [^y]
SFX B y ied y
```

An affix is either a prefix or a suffix attached to root words to make other words. For example supply -> supplied by dropping the "y" and adding an "ied" (the suffix), or work->rework by simply adding "re".

An affix file (*.aff) may contain a lot of optional attributes. For example, the first line consisting of **SET** specifies the character set used for both the dictionary file and the affix file (should be all uppercase). The above affix file example defines UTF-8 character encoding. **TRY**, in the second line, sets the change characters for suggestions. It is used in building suggestions for misspelled words, for example, the misspelled word "agg" will have a suggestion "egg" by substituting the "a" with an "e". The characters in the string should be listed in order or according to the character frequency (highest to lowest). The suggestions produced using the 'TRY' option differs from the bad word with a single English letter or an apostrophe. A good way to develop this string is to sort a simple character count of a word list. **REP** sets a replacement table for multiple character corrections in suggestion mode. The first line that consists of REP informs that there are two entries for the REP option. With these REP definitions, Hunspell can suggest the right word form, when the misspelled word contains f instead of ph and vice versa, for example, if we write 'fase' it can suggest the right word 'phase'. **PFX** and **SFX** defines prefix and suffix classes named with affix flags.

The affix file is space delimited and case sensitive. So we can interpret the affix file's rule lines , as follows:

```
SFX D Y 4
SFX D   0    d        e
SFX D   y    ied      [^aeiou]y
SFX D   0    ed       [^ey]
SFX D   0    ed       [aeiou]y
```

In the first line there are four fields, whose description are given in the table below:

Field	***Name***	***Description***
1	SFX	indicates that this is a suffix(PFX indicates a prefix)
2	D	this is a name for the suffix it represents which should be unique for every different suffix or prefix entry(or the name for the prefix when PFX is present)
3	Y	indicates it can be combined with prefixes(cross product)
4	4	indicates that sequence of 4 affix entries are needed to properly store the affix information

Table 3. Affix file information

The remaining lines describe the unique information for the 4 affix entries that make up this affix. All the fields in the remaining line are the same, fields in the second line are described below:

Field	*Name*	*Description*
1	SFX	indicates that this is a suffix(PFX indicates a prefix)
2	D	this is a name for the suffix it represents which should be unique for every different suffix or prefix entry
3	y	the string of chars to strip off at the end before adding affix (a 0 here indicates the NULL string, and in case of PFX, the chars are stripped off at the beginning of the word)
4	ied	the string of affix characters to be added at the end of the word(a 0 here indicates the NULL string,and in case of PFX, the chars are added at the beginning of the word)
5	[^aeiou]y	the conditions which must be met before the affix can be applied, which represents a regular expression("." a dot means there is no condition)

Table 4. Affix file information contd..

Field 5 might be confusing. Since this is a suffix, field 5 tells us that there are 2 conditions that must be met. The first condition is that the character next to the last character in the word must *NOT* be any of the following "a", "e", "i", "o" or "u". The second condition is that the last character of the word must end in "y".

If we have a dictionary file (*.dic) as shown below:

1
supply/D
marry/D
and the rule given above in our affix file, then Hunspell will be able to show that the words supply, marry, supplied and married are all correct, by dropping the "y" and adding "ied".

To make more combinations with the dictionary file and affix file we could look at the dictionary file dic1.dic and affix file affix1.aff given above. With the two files we could figure out that Hunspell would detect hello, try, tried, work, worked, rework and reworked as correct words.

The above examples show a single stripping of affixes. Hunspell allows us to strip another one. The twofold suffix stripping is a significant improvement in handling of immense number of suffixes, that characterize agglutinative languages. Consider the example given below:

Affix file:
SFX Y Y 1
SFX Y 0 s . +PLUR

SFX X Y 1
SFX X 0 able/Y . +ABLE
Dictionary file:
drink/X [VERB]
Correct Words:
drink
drinkable
drinkables
In the dictionary file, the word "drink" has the flag X indicating the rule named X in the affix file will be applied to it resulting into "drinkable". In the rule named X in the affix file, you will notice a "/Y", this indicates that after the application of the rule X, go to the rule named Y and further apply the rule resulting into drinkables.

In the affix file, optional morphological description fields could also be present, separated by tabulator as in the dictionary file:

Rule morphology

Below is an example of a simple affix file with morphological informations:

Affix file:

```
SFX X Y 1
SFX X 0 able . +ABLE
```

The morphology can indicate that the rule named "X" is for adding the affix "able" to the words.

Till now, we have had a general overview of the affix and dictionary files and their working mechanisms. Using only the above format will work just fine for Hunspell to check for misspelled words, but there are many other options in Hunspell that will help make a better spell checker for languages, especially if the language is complicated and if the above format is insufficient to define the language. Below are other options to define such a language. The options and their respective explanations are based on the documentation of HunSpell [10.8.a]. Choose options that would define your language efficiently.

GENERAL OPTIONS
The heading of each option is the syntax in which it should be kept, in the affix file.

1. SET encoding
   This option allows you to set the character encoding of words and morphemes in both the affix and dictionary files. Possible values for the encoding are: UTF-8,ISO8859-1,ISO8859-10, ISO8859-15,KOI8-R,KOI8-U,microsoft-cp1251,ISCII-DEVANAGARI.

2. FLAG value
   This option allows you to set the flag type. The flag type is the type of the name of the rule. In the above examples we wrote single characters like "D", "A","B" etc. For English language, the number of affix rules are less so the characters from A to Z will cover the number of rules. Other languages like Nepali may have more affix rules which will not be covered by these single characters. The 'num' value sets the decimal number flag type. Decimal flags numbered from 1 to 65535, and in flag fields in the dictionary file (*.dic), they are separated by commas.

   FLAG num command sets numerical flags separated by comma as shown in the example below:
   FLAG num

   ```
   SFX 65000 Y 1
   SFX 65000 0 s 1
   ```

   Dictionary example:

   ```
   foo/65000,12,2756
   ```

   There is yet another syntax for giving flags in the affix and dictionary files.

   FLAG long command sets 2-character flags:

   ```
   FLAG long
   SFX Y1 Y 1
   SFX Y1 0 s 1
   ```

   Dictionary record with the Y1, Z3, F? flags:

foo/Y1Z3F?

The default type is the extended ASCII(8-bit) character. 'UTF-8' parameter sets UTF-8 encoded Unicode character flags. The 'long' value sets the double extended ASCII character flag type. However, the UTF-8 flag type doesn't work on ARM platform.

3. COMPLEX PREFIXES

This option sets the twofold prefix stripping (but single suffix stripping) for agglutinative languages with right to left writing system.

4. LANG langcode

The language code can be set through this option. In Hunspell, language specific codes can be enabled by LANG code. At present there are az_AZ, hu_HU, TR_tr specific codes in Hunspell (for further information please have a look in the source code).

5. AF number _of_flag_vector_aliases
AF flag_vector

Hunspell can substitute affix flag sets with a natural number in affix rules(alias compression). Let us take an example to make it clear, with an alias compression:
let us take the dictionary (*.dic) file which has the following lines:

```
3
hello
try/1
work/2        [NOUN]
```

Then the AF definitions in the affix file will be

```
SET UTF-8
TRY esianrtolcdugmphbyfvkwzESIANRTOLCDUGMPHBYFVKWZ'

AF 2
AF A
AF AB
```

Now, comparing the dictionary file with the dict1.dic file given previously, we can see that the dictionary file has been replaced by numbers 1 and 2 rather than placing B and AB. This also shows that the AB can be compressed to a single number 2. This could be quite helpful, to compress the aff and dic files. If the affix file contains the FLAG parameter, one needs to define it before the AF definitions.

6. AM number_of_morphological_description_aliases
AM morphological_description

Hunspell can also substitute morphological descriptions with a natural number in the affix rules (alias compression) through this option.

Options for suggestion

1. TRY characters

Hunspell can suggest right word forms, when the typed one differs from the bad form by one TRY character. The parameter of TRY is case sensitive. This option has already been discussed in the beginning of the chapter.

2. NOSUGGEST flag

Words that have been signed with NOSUGGEST flag are not suggested. This flag may come in handy for vulgar and obscene words. The flag should be an English character.

3. MAXNGRAMSUGS num

This option sets the number of n-gram suggestions. Value 0 switches off the n-gram suggestions.

4. NOSPLITSUGS

This option disables the split-word suggestions. For instance, if the word “into” is not in the dictionary and we type “into”, then hunspell will give “in to” in the suggestion list. To prevent such type of suggestions, this option is used.

5. SUGSWITHDOTS
With this option Hunspell adds dot(s) to suggestions, if the input word terminates in dot(s). (Not for OpenOffice.org dictionaries, because OpenOffice.org has an automatic dot expansion mechanism.)

6. REP number_of_replacement_definitions
REP the replacement

We can define language-dependent phonetic information in the affix file (.aff) by a replacement table. First REP is the header of this table and one or more REP data line follows it. With this table, Hunspell can suggest the right forms for the typical faults of spelling when the incorrect form differs by more than 1 letter with the right form. For example a possible English replacement table definition to handle misspelled consonants is shown below:

```
REP 4
REP f ph
REP ph f
REP k ch
REP ch k
```

for the misspelled word “phan”, Hunspell can give a suggestion “fan” in the suggestion table.

NOTE: It is very useful to define replacements for the most typical one-character mistakes too, with REP you can add higher priority to a subset of the TRY suggestions (suggestion list begins with the REP suggestions) and the replacement table can be used for a stricter compound word checking (forbidding generated compound words, if they are also simple words with typical faults, see the option CHECKCOMPOUNDREP).

7. MAP number_of_map_definitions
MAP string_of_related_characters

We can define language dependent information on characters that should be considered related (i.e. nearer than other characters not in the set) in the affix file (.aff) by a character map table. With this table, Hunspell can suggest the right forms for words, which incorrectly choose the wrong letter from a related set more than once in a word. For example a possible mapping could be for the German umlauted ü versus the regular u; the word Frühstück really should be written with umaluted u's and not regular ones

```
MAP 1

MAP uü
```

Options for compounding

1. BREAK number_of_break_definitions
   BREAK character_or_character_sequence

This option defines break points for breaking words and checking word parts separately. It is useful for compounding with joining character or strings (for example, hyphen in English and German or hyphen and n-dash in Hungarian). Dashes are often bad break points for tokenization, because compounds with dashes may contain invalid parts, too.) With BREAK, Hunspell can check both sides of these compounds, breaking the words at dashes and n-dashes:

```
BREAK 2
BREAK -
BREAK --   #n-dash
```

with these rules the breaking becomes recursive, so foo-bar, bar-foo and foo-foo--bar-bar would be valid compounds.

Note: COMPOUNDRULE is better (or will be better) for handling dashes and other compound joining characters or character strings. BREAK, should be used if one wants to check words with dashes or other joining characters. COMPOUNDRULE has handled only the last suffixation of the compound word till date.

2. COMPOUNDRULE number_of_compound_definitions
COMPOUNDRULE compound_pattern
This optioin enables the user to define custom compound patterns with a regex-like syntax. The first COMPOUNDRULE is a header with the number of the following COMPOUNDRULE definitions. Compound patterns consist ofcompound flags and star or question mark meta characters. A flag followed by a '*' matches a word sequence of 0 or more matches of words signed with this compound flag. A flag followed by a '?' matches a word sequence of 0 or 1 matches of a word signed with this compound flag.

Note: '*' and '?' meta characters work only with the default 8-bit character and the UTF-8 FLAG types. COMPOUNDRULE flags haven't been compatible with the COMPOUNDFLAG, COMPOUNDBEGIN, etc. compound flags yet (use these flags on different words).

3. COMPOUNDMIN num
This defines the minimum length of words in compound words. The default value is 3 letters. see Example of compounding

4. COMPOUNDFLAG flag
Words signed with COMPOUNDFLAG may be in compound words (except when word shorter than COMPOUNDMIN). Affixes with COMPOUNDFLAG also permits compounding of affixed words. For example:

#affix file
COMPOUNDFLAG X

dic file:

2
foo/X
bar/X

With this resource, foobar and barfoo also are accepted words.
This has been improved upon with the introduction of direction-sensitive compounding, i.e., lexical features can specify separately whether a base can occur as leftmost or rightmost constituent in compounds. This, however, is still insufficient to handle the intricate patterns of compounding, not to mention idiosyncratic (and language specific) norms of hyphenation.

5. COMPOUNDBEGIN flag
This option makes the flagged words (or with a signed affix) be the first elements in compound words. see Example for compounding

6. COMPOUNDEND flag
This option makes the flagged words (or with a signed affix) the last elements in compound words. see Example for compounding

7. COMPOUNDMIDDLE flag
This option makes the tagged words (or with a signed affix) the middle elements in compound words. see Example for compounding

8. ONLYINCOMPOUND flag
Suffixes signed with ONLYINCOMPOUND flag may be only inside compounds (Fuge-elements in German, fogemorphemes in Swedish). ONLYINCOMPOUND flag also works with words. see Example for compounding

9. COMPOUNDPERMITFLAG flag
This option allows the prefixes at the beginning of compounds, suffixes are allowed at the end of compounds by default. Affixes with COMPOUNDPERMITFLAG may be inside compounds. see Example for compounding

10. COMPOUNDFORBIDFLAG flag
Suffixes with this flag forbid compounding of the affixed word.

11. COMPOUNDROOT flag
COMPOUNDROOT flag signs the compounds in the dictionary (Now it is used only in the Hungarian language specific code).

12. COMPOUNDWORDMAX number
This option sets the maximum word count in a compound word. (Default is unlimited).

13. CHECKCOMPOUNDDUP
This option forbids word duplication in compounds (eg. foofoo).

14. CHECKCOMPOUNDREP
This option forbids compounding, if the (usually bad) compound word may be a non compound word with a REP fault. This is useful for languages with 'compound friendly' orthography.

15. CHECKCOMPOUNDCASE
This option will forbid upper case characters at word bound in compounds. see Example for compounding.

16. CHECKCOMPOUNDTRIPLE
This option forbids compounding, if the compound word contains triple letters (eg. foo|ox or xo|oof). There is a bug in this option,i.e. the missing multi-bytecharacter support in UTF-8 encoding (works only for 7-bit ASCII characters).

17. CHECKCOMPOUNDPATTERN number_of_checkcompoundpattern_definitions
CHECKCOMPOUNDPATTERN endchars beginchars

18. This option forbids compounding, if first word in compound ends with enchars, and next word begins with beginchars.

19. COMPOUNDSYLLABLE max_syllable vowels
This option is needed for special compounding rules in Hungarian. First parameter is the maximum syllable number, that may be in a compound, if words in compounds are more than COMPOUNDWORDMAX. Second parameter is the list of vowels (for calculating syllables).

20. SYLLABLENUM flags
This option is usually required for special compounding rules in Hungarian.

Example for compounding

Example Affix file:

```
# This example is for German compounding

# set language to handle special casing of German sharp s

LANG de_DE

# compound flags

COMPOUNDBEGIN U
COMPOUNDMIDDLE V
COMPOUNDEND W

# Prefixes are allowed at the beginning of compounds,
# suffixes are allowed at the end of compounds by default:
# (prefix)?(root)+(affix)?
# Affixes with COMPOUNDPERMITFLAG may be inside of compounds.

COMPOUNDPERMITFLAG P

# for German fogemorphemes (Fuge-element)
# Hint: ONLYINCOMPOUND is not required everywhere, but the
# checking will be a little faster with it.

ONLYINCOMPOUND X

# forbid uppercase characters at compound word bounds

CHECKCOMPOUNDCASE

# for handling Fuge-elements with dashes (Arbeits-)
# dash will be a special word

COMPOUNDMIN 1
WORDCHARS -

# compound settings and fogemorpheme for ‘Arbeit’

SFX A Y 3
SFX A 0 s/UPX .
SFX A 0 s/VPDX .
SFX A 0 0/WXD .

SFX B Y 2
SFX B 0 0/UPX .
SFX B 0 0/VWXDP .

# a suffix for ‘Computer’

SFX C Y 1
SFX C 0 n/WD .

# for forbid exceptions (*Arbeitsnehmer)

FORBIDDENWORD Z

# dash prefix for compounds with dash (Arbeits-Computer)

PFX - Y 1
PFX - 0 -/P .
```

```
# decapitalizing prefix
# circumfix for positioning in compounds

PFX D Y 29
PFX D A a/PX A
PFX D Ä ä/PX Ä
 .
 .
PFX D Y y/PX Y
PFX D Z z/PX Z
```

Example dictionary:

```
4
Arbeit/A-
Computer/BC-
-/W
Arbeitsnehmer/Z
```

Accepted compound compound words with the previous resource:

```
Computer
Computern
Arbeit
Arbeits-
Computerarbeit
Computerarbeits-
Arbeitscomputer
Arbeitscomputern
Computerarbeitscomputer
Computerarbeitscomputern
Arbeitscomputerarbeit
Computerarbeits-Computer
Computerarbeits-Computern
```

Not accepted compoundings:

```
computer
arbeit
Arbeits
arbeits
ComputerArbeit
ComputerArbeits
Arbeitcomputer
ArbeitsComputer
Computerarbeitcomputer
ComputerArbeitcomputer
ComputerArbeitscomputer
Arbeitscomputerarbeits
Computerarbeits-computer
Arbeitsnehmer
```

Other options

1. CIRCUMFIX flag

Conditional affixes implemented by a continuation class are not enough for circumfixes, because a circumfix is one affix in morphology. We also need CIRCUMFIX option for correct morphological analysis.

```
# circumfixes: ~ obligate prefix/suffix combinations
# superlative in Hungarian: leg- (prefix) AND -bb (suffix)
# nagy, nagyobb, legnagyobb, legeslegnagyobb
# (great, greater, greatest, most greatest)

CIRCUMFIX X

PFX A Y 1
PFX A 0 leg/X .

PFX B Y 1
PFX B 0 legesleg/X .

SFX C Y 3
SFX C 0 obb . +COMPARATIVE
SFX C 0 obb/AX . +SUPERLATIVE
SFX C 0 obb/BX . +SUPERSUPERLATIVE
```

Dictionary:

```
1
nagy/C [MN]
```

Correct Words:

```
nagy
nagyobb
legnagyobb
legeslegnagyobb
```

2. FORBIDDENWORD flag

This flag assigns the forbidden word form. Because affixed forms are also forbidden, we can subtract a subset from set of the accepted affixed and compound words.

3. KEEPCASE flag

This option forbids uppercased and capitalized forms of the words signed with KEEPCASE flags. Useful for special ortographies (measurements and currency often keep their case in uppercased texts) and writing systems (eg. keeping lower case of IPA characters).

Note: With CHECKSHARPS declaration, words with sharp s and KEEPCASE flag may be capitalised and uppercased, but uppercased forms of these words may not contain sharp s, only SS. See germancompounding Example in the tests directory of the Hunspell distribution.

4. LEMMA_PRESENT flag

Generally, there are dictionary words as lemmas in output of morphological analysis. Sometimes dictionary words are not lemmas, but affixed (not real) stems and virtual stems. In this case lemmas (real stems) need to put into morphological description, and forbid not real lemmas in morphological analysis

adding LEMMA_PRESENT flag to dictionary words.

5. NEEDAFFIX flag
This flag signs virtual stems in the dictionary. Only affixed forms of these words will be accepted by Hunspell. Except, if the dictionary word has a homonym or a zero affix. NEEDAFFIX works also with prefixes and prefix + suffix combinations.

6. WORDCHARS characters
WORDCHARS extends the tokenizer of Hunspell command line interface with an additional word character. For example, dot, dash, n-dash, numbers, percent sign are word characters in Hungarian. see Example for compounding

7. CHECKSHARPS
SS letter pair in uppercased (in German) words may be upper case sharp s (ß). Hunspell can handle this special casing with the CHECKSHARPS declaration (see also KEEPCASE flag) in both spelling and suggestion.

10.7 Thesaurus in OpenOffice.Org

History

MyThes is a thesaurus made by Kevin Hendricks. This thesaurus has the facility to provide a words meaning and synonym but not it's antonym, which a thesaurus should be able to provide. MyThes was made specially to provide OOo with a thesaurus. It is the first thesaurus for OOo and is still being used with some enhancements from the OOo community. Originally, it did not support UTF-8 encoding, which was a big setback for countries lacking their own 8-bit ASCII character set. Recently, László Németh, the creator of Hunspell, provided a patch for MyThes to support unicode. This patch could be patched to MyThes if versions of OOo older than 2.0.2 were present. In versions OOo 2.0.2 and above, the patch has been automatically integrated into OOo. The creation of this patch has been a milestone in the internationalization (I18n) of MyThes, because non-latin languages now can be integrated into the thesaurus of OOo.

Thesaurus implementation in Ooo

MyThes is a very simple thesaurus. This thesaurus does not only provide information on synonyms, but also meanings and theparts-of-speech of a word. The main features of MyThes are listed below:

1. 1.It is written in C++ to make it easier to interface with Pspell, OpenOffice, AbiWord, etc.
2. It is stateless, as no static variables are used and should be completely reentrant with no ifdefs .
3. It compiles with -ansi, -pedantic, and -Wall with no warnings, making it quite portable.
4. It uses a simple perl program to read the structured text file and generate the index file which contains the index needed for binary searching.
5. It is very simple with “lots” of comments. The main "smarts" are in the structure of the text file that makes up the thesaurus data.
6. It comes with a ready-to-go structured thesaurus data file for en_US extracted from the WordNet-2.0 data.
7. The source code has a BSD license (and no advertising clause).

Making the Data files for MyThes

The thesaurus depends entirely upon the data file which should be provided. The program is very simple, which implements binary search. The format of the data file is simple. The example that is presented in this section is taken from the data_layout file of the MyThes thesaurus [10.8.d]. A top portion of the data file which includes one word entry for “ simple” is shown below:

```
ISO8859-1
simple|9
(adj)|simple|elemental|ultimate|oversimplified|simplistic|simplex|simplified|unanalyzable|undecomposable|un
complicated|unsophisticated|easy|plain|unsubdivided
(adj)|elementary|uncomplicated|unproblematic|easy
(adj)|bare|mere|plain
(adj)|childlike|wide-eyed|dewy-eyed|naive |naif
(adj)|dim-witted|half-witted|simple-minded|retarded
```

(adj)|simple |unsubdivided|unlobed|smooth
(adj)|plain
(noun)|herb|herbaceous plant
(noun)|simpleton|person|individual|someone|somebody|mortal|human|soul

The general format for the data file is assimple as shown above. In this file, there should be no binary data and lines should end in a '\n' and not in a carriage return or a line feed. The first line of the file data file is the encoding that the file uses. Encodings that are recognized by MyThes are listed as below:

ISO8859-1
ISO8859-2
ISO8859-3
ISO8859-4
ISO8859-5
ISO8859-6
ISO8859-7
ISO8859-8
ISO8859-9
ISO8859-10
KOI8-R
CP-1251
ISO8859-14
ISCII-DEVANAGARI
UTF-8 The remaining structure of the data file will be as shown below:

entry|num_mean
pos|syn1_mean|syn2|...
.
.
.
pos|mean_syn1|syn2|...

The fields in the remaing part are delimited by a pipe '|', and can be described as below:

For the first line:

FIELD	DESCRIPTION
entry	All lowercase versions of the word that is being described for the synonym
num_mean	The number of different meanings of the entry. Indirectly it also indicates the remaining number of lines.

For the second line:

FIELD	DESCRIPTION
pos	The Parts-of-Speech or other meaning specific descriptions of the word being described
syn1_mean	The meaning of the word,which could be a synonym
syn2	The synonym of the word having the same meaning

The fields for the remaining lines of a word is the same as in the second line. This type of entry has to be made for each word which should be in the thesaurus. The data file that was given above as an example can be interpreted as “the file is encoded in ISO8859-1, and consists of the word “simple” which has 9 different meanings and each meaning having it's part of speech and at least one synonym, if present, following on the same line. MyThes software, by default has files for English, which could be looked into as a reference.

MyThes uses the data file to store the words along with it's synonym,meaning, and part of speech. Along with MyThes, a perl program “th_gen_idx.pl” is shipped. This program is used to generate a file that will help index the words in the data file for MyThes. The proper way to run the program is by executing the following

command in the terminal.

#cat th_en_US_new.dat | ./th_gen_idx.pl > th_en_US_new.idx

In the above command “th_en_US_new.dat” is the name of the data file that the user creates. The file name that the program th_gen_idx.pl generates is “th_en_US_new.idx”. The name of the .dat file and the .idx file must be same. This command should be executed from the terminal, and the directory from where the .pl program is executed should be where both the .dat file and the .pl file exists.

The .idx file that is generated will look as shown below:
ISO8859-1
1
simple|10

The first line indicates the encoding of the file. The second indicates that there is only one word entry, and the next line indicates that the word “simple” in the .dat file is after 10 bytes from the beginning of the file. After this information MyThes will start to read the information on the word from the location mentioned in bytes.

The complicated portion is over if both the .dat and .idx file have been generated. The only part remaining is letting the OOo know where the files for MyThes is present. This is done by making an entry into the dictionary.lst file which is present in the Ooo_directory/share/dict/ooo/ folder, as mentioned in the SpellChecker section of this guide and placing both the .dat and .idx file in the same directory where the dictionary.lst is placed.

10.8 References for Further Reading

a) http://sourceforge.net/docman/display_doc.php?docid=29374&group_id=143754
b) http://lingucomponent.openoffice.org/
c) READ ME file of MyThes : http://lingucomponent.openoffice.org/MyThes-1.zip
d) data_layout.txt file of MyThes :http://lingucomponent.openoffice.org/MyThes-1.zip
e) http://en.wikipedia.org/wiki/Myspell
f) http://about.openoffice.org/
g) http://www.openoffice.org/license.html
h) http://www.it46.se/localegen/
i) http://www.khmeros.info/tools/localization_of_openoffice_2.0.html
j) http://www.khmeros.info/tools/how_to_patch.html
k) http://www.khmeros.info/tools/openoffice_locale_.htm
l) http://www.khmeros.info/tools/Collation_in_ooo_2.0.html
m) http://www.khmeros.info/tools/oo2.0_program_translaltion.html
n) http://ne.openoffice.org/stats/
o) http://translate.sourceforge.net/
p) http://www.khmeros.info/tools/translate.html
q) http://ftp.linux.cz/pub/localization/OpenOffice.org/devel/build/build
r) http://tools.openoffice.org/dev_docs/build_linux.html
s) http://wiki.services.openoffice.org/wiki/CPAN_install

11 Linux Distribution Development for Localization

11.1 Introduction

In this Chapter, we deal with the Live CD based Linux Distribution. We have primarily focused on Debian GNU/Linux. Linux Distributions and Localization have also been dealt with. Next, we also discuss the development of a Live CD Linux Distribution. References for further reading are provided at the end of the chapter.

About Linux

Linux is an operating system initially created by Linus Torvalds in the early 90s. Since then thousands of developers worldwide have contributed to its development and have also been supported by major corporations as: IBM, Novell , HP etc. Linux is the name of the core kernel which has all the functionality of an operating system. The kernel, at the heart of all Linux systems, is developed and released under the GNU and its source code is freely available to everyone. Linux is also popularly known as Gnu/Linux. It is one of the most prominent examples of free software and open source development. It is this kernel that forms the base around which a Linux operating system is developed and we call this the Linux Distribution. The distribution consists of several GNU applications and tools. Some of the most popular Linux distributions are RedHat, Debian, SuSe etc. Since Linux and some of the distributions are released under GNU and are open source, we can modify and redistribute them. A wide range of applications for Linux are developed and used today in various sectors. The main reasons for the popularity of Linux applications are it's low cost, flexibility and stability.

11.2 Linux Distribution

Linux distribution, is a Unix-like system comprising of software components such as the GNU/Linux kernel and assorted free, open source, and possibly proprietary software. There are currently over three hundred Linux distribution projects in active development, their respective distributions being revised and improved. A Linux Distribution can be derived from another Linux distribution by the necessary customizations to the original ones. Some of the most popular Linux Distributions are:

- Debian GNU/Linux
- Red Hat Linux
- Fedora Core
- SuSe Linux
- Knoppix

Distributions are developed and supported by communities or commercial companies. Debian GNU/Linux for example is supported by the community. It is up to the user, which one to use, either commercial or the other. Distributions are developed according to their usage . For example : desktops, servers, routers, multimedia, clusters etc. So, one may choose the distribution according to his/her need.

Distributions are normally segmented into packages, each package holding a specific application or service. This is achieved with the aid of the package management system. A package management system is a collection of tools to automate the process of installing, upgrading, configuring, and removing software packages from a computer. Some of the popular package management tools are: rpm , dpkg etc.

Few of the different Linux distributions available today are

- Hard Disk based distributions
- Live CD Linux distributions
- Embedded Linux distributions
- Floppy based Linux distributions
- Diskless terminal Linux distributions
- Handheld/PDA Linux distributions
- Flash Disk based Linux distributions

Hard Disk based Linux Distributions

This type of Linux distribution is the most common type Distribution. The method of installing Linux is by

booting from a CD that contains the installation program and installable software. The CD can be burned from a downloaded ISO image, purchased alone for a low price, or can be obtained as part of a box that may also include manuals and additional commercial software. The most popular Red Hat Linux falls into this category.

Live CD Linux distributions
Live CD based Linux distribution has the operating system kernel, scripts and software stored on a bootable CD or DVD that can be run directly from the CD or DVD drive, without installing into permanent memory, such as a hard drive. It runs with the help of Ramdisk. A Ramdisk is a virtual solid state disk that uses a segment of active computer memory. Ramdisk provides a role typically functioned by hard drives. A live CD does not alter the current operating system or files without a user's intervention. The system returns to its previous OS state when the live CD is ejected and the computer is rebooted. Morphix and Knoppix are the most popular Live CD Linux Distributions.

Debian GNU/Linux
Debian is a widely used distribution of free software developed by the joint efforts of volunteers from around the world. It consists of a lot of basic tools of the operating system from the GNU project and supports the common computer architectures like: x86, Power PC etc. Debian GNU/Linux is the basis for several other distributions, including Knoppix, Ubuntu Linux, linspire, etc. Debian is also supported by donations made available by the Software in the Public Interest, a non-profit umbrella organization for free software projects.
Debian is also well known for its package management system, especially APT, the Advanced Packaging Tool. APT simplifies the process of installing and removing software on Debian systems, by automating the retrieval, the configuration, the compiling (sometimes) and the installation of software from APT sources.

Lots of Debian derivatives exist today. Debian Derivatives are subsets of Debian which are configured to support a particular target group out-of-the-box. For example: Debian aimed for science, Debian aimed for schools etc. In our case, NepaLinux, which is also a Debian Based Linux Distribution, is targeted specially towards Desktop users in the Nepali Language. Popular Debian Derivatives are:

- DebianGIS: a CDD for Geographical Information and Earth Observation Systems
- Debian Junior: For children
- Debian Med: For Medical
- DebianNeo: Debian Stable for newbies
- Skolelinux (built by the DebianEdu project): aimed for schools.
- Knoppix
- Mepis
- Linspire
- Gnoppix
- Morphix

11.3 Linux Distributions and Localization

The idea behind the development of the Linux distribution by the localizers is to install all the localized and related software in one CD or DVD. For this kind of Linux distributions, most preferred type of Linux distribution is a Live CD or a Live DVD, which can run directly without installing in the Hard disks. This will help the community with a single source of all the localized applications and users do not have to deal with the complexity of the manual installation of localized applications and tools. Some of the applications that a typical Live CD or Live DVD Linux distribution includes:

- Unicode Support
- Language and Culture Locales
- Different types of Input Methods
- Open type and true type fonts
- Localized Desktop
- Localized Icons and themes
- Localized Office suite and Internet browsers
- Spell Checkers
- Thesaurus
- Many more applications

The following section will guide you in developing a Localized Live CD Linux distribution based on the popular Debian GNU/Linux and Morphix Live CD Linux distribution.

11.4 Development of a Live CD Linux Distribution

This section will cover the following topics regarding the development of the Live CD Linux Distribution.

- Setting up the Build Environment
- Creating the Main Module consisting of Localized Desktop, Input Methods etc
- Base Module Modification and Customization, adding Language option on the boot menu, using customized Images, tweaking etc.
- Generating the ISO to burn on CD

Morphix and Debian Gnu/Linux for the Live CD

Morphix is a modular Linux distribution, based on Knoppix. Morphix uses the Live CD features of Knoppix, but it is more modular. Different modules can be combined on a CD for varying purposes, making Morphix a sort of Live CD construction kit. For example, a Morphix CD can contain a normal base system and use the Localized Gnome Desktop as a module. Other modules can be constructed for other specific purposes, such as a firewall, a rescue disk, a basic office suite etc. A base module can be considered as a core image of the Live CD and a main module is a compressed image containing the software applications. In this section, we will look at the steps required for creating a Morphix and Debian GNU/Linux for the Live CD.

For this, we need to download the Morphix base module and create a Main Module consisting of Gnome Desktop using Debian Repositories. A Debian repository is either anftp or httpd source which contains software packages.

There are different methods of creating a Main Module for the Live CD. One popular method is to use the Debian repositories. Basically there are 5 Debian Repositories.

1. **Stable**
   This repository is the latest official release of the Debian GNU/Linux distribution. This is stable and well tested software, which changes only if major security or usability fixes are incorporated. Presently, Sarge is a codename for the Debian stable distribution.

2. **Testing**
   Testing repository contains packages that are intended to become part of the next stable distribution and are less stable as compared to the packages from the stable repository. Testing does not get the timely security updates from the security team. Currently, Etch is the codename for the testing distribution.

3. **Unstable**
   Unstable area contains the most recent packages in Debian. Once a package has met our criterion for stability and quality of packaging, it will be included in testing. "Unstable" is also not supported by the security team. Sid is the current codename of the unstable repository.

4. **Woody**
   Woody is considered as the obsolete Debian stable release. It was the stable version from July 19th to June 6th 2005, when Sarge took over.

5. **Experimental**
   The experimental repository is the part of Debian where the most bleeding-edge and the unstable software is being soaked before it will be robust and tested at least to be able to get into the unstable repository.

Setting up the Build Environment

Build environment is the system in which we are actually undertaking all the distribution development. Following are the detailed steps to setup a build environment.

Downloading Debian GNU/Linux

Different flavors of Debian like: sarge, etch or sid CD/DVD images can be downloaded from the website

http://www.debian.org/CD/. It depends on the user which one to use, either CD image, DVD images or other available options. The easiest route for most people will be to use a set of Debian CDs. So, in this document we will be considering the Debian Sarge Distribution that can be downloaded from the url http://cdimage.debian.org/debian-cd/current/i386/iso-cd/ . There are altogether 14 CDROM iso files which can be burnt to the blank CDs using GnomeBaker , cdrecord or Nero CD burning utility. For setting up the build environment, download first 2 iso named debian-testing-i386-binary-1.iso and debian-testing-i386-binary-2.iso. The number of iso, version, and codename can subjected to change later. The name and version used in this guide is the one released at the time of writing this guide. You can get more information regarding releases on the site http://www.debian.org/releases/ .

Installing Debian Sarge
Basic Steps for installing Debian Sarge from the CD images are described below.

Making the system bootable from CD
In order to install Debian in your computer, you need to boot the computer with the Sarge first CD. Follow the steps below to make the system bootable from the CD.

a) As the computer starts, press the keys to enter the BIOS utility. Often, it is the **Delete** key. However, consult the hardware documentation for the exact keystrokes.
b) Find the boot sequence in the setup utility. It's location depends on your BIOS, but you are looking for a field that lists the drives. Common entries on IDE machines are C, A, cdrom or A, C, cdrom. C is the hard drive, and A is the floppy drive.
c) Change the boot sequence setting so that the CD-ROM is first. Usually, the Page Up or Page Down keys cycle through the possible choices.
d) Save your changes. Instructions on the screen tell you how to save the changes on your computer.

Booting Debian Sarge from the first CD and Continuing the Installation

- Turn on your computer and insert the Debian Sarge first CD in your CDROM Drive.
- Type linux26 and press Enter to Boot. Using linux26 option will install the 2.6.x series kernel into the system and simply pressing Enter will install 2.4.x series Kernel. Here we will use linux26 option.
- Choose
    - Language = English
    - Country or Region = United States
    - Keymap = American English
- If the “Configure the Network” screen appears, Press Continue and select “Do not configure the network at this time” . We are installing Debian just from CDs, so it's OK to proceed without configuring the network. Note that if your computer is connected to the network and your network is DHCP enabled network, the screen will not appear.
- Next, type any name like “Buildmachine” when prompted for the Hostname
- The Partition disks screen appears now. First you will be given the opportunity to automatically partition either an entire drive, or free space on a drive. This is also called “guided” partitioning. If you do not want to autopartition, choose Manually edit partition table from the menu. If you choose guided partitioning, you will be able to choose from the following schemes as shown in the table below.

Partitioning scheme	***Minimum space***	***Created partitions***
All files in one partition	600MB	/, swap
Desktop machine	500MB	/, /home, swap
Multi-user workstation	1GB	/, /home, /usr, /var, /tmp, swap

Table 5. Schemes for guided partioning

If you select Manually edit the partition table, you need to create at least a Linux ext3 partition and a Linux swap Partition. To make a new partition,

a) Select the FREE SPACE entry and press Enter.

b) Choose Create a New Partition.
c) Use any size you may wish . But since we are installing the system for the development of the Live CD, more disk space will be more comfortable to work with . Hence we choose 15.0 GB.
d) Next choose “Primary” following with the “Beginning” option. Note that if you have other operating system installed already, you may have to choose Logical partition also. Reading more on Disks and partitions is highly recommended.
e) By default, the Debian installer will choose the ext3 file system for the created partition and automatically selects the mount point as /
f) Linux ext3 partition is created and as per the requirement Linux swap has also to be created. So, Select the FREE SPACE entry and press Enter once again.
g) Choose Create a New Partition. Linux swap partition size must be double the size of the physical RAM of the computer. For example: I have 256 MB of RAM, so i decided the Linux Swap partition to be 512 MB. Hence, provide the size which best suits your case.
h) Repeat step 'd'.
i) Under Partition settings, Press Enter on the menu “Use as”. Select “swap area” and press Enter.
j) Now run “Done Setting up Partition”.
k) Press Enter on “Finish partitioning and write changes to disk” menu.
l) Finally, Click“Yes” to “Write the changes to disk?”.

- Base Installation process starts from this point and after the installer will prompt you whether to Install the Grub boot loader in the master boot record. Choose “Yes”
- After writing Grub to the master boot record, the installer will eject the CD . Then press Continue. This will restart the machine and boot into the new installed system. But note that this is not the end of the installation and only half way to the completion.
- Now, the Welcome Screen appears . Choose “Ok”.
- Choose

a) Hardware clock set to GMT = yes.
b) Time Zone = Your Zone.
c) root password = any strong password.
d) re-type password = password you typed just a step before.
e) Full name for the new user = Firstname Lastname.
f) Username for the new user = Firstname or any other preferred.
g) Password for the new user= any strong password.
h) re-type password for the new user= password you used for the new user.
i) Use a PPP connection to install the system = No

- After following the above steps , the Apt configuration screen appears. APT is the mechanism Debian uses to manage and install software packages. We are installing from CDs so insert the CD 1 and select the “cdrom” method and press Ok on the next Screen. This will scan the inserted CD and put information of packages of the CD.Afterwards, apt will ask you to Scan another CD. Just opt yes and all the CDs that you have will be scanned. Press OK on the next screen.
- Then it will ask you to insert the CD 1. Insert the CD 1 and press Enter. The necessary packages will be installed in this phase.
- “Debian Software Selection” screen will appear now. Press Space bar to select/deselect your choice. Here we will choose the “Desktop Environment” option for our purpose and then press OK. Selecting this option will install a Debian Sarge system with a Gnome Desktop Environment which will be more than enough for our Live CD development task. If the installer prompts the questions, answer them which best suits your system or use the default options. In most of the cases default option should work except while answering the choice for the mouse configuration, you need to use PS2 if ps2 mouse is connected to your PC .
- The installation process will take some time at this stage depending on the configuration of you PC. While installation, the process may ask you the required CDs , so insert the right CD if prompted.
- After the completion of the packages installation, few configurations will be remaining for finishing the entire process.
- Choose, General Type of Mail configuration: no configuration at this time.
- Choose Yes and the installer will ask where to have administrative email sent. This will normally be your own user account or email address.
- Finally Gnome Login Prompt will appear . Use the cerated user name and password to login to the

gnome desktop.

This completes the Debian Sarge Installation and Now you can log in to your new Debian system.

Network Configuration
Build machine needs numerous packages to be download for the development of the Live CD. Hence we will configure our build machine to access the Internet with the following steps:

Steps:
1. Login to the system using the account created at the install time.
2. Go to Applications --> System Tools --> Networking.
3. It will ask the Administrator Password. This is the root password that was created at the install time. So, use the root password.
4. Click Forward.
5. Check Ethernet or any other possible media.
6. Select Manual or Automatic.
7. If you select Manual, insert IP address, netmask and Gateway address. Consult your network administrator for the information.
8. Click Forward.
9. Check Apply and active connection.
10. Click Forward and then Apply button.
11. Next, Click the DNS tab and type the DNS server IP address of hostname. Consult your network administrator for the DNS information.
12. Click Add and Press OK.

Upgrading the System
The installed Debian System can be upgraded to the latest version by using the Debian online repositories. This is not mandatory but recommended. Following are the basic steps required to upgrade the system.

a) Add the following lines to the /etc/apt/sources.list file using your favorite text editor like vi, gedit etc
   deb http://ftp.debian.org/debian/ sarge main contrib non-free
   deb http://security.debian.org/ sarge/updates main contrib
b) Run: apt-get update
c) Now upgrade your system using: apt-get dist-upgrade. In this process, the apt tool will download all the necessary packages and may ask question while configuring those. You can use the default ones or make the choice that suits your system
d) Restart the system.

Installation of the required tools for building and customization of the Live CD
Tools required to a build Live CD:

debootstrap:
debootstrap bootstraps a basic Debian system into the target folder from the debian mirror. After debootstrap, the downloaded debian system will only contain the basic minimal packages. Install debootstrap using the following command.
apt-get install debootstrap

cloop tools:
Compressed loopback device or cloop is mostly used as a convenient way to compress conventional file systems onto LiveCDs. Cloop is a module for the Linux Kernel. The module image that is used on the Morphix , Knoppix based or some other Live CDs uses cloop system for the compression. A compression ratio of about 2.5:1 is common for software. While building the module for the LiveCD cloop tools must be installed. Use the steps to install cloop tools.
a) apt-get install cloop-utils
b) apt-get install cloop-src

morphing tools:
A number of command line tools and scripts are provided in order to create the modules for the Live CD. Basic steps to install morphing tools is

a) Add "deb http://www.morphix.org/debian ./" to your /etc/apt/source.list
b) apt-get update
c) apt-get install morphing-tools

Note that if the installation of morphing-tools failed due to dependencies you can only install morphix-modulebuilder . Use the following command to install it.

apt-get install morphix-modulebuilder

Downloading the Base ISO
A base module can be referred as a heart of the Live CD. When we download the base ISOs , we can find the base module by mounting the downloaded ISO. Two flavours of Base ISOs can be downloaded.

Downloading the Base ISO from the morphix website downloads; URL: http://www.morphix.org/
Downloading from the Daily autobuilds; URL: http://www.morphix.org/autobuilds/base/

The base ISO that can be downloaded from the morphix website is considered more stable than the one which can be downloaded from the autobuilds. For those, who need the latest kernel with the latest packages are recommended to use the ISO from the autobuilds. Otherwise, use the ISO from the morphix download section.

Extracting the Base ISO
The Base ISO can be mounted to some folder and all the contents can be copied to a folder. The following steps need to be followed.

a) mkdir /iso
b) mount -o loop basemod-2.6.15-2006-05-17_0015.iso /iso
c) Where basemod-2.6.15-2006-05-17_0015.iso is the downloaded file from morphix autobuilds
d) mkdir /mylivecd
e) cd /iso
f) cp -Rp * /mylivecd
g) umount /iso

Main Module Construction
The Main Module is a compressed image consisting of software packages like: Gnome Desktop, Office applications etc. In this section, we will create a main module for our Live CD which is considered as a major task.

1. **First, debootstrap the base system to some folder**
   mkdir /mainmodule
   debootstrap sarge /mainmodule
   Instead of sarge you can use etch or sid, whichever you like while debootstraping. Since sarge is very stable, it is the one that is highly recommended.

2. **Chroot to the folder mainmodule.**
   Chroot is an operation which changes the root directory. Root directory is the first or top-most directory in a hierarchy.
   chroot /mainmodule

3. **Make entry for the apt sources.**
   Here we will use the Debian sarge repositories
   Add the following lines to the /etc/apt/sources.list file using your favorite text editor like vi, gedit etc

   deb http://ftp.debian.org/debian/ sarge main contrib non-free
   deb http://security.debian.org/ sarge/updates main contrib

4. **Run: apt-get update**

5. **Now install the packages of your choice.**

Here we install xfree86 or xorg X server, x window system (X libraries, a set of fonts, group of basic X clients and utilities,etc), and gnome desktop. xfree86 is an implementation of the x window system. It is a free and open source software under the XFree86 License version 1.1. It is developed by the XFree86 Project, Inc. The XOrg Server is the official reference implementation of the x window system . It is a free and open source software released under MIT License. X11R6.7.0. The first version of the X.Org Server was forked from xfree86 4.4 RC2. The immediate reason for the fork was a disagreement with the new license for XFree86 4.4 final, as several disagreements among the contributors surfaced prior to the split. When the fork was created changes were folded in from X11R6.6 creating a common codebase. Many of the previous XFree86 developers have joined the X.Org Server project. Debian sarge repositories have xfree86 packages. Hence we consider xfree86 in this document.

Run: apt-get install xserver-xfree86 or xserver-xorg
Opt Y
Run: apt-get install x-window-system

Note that after you install x windows by using the above commands,you need to make sure that the folder /tmp has sticky bit on and looks like drwxrwxrwt . If not, you can use the following commands to change the permission

chown 1777 /tmp -R

iv) Install Gnome Desktop using: apt-get install gnome-desktop-environment
After the installation of the packages, the system will ask some questions for configuration for which you can use the default options

6. Installing other additional packages

apt-get install xscreensaver
apt-get install pcmcia-cs

7. Adding up Locales, Fonts, Translations and Input Methods

This section guides you how to install locales, fonts , translated files and different types of input methods for our localized distribution.

1) Locales

i) Install the locale package using:

apt-get install locales

ii) Remove the previous (old) one if exist :

rm -rf /usr/lib/locale/xx_XX , where xx=Language code & XX=Country code

iii) Copy the UTF-8.gz file to /tmp

cp /usr/share/i18n/charmaps/UTF-8.gz /tmp

iv) gunzip the UTF-8.gz file

/bin/gunzip -d /tmp/UTF-8.gz

v) Copy the locale file to /tmp:

cp xx_XX /tmp
Note that if your language locale is already listed at glibc, you can find it under /usr/lib/locale/

vi) Run:

/usr/bin/localedef -i /tmp/xx_XX -f /tmp/UTF-8 /usr/lib/locale/xx_XX

vi) Check using:

locale -a

vii) Removing the files:

1. rm -rf /tmp/UTF-8
2. rm -rf /tmp/xx_XX

2) Fonts

i) Copy your open type unicode fonts

```
cp fontname.ttf /usr/share/fonts/truetype
```

ii) Run:

```
fc-cache -f
```

iii) Verify using:

```
fc-list
```

Note that the font folder, where you copy the fonts should be reachable by the X server. This behavior can be modified using the files inside /etc/fonts and /etc/X11/XF86Config-4 or /etc/X11/Xorg.conf

3) Translations

Adding translated files is dependent on the type of packages that are localized. Here we will cover adding translations for the software packages that are localized using the gettext internationalization library. Note that the package version and the package from which the translated file is extracted should be the same.

i) Copy the files

```
cp *.mo /usr/share/locale/xx/LC_MESSAGES
```

4) Input Methods

This section will cover how to use xkb or scim input methods.

i) XKB

Copy your developed xkb file

1. cp -f xx /usr/X11R6/lib/X11/xkb/symbols/pc
2. cp -f files/xx /usr/X11R6/lib/X11/xkb/symbols

Add the following lines to /usr/X11R6/lib/X11/xkb/rules/xorg.xml or /usr/X11R6/lib/X11/xkb/rules/xfree86.xml file. While adding the following lines, check for already present layout sections. You can insert your section somewhere following the alphabetical order of the countries.

```
<layout>
<configItem>
<name>xx</name>
<description>XX</description>
</configItem>
<variantList/>
</layout>
```

If your xkb file is already submitted to the official xkb main stream there is no need perform the above steps.

ii) SCIM

Install scim pacakges

- apt-get install scim
- apt-get install scim-gtk2-immodule
- apt-get install scim-tables-xx OR apt-get install scim-tables-additional

(Use this command only if the scim keyboard layout mapping table is submitted to the official scim main stream. If not submitted, you can copy your mapping binary table file to the location /usr/share/scim/tables/)

iii) IIMF

Install iiimf packages

- apt-get install iiimf-htt-csconv
- apt-get install iiimf-htt-xbe iiimgcf libiiimcf2 libiiimp0 iiimf-htt-server

If the iiimf keyboard mapping table is not submitted to the official iiimf main stream , use the following steps

- Create a folder with your country name
  mkdir -p /usr/lib/im/locale/UNIT/LANUAGE/data
- Copy the binary file
  cp yourlayout.data /usr/lib/im/locale/UNIT/COUNTRY/data
- Add the following to /usr/lib/im/locale/UNIT/sysime.cfg

  [xx_XX]
  yourlayout common/ctim.so LANGUAGE

5) Adding the necessary scripts

i) Create a locale.gen file under /etc and add your ane en_US locale entry.

Example of locale.gen file. Here ne_NP is the locale for Language Nepali and Country Nepal

```
----------------------
en_US ISO-8859-1
en_US UTF-8
ne_NP UTF-8
----------------------
```

ii) mkdir /morphix cdrom cdrom MorphixCD floppy

iii) Create a file inside /morphix named init.sh. Copy paste the following lines to it.

```
---------------------------
#!/bin/sh
#
# Module-dependent initscript
#
# copyleft 2003, Alex de Landgraaf
# GPL, (www.gnu.org for details)

# This is a placeholder adapted for the HeavyGUI mainmodule
# Place actions to be started on initialisation
# of your MainModule in here

USER=$USERNAME
WINDOWMANAGER=gnome-session
INSTALLED=no
XSERVER=XFree86
PATH="/bin:/sbin:/usr/bin:/usr/sbin:/usr/X11R6/bin:/usr/local/bin:."

# load the config's generated by our basemodule
XMODULE=""
[ -f /etc/sysconfig/xserver ] && . /etc/sysconfig/xserver
[ -f /etc/sysconfig/morphix-all ] && . /etc/sysconfig/morphix-all

echo "MainModule loaded"

[ -f /etc/sysconfig/xserver ] && . /etc/sysconfig/xserver
[ -f /etc/sysconfig/morphix-all ] && . /etc/sysconfig/morphix-all
[ -f /etc/sysconfig/keyboard ] && . /etc/sysconfig/keyboard
[ -f /etc/sysconfig/i18n ] && . /etc/sysconfig/i18n
```

```
export LANG COUNTRY CHARSET

[ -f /etc/sysconfig/keyboard ] && . /etc/sysconfig/keyboard
# Set default keyboard before interactive setup
[ -n "$KEYTABLE" ] && loadkeys -q $KEYTABLE
[ -n "$CONSOLEFONT" ] && consolechars -f $CONSOLEFONT

# Try to find and load a drm module for this graphics card
# ripped from Knopper's xsession
if [ -n "$XMODULE" ]; then
for i in /lib/modules/*/kernel/drivers/char/drm/*; do
case "$i" in *$XMODULE*) modprobe $XMODULE;; esac
done
fi

stringinfile(){
case "$(cat $2)" in *$1*) return 0;; esac
return 1
}

# Check if there isn't a background in /tmp (for minimodule)
# Actually, this should be done for each initscript too,
# as it would make them adaptable for minimodules...

#if [ -e /tmp/background.png ]; then
#BGIMAGE=/tmp/background.png
#else
#BGIMAGE=/morphix/background.png
#fi

/etc/init.d/cupsys start &
#mount -avF -o ro -t nonfs,nosmbfs,noncpfs,noproc

if [ $INSTALLED = no ]; then
# Setting up our handy-dandy console-shells, thanks to popular demand...
# ripped from Knopper's knoppix-autoconfig
   while true; do /bin/bash >/dev/tty2 2>&1 </dev/tty2; done &
   while true; do /bin/bash >/dev/tty3 2>&1 </dev/tty3; done &
   while true; do /bin/bash >/dev/tty4 2>&1 </dev/tty4; done &
   while true; do /bin/bash >/dev/tty5 2>&1 </dev/tty5; done &
   while true; do /bin/bash >/dev/tty6 2>&1 </dev/tty6; done &

   echo "allowed_users=anybody" >> /etc/X11/Xwrapper.config
   echo "nice_value=-10" >> /etc/X11/Xwrapper.config

#    echo "xsetbg -fullscreen $BGIMAGE &" >> /etc/X11/xinit/xinitrc
#    echo "xsetbg -fullscreen $BGIMAGE &" >> /home/$USER/.xinitrc
#    echo "xsetbg -fullscreen $BGIMAGE &" >> /home/$USER/.xsession

echo "exec $WINDOWMANAGER" >> /etc/X11/xinit/xinitrc
echo "exec $WINDOWMANAGER" >> /home/$USER/.xinitrc
echo "exec $WINDOWMANAGER" >> /home/$USER/.xsession

su -c"exec /usr/bin/X11/startx" - $USER

# wait until X is locked
```

```
for i in 1 2 3 4 5 6 7 8 9 10
do
if [ -f /tmp/.X0-lock ]; then
        break
        sleep 1
        fi
done

   for i in $XSERVER; do
        killall -TERM $i 2> /dev/null && echo "X-Server shut down." && break
   done
fi

echo "Rebooting... (disabled, just to be sure, for now)"
echo "Use reboot or halt to shutdown your computer"
halt
#exec reboot
----------------------------------
```

5. Generate the main module

   i) Run: apt-get clean

   Before creating the main module, if you want to save space on the CD , you can remove the deb files that are used to install the pacakges. This will remove the deb files of the folder /var/cache/apt/archives

   ii) Exit the chroot environment
   Type: exit

   iii) Execute:
   module-builder /mainmodule /mylivecd//mainmod/mymainmodule.mod

   Base Module and Customization

   Base module customization and modification is useful if you want to add your language on the Live CD boot menu, changing images , adding modified icons etc.

   1. Extract the base module
      i) mkdir /basemodule
      ii) extract_compressed_fs /mylivecd/base/morphix > /tmp/morphix.iso
      iii) mount -o loop /tmp/morphix.iso /basemodule
      iv) cd /iso
      v) cp -Rp * /basemodule
      vi) umount /iso

   2. Adding Language for the Live CD

      i) Change lang=us to lang=xx in every grub entry of the file /mylivecd/boot/grub/menu.lst. By default, this will boot Live CD to your language locale

      ii) Adding Language to the Supported Language menu list . Add the following lines to /mylivecd/boot//grub/lang.lst

      ```
      title Morphix | YourLanguage

        kernel (cd)/boot/vmlinuz ramdisk_size=100000 noapic acpi=off apm=power-off
      vga=791 splash=silent initrd=miniroot.gz quiet BOOT_IMAGE=morphix lang=xx
      ```

initrd (cd)/boot/miniroot.gz

iii) Change lang=us to lang=xx in every grub entry of the file /mylivecd/boot/grub/options.lst. This will boot the Live CD to your language even if you use submenu options to boot the CD

Note: You can also change the labels (eg: Morphix) to your own (eg: NepaLinux) by editing the files under /mylivecd/boot/grub/

iv) Add the following lines to the file /basemodule/etc/init.d/knoppix-locales at proper locations

---

```
COUNTRY="XX" ( where XX = Country code in small letters )
LANG="xx_XX"
 KEYTABLE="us"
 XKEYBOARD="us"
 KDEKEYBOARD="dev"
 CHARSET="UTF-8"

        # Additional KDE Keyboards
 KDEKEYBOARDS="us,dev"
 TZ="Continet/City"
 ;;
```

---

3. Grub Boot Image modification

Grub boot Image here refers to the first background image which is in blue and white image displayed when we boot from the CD. You can modify the existing image to a new image. Gimp is highly recommended tool for this modification . Find the steps below to modify the image.

i) Create a directory to work:

mkdir /message

ii) cd /message

iii) The image resides in the file /mylivecd/boot/grub/message. Since it is cpio archive we use the following command to extract it:

cpio -i < /mylivecd/boot/grub/message

iv) Modify the background.pcx

v) Create the message file from the /message folder

a) cd /message

b) ls . | cpio -o > /mylivecd/boot/grub/message

While modification. the specification of background.pcx image should be exactly as below

a) A pcx format file
b) 640x503 Resolution
c) Size not exceeding 35KB
d) 14 Color Image

vi) Boot Splash Images Customization

BootSplash Images of the Live CD refers to the images that are displayed while the Live CD boots. The current base module uses 6 images. So, create your own 6 images and replace the existing ones. Find the steps for the modification.

- gunzip /mylivecd/boot/miniroot.gz
- mount -o loop /mylivecd/boot/miniroot /iso

- Bootsplash images resides under /iso/bootsplash/images. Replace them with the new ones.
- umount /iso
- gzip /mylivecd/boot/miniroot

File name	*Size (not exceeding)*	*Resolution*
bootsplash-1024x768.jpg	*21KB*	*1024x768*
silent-1024x768.jpg	22KB	*1024x768*
silent2-1024x768.jpg	35KB	*1024x768*
silent3-1024x768.jpg	88KB	*1024x768*
silent4-1024x768.jpg	84KB	*1024x768*
silent5-1024x768.jpg	91KB	*1024x768*

Table 6. Specification of the images to be used

4. Generating the Base Module after Customization

After modification and making the necessary changes you need to create the modified base module. Use the steps below to do the same.

i) mkisofs -R -U -V "morphix" -P "Morphix" -cache-inodes -nobak -pad /basemodule > /tmp/morphix.iso

ii) create_compressed_fs /tmp/morphix.iso 65536 > /mylivecd/base/morphix **Extra Modification and Customization**

a) Themes and Icons

Gnome uses themes to alter the appearance of buttons, scrollbars, list elements, to customize the appearance of windows. You can download themes for Gnome from Websites like http://www.gnome-look.org and extract them to the folder /mainmodule/usr/share/themes or modify the existing ones in the folder /mainmodule/usr/share/themes.

As with themes, gnome also uses gnome icons themes for the desktop icons. Download icons themes (eg: from gnome-look.org) to the folder /mainmodule./usr/share/icons or modify the existing ones. Check for the /mainmodule/usr/share/pixmaps folder for the default icons that are used by the Gnome Desktop.

b) Changing the Distribution name and other texts that are displayed while booting the Live CD and pressing F2

- gunzip /mylivecd/boot/miniroot.gz
- mount -o loop /mylivecd/boot/miniroot /iso
- Check the file /iso/linuxrc and do the necessary changes. For example: To change the distribution name change DERIVATIVE =yourdistrbutionname
- umount /iso
- gzip /mylivecd/boot/miniroot

c) Changing Default username

The Live CD boots to GUI using the username morph. To change the username, follow the steps below:

- Edit the file /basemodule/etc/init.d/morphix-start and /basemodule/etc/init.d/knoppix-autoconfig of the base module. In this file change username=morph to username=newusername.

- Change morph to newusername in file /basemodule/etc/passwd
- Change the user in the file passwd and group file of miniroot.gz

  i) gunzip /mylivecd/boot/miniroot.gz
  ii) mount -o loop /mylivecd/boot/miniroot /iso
  iii) Change user morph to newusername in the files /iso/etc/passwd and /iso/etc/group
  iv) umount /iso
  v) gzip /mylivecd/boot/miniroot

Refer to the section base module customization and modification to extract and generate the base module.

d) Setting password for the newusername

After changing the default username to newusername, the password should be set to the newusername. Steps to set the password are;

- Copy the binary mkpasswd to your base module
  cp mkpasswd /basemodule/usr/bin/
- Replace
  chroot /mnt/main useradd -m $USERNAME -s /bin/bash
  chroot /mnt/main useradd -s /bin/bash $USERNAME

  with the following lines

  chroot /mnt/main useradd -m $USERNAME -s /bin/bash -p `/usr/bin/mkpasswd $USERNAME`
  chroot /mnt/main useradd -s /bin/bash $USERNAME -p `/usr/bin/mkpasswd $USERNAME`

  in the file /basemodule/etc/init.d/morphix-start. This will set the password same as the username. Refer to the section base module customization and modification to extract and generate the base module.

e) Set the root password

After the Live CD starts and ready to use, in some cases you need the password of root.

- chroot /mainmodule
- passwd root
- Type the password
- umount /mainmodule

This will set the new password for root. Refer to the section Main Module Construction to extarct and generate the Main Module

f) Hostname modification

Hostname is used to either set or display the current host or domain name of the system. This name is used by many of the networking programs to identify the machine. The default hostname is Morphix. See below to change to a new one.

- Edit the basemodule file /basemodule/etc/init.d/knoppix-autoconfig. Change 'hostname Morphix' to 'hostname newname'
- Replace the existing hostname in the file Base module file /basemodule/etc/hostname with the newname
- Change Morphix to newname in the Base module file /basemodule/etc/hosts

See the section base module customization and modification to extract and generate the base module.

g) Changing the Distribution name while your Live CD is going to be halt or reboot
Check the file /basemodule/etc/init.d/knoppix-halt and replace MORPHIX with your YOURDISTRIBUTIONNAME

Refer to the section base module customization and modification to extract and generate the base module.

h) Using your own Gnome splash screen
While the Gnome Desktop starts , a splash screen is displayed. This image resides in the folder /usr/share/images/desktop-base of the main module. If you want to use your own created splash screen, first create the image in png format , 450x200 in size and copy it in the folder /mainmodule/usr/share/images/desktop-base/ . Then follow the steps listed below.

Let the filename of the newly created image be mysplash.png.

- chroot /mainmodule
- cd /etc/alternatives
- rm desktop-splash
- ln -s /usr/share/images/desktop-base/mysplash.png desktop-splash
- exit

Create the main module by refering to Main Module Construction section.

i) Changing the default Background of the Gnome Desktop
If you want to use your own background for the Gnome Desktop, create an image which is 1600x1200 in size of png file format and copy it to the folder /usr/share/images/desktop-base of the main module. Let's say the name of the background image you created is mybackground.png.Then the following steps listed below should be followed:

- chroot /mainmodule
- cd /etc/alternatives
- rm desktop-background
- ln -s /usr/share/images/desktop-base/mybackground.png desktop-background
- exit

See the Main Module Construction section to create the main module.

Live CD Directory Structure
Before generating a final ISO for your Live CD, the minimal mandatory directories are base, boot and mainmod. If you look inside the folder mylivcd which is generated from the Base ISO, you can see many folders. A general overview of the functionality of those folders is given below.

base
Folder that contains the base module which detects and configures your hardware and contains your standard kernel modules

boot
Contains the init ramdisk, grub menu and kernel

copy
If you place files in this directory, they will be copied over to the root of your filesystem of the Live CD

mainmod
Contains main modules

minimod
Contains mini modules. Mini module is a module that does only one task. For example , you can create a mini module of the package nmap and place in this directory. Doing this, you can use namp in your Live CD without installing it into the main module. Refer to the morphix wiki (http://www.morphix.org/wiki/) to know

more about mini modules and see how to create them.

/**deb**
If .deb (debian packages) are placed in this directory, they will be installed at boot time of the Live CD

/**exec**
If you place files in this directory, they will be executed at the boot time.

Note: You can delete all files from the /mylivecd folder. But it is highly recommended to put the Live CD license file and MD5 checksum of the Live CD in this folder. Use the following command to generate the MD5 check sum of your Live CD after copying main module , base mod and other required files to the /mylivecd folder.

find /mylivecd -type f -exec md5sum {} \; | awk '{print $1"\r"}' | sort | md5sum > /mylivecd/md5sums

Creating the ISO image
An ISO image (.iso) is an informal term for a disk image of an ISO 9660 file system. Hence we create an ISO image of our Live CD Linux distribution to burn it to the CD. Use the command below to generate the iso file.
mkisofs -pad -l -r -J -v -V "Live CD" -b boot/grub/iso9660_stage1_5 -c base/boot.cat -no-emul-boot -boot-load-size 4 -boot-info-table -hide -rr -moved -o /mylivecd.iso /mylivecd

Burning the ISO
There are many tools to burn the ISO image to the Live CD. Command line tool 'cdrecord' and GUI based tools like gnomebaker, K3B are recommended tools.

Using cdrecord to burn the ISO

- Check your CD Writer's information
  cdrecord -scanbus -dev=ATAPI
  Here note the scsibus in which you CDRW is connected.

- Supposing 0,0,0 is the scsibus in which your CDRW is connected. Use the following command if you want to erase your CDRW
  cdrecord -dev=ATAPI:0,0,0 blank=fast

- Finally burn the ISO using the following command
  cdrecord -dev=ATAPI:0,0,0 /mylivecd.iso

Testing ISO using QEMU
QEMU is free software written by Fabrice Bellard that implements a fast processor emulator, allowing a user to run one operating system within another one. In our case, we can test our Live CD ISO without burning it to the CD. See below how to use QEMU to run Live CD iso in the buildmachine.

- Open a linux terminal from the Gnome Desktop
- Run:
  qemu -cdrom /mylivecd.iso

11.5 References for Further Reading

a) http://www.morphix.org
b) http://www.debian.org
c) http://www.linuxdevcenter.com
d) Linux Man Pages
e) http://www.kernel.org
f) http://www.gnu.org
g) http://www.freshmeat.net
h) http://www.sourceforge.net
i) Research report on creating a bootable Live-CD. Paras Pradhan, Basanta Shrestha, Subir B. Pradhanang. Madan Puraskar Pustakalaya, Nepal.

12 Development of Internationalized Open Source Applications

12.1 Introduction

In this Chapter, we begin with a brief introduction on Internationalization following which we deal with developing QT based applications and localizing them. We have tried to give detailed information on developing and localizing QT based applications. References to links for further reading are provided at the end of the chapter.

What is Internationalization?

Internationalization is defined as a process of developing a software product whose core design is not based on a particular locale. Hence it should potentially handle all targeted linguistic and cultural variations (such as text orientation, date/time format, currency, accented and double-byte characters, sorting, etc.) within a single code base. Another necessary step to prepare a product for localization is separating all message strings in text files [12.3.b]. The distinction between internationalization and localization is subtle but important. Internationalization is the adaptation of products for potential use virtually everywhere, while localization is the addition of special features for use in a specific locale.

12.2 Developing and localizing QT-based applications

Introduction to QT

Qt Toolkit is a cross-platform graphical widget toolkit for the development of GUI programs. It is extensibly used in the K Desktop Environment. The producer of Qt is the Norwegian company Trolltech, formerly known as Quasar Technologies.

Extended version of the C++ programming language is used by Qt. At the same time, bindings exist for Python, Ruby, C, Perl and Pascal. It runs on all major platforms, and has extensive internationalization support. In addition to this, there are also various non-GUI features like SQL database access, XML parsing, thread management, and a unified cross-platform API for file handling [12.3.c].

QT Open Source download can be found at http://www.trolltech.com/products/qt/downloads.

Open Source Edition download is available for the following platforms :

- Qt/Windows Open Source Edition
- Qt/X11 Open Source Edition
- Qt/Mac Open Source Edition

Why use QT?

Because QT is Comprehensive, Cross-Platform, Easy to Use, Robust & Open Source [12.3.d].

- Qt is a comprehensive development framework that includes an extensive array of features, capabilities and tools that enable development of high-performance, cross-platform rich-client and server-side applications.
- Using Qt delivers true platform independence - code once and deploy anywhere. Targeting a new platform demands little more than a simple recompile of a single source code base.
- Qt developers only have to learn one API to write applications that run almost anywhere.
- Qt has been tested by worldwide commercial and open source application developers over different platforms and compilers - forming the foundation for high-performance, resource-intensive applications.
- Open Source edition is available under the GPL license . Open source benefits include an active open source developer community contributes to the ongoing development of Qt while complete code transparency allows Qt developers to "see under the hood", customizing and extending Qt to meet their unique needs.

Prerequisites for developing localized applications using QT

The following prerequisites must be at hand in order to develop the localized QT application[12.3.d] .

1. **Qt Designer** is a powerful GUI layout and forms builder, enabling rapid development of high-performance user interfaces with native look and feel across all supported platforms.
2. **Qt Linguist** is a set of tools designed to smoothen the internationalization workflow. Using Qt Linguist, development teams can outsource the translation of applications to non-technical translators, increasing accuracy and greatly speeding the localization process.
3. **Input Method** for providing local language Input method to localize the application.

You need to follow the following steps to develop the localized QT application.

1. Start up *Qt Designer* to invoke the *Qt Designer New/Open* dialog .
This dialog contains three tabs once you start up *Qt Designer.*
2. Select C++ project and provide a project name for example : qtproj1.pro in directory 'qtproj1'. C++ projects are saved as .pro files, which include the information *Qt Designer* needs to manage projects.
3. Click File|New (or press Ctrl+N) to invoke the *New File* dialog.
4. Create a “Push Button” in the form with name “pushButton1” and text “Quit”.
5. Create a “Text Label” in the form with name “textLabel1” and text “Hello World”.
The form will look as shown below in fig. 16 :

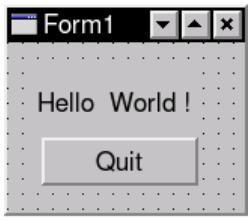

Figure 16. Form in the Qt Designer

6. Click Edit|Connections to invoke the *View and Edit Connections* dialog. Use this dialog to view and edit signal and slot connections. To add a new connection, click the New button. Specify the Sender as 'pushButton1', Signal as 'Clicked', Receiver as 'Form1' and Slot as 'close()' .

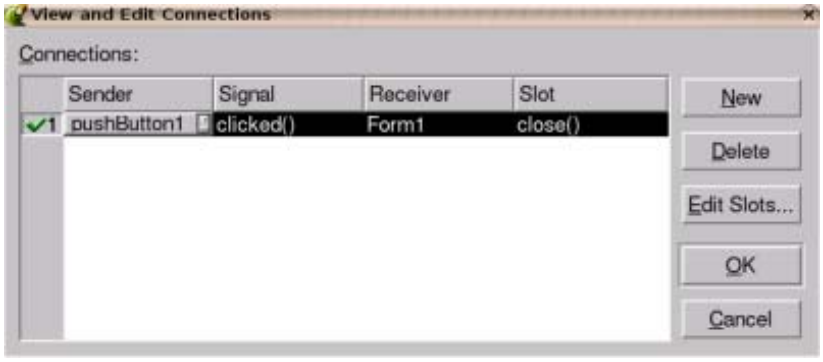

Figure 17. View and Edit Commections

7. Right click the form and select the source. Click yes when it confirm to create 'ui.h' file.

Code Sample for form1.ui.h

```
void Form1::init(){
   pushButton1->setText(QPushButton::tr("Quit"));
  textLabel1->setText(QLabel::tr("Hello  World !"));
}
```

8. Click **File|New|C++ Main-File** to invoke the *Configure Main-File* dialog.

Code sample for main.cpp

```
#include <qapplication.h>
#include form1.h

int main( int argc, char *argv[] ) {

QApplication app( argc, argv );
        //creates a object of Qtranslator class without parent
        QTranslator trans( 0 );
        //Load a file ne_NP.qm
        trans.load( "ne_NP", "." );//loads the translation ne_NP.qm
        //Adds a translation from ne_NP.qm to the pool of          //translation used by the program
        app.installTranslator( &trans );
        QObject::connect( &button, SIGNAL( clicked() ), &app, SLOT( quit() ) );
        app.show(); //display the form
        return app.exec();
}//end main
```

9. Open the 'qtproj1.pro' project file and add the line “TRANSLATIONS = ne_NP.ts” at the end of file.

```
TEMPLATE = app
LANGUAGE = C++

CONFIG += qt warn_on release

SOURCES += main.cpp

FORMS = form1.ui

unix {
  UI_DIR = .ui
  MOC_DIR = .moc
  OBJECTS_DIR = .obj
}
```

TRANSLATIONS = ne_NP.ts

10. Open the terminal and go to the QT Project directory 'qtproj1'. Provide the following command to generate the translation file 'ne_NP.ts'.

```
$lupdate qtproj1.pro
```

***lupdate** reads a Qt .pro project file, finds the translatable strings in the specified source, header and *Qt Designer* interface files, and produces or updates the .ts translation files listed in the project file (.pro file).

The generated ne_NP.ts will look like :

```
<!DOCTYPE TS><TS>
<context>
  <name>Form1</name>
   <message>
      <source>Form1</source>
      <translation type="unfinished"></translation>
   </message>
   <message>
      <source>Hello World</source>
      <translation type="unfinished"></translation>
```

```
  </message>
  <message>
    <source>Quit</source>
    <translation type="unfinished"></translation>
  </message>
</context>
<context>
  <name>QLabel</name>
  <message>
    <source>Quit</source>
    <translation type="unfinished"></translation>
  </message>
</context>
<context>
  <name>QPushButton</name>
  <message>
    <source>Hello World</source>
    <translation type="unfinished"></translation>
  </message>
</context>
</TS>
```

*****<translation type="unfinished"> states that the translation of the string is still incomplete. Between two opening “<translation>” and closing“</translation>” tag, we can put the translation of the given string manually.

11. Open the ne_NP.ts file with QT Linugist by providing following command.

```
$linguist ne_NP.ts
```

***** Here we are opening QT Linguist to load the ne_NP.ts translation file. After translation of each string in the .ts file, mark each string as finished.

12. After translation, the ne_NP.ts file will look like this:

```
<!DOCTYPE TS><TS>
<context>
  <name>Form1</name>
  <message>
    <source>Form1</source>
    <translation> मेरो नेपाली अनुप्रयोग </translation>
  </message>
  <message>
    <source>Hello World</source>
    <translation></translation>
  </message>
  <message>
    <source>Quit</source>
    <translation></translation>
  </message>
</context>
<context>
  <name>QLabel</name>
  <message>
    <source>Quit</source>
    <translation>बन्द गर्नुहोस्</translation>
  </message>
</context>
<context>
```

```
  <name>QPushButton</name>
  <message>
    <source>Hello World</source>
    <translation>नमस्कार संसार</translation>
  </message>
</context>
</TS>
```

13. After adding the equivalent localized string of the English string, ne_NP.qm can be generated by clicking File|Release. An alternate way to generate ne_NP.qm is by giving the following command.

    $lrelease qtproj1.pro

    ***lrelease** reads a Qt .pro project file and produces the .qm files used by the application, one for each .ts translation source file listed in the project file. Translation file (.ts file) is a human readable file but cannot be used to load a translation unless it is compiled to .qm file. The .qm file format is a compact binary format that provides extremely fast lookups for translations.

14. To compile the QT project,enter the following command :

    $qmake
    $make

    ***qmake** reads the .pro file and configures the current project. It generates the Makefile which will later be used by make process.
    ***make** reads .pro and compiles the project using c++ compiler and builds a binary 'qtproj1'.

15. ne_NP.qm will now be used by our application. To execute the application,provide the following command :

    $./qtproj1

    Now the application will load the translated file (ne_NP.qm) and will look like fig.17:

Figure 18. Example of the interface in Nepali

12.3 References for Further Reading

a) http://www.digitalfanatics.org/projects/qt_tutorial/
b) http://doc.trolltech.com/3.3/
c) http://www.remedy.com/customers/dev_community/UserExperience/glossary.htm
d) http://en.wikipedia.org/wiki/Software_localization

13 Building Free Open Source Software (FOSS) Communities

13.1 Introduction

In this Chapter, we talk about building Free Open Source Software (FOSS) communities. We start with a brief introduction on FOSS communities. Later we will discuss the role of an FOSS community, the necessity for building such communities and ways of building them. Finally, we briefly highlight some of the efforts in building FOSS communities in South Asia.

13.2 What is a FOSS community?

A set of people, in general sharing the same idea that Free and Open Source Software(FOSS) should be promoted can be termed as a FOSS Community. A Foss Community may comprise of developers, FOSS advocates, experts, end-users etc. This Community is assumed to be actively involved in interaction on FOSS issues both formally and informally – through emails and internet, formal meetings and discussions, gatherings etc.

13.3 What does a FOSS Community do?

A FOSS Community acts as an information center for answering to the queries and problems on Free Open Source Software. Besides, FOSS Communities can also facilitate the conducting of awareness campaigns and trainings on FOSS. In addition, it can also be a common forum for filing bugs and correspondingly discussing and providing the bug-fixes. Hence, a FOSS Community has a wide sphere of activities for involvement.

13.4 Why build a FOSS Community?

Despite the gradual increase in the popularity of FOSS in the recent years, one of the bottlenecks in the wide usage of the FOSS is lack of adequate technical support. FOSS Developers are basically enthusiasts and volunteer programmers, who spend no time preparing proper documentation on the software they develop. Besides, they also do not see reason to document things they know or regard simple enough to be understood by their partner programmers. This has had a negative impact on the potential users of FOSS. People tend to shy away from using FOSS because they find the proprietary ones richly furnished with the required documentation. Besides, in terms of installation and troubleshooting too, they find plenty of people to turn to, who can easily fix their problems as opposed to FOSS applications. Usage of FOSS applications still remain feasible and accessible to only a limited number of Linux/Unix masterminds. Such a scenario, if not counterchecked by the FOSS supporters,will increase the divide and inaccessibility of the general users to the enormous benefits of FOSS. Building a FOSS Community could be the first initiative to provide a solution to this problem. As the FOSS Community grows up, the technical support required for the deployment of FOSS becomes more feasible and more accessible to the general masses.

13.5 How to build a FOSS Community?

There are various ways of building a FOSS Community. The most common of them being the member of Linux User Groups (LUGs), GNU/Linux User Groups (GLUGs), Free Software User Groups (FSUGs), BSD User Groups (BUGs). In addition to this, these user groups also may be created locally under someone's initiative for some local purpose. Usually, people from the same city or country form a team and thus create a group. These people meet regularly and organize various meets or events like installfests, thus providing the technical help and support to the local people. One way to give continuity to the groups is by maintaining the mailing lists, where people can ask questions, provide answers to them, and make comments or suggestions and so on. Online forums are equally popular for asking questions and getting answers. Internet Relay Chat (IRC) channels are also available for most FOSS Projects where people can consult with each other regarding their respective problems. Wikis are equally an important medium for sharing the knowledge. Other important mediums of active communication and contribution include Concurrent Versioning System (CVS) and SubVersion (SVN) for the management of the source code. The error submitting tool or BugZilla is important tool for filing bugs, patches, issues and so on.

FOSS communities may be created both locally and globally. For instance, in terms of celebrating the Software Freedom Day (SFD) on 16 September, communities could be moderated both in the national and international level. In the local level, if there are organizations working under FOSS Projects, they could take

the initiative for creating a platform of common interaction. Talk programs could be conducted to which more and more organizations, educational institutions and other stakeholders could be involved. This creates a base for the development of a Free and Open Source Community.

13.6 Efforts in building FOSS Communities in South Asia

Although locally there may be several groups and communities formed in the different countries of South Asia, lately there has been some talks as to how individual country FOSS groups and communities could be assembled under one common umbrella. The email group forum bytesforall_floss@yahoo.com is one of such initiatives. It is a network to link Free/Libre and Open Source Software (FLOSS, or FOSS) advocates in South Asia, with an intention of building regional links and specially encouraging localisation efforts in this populous part of the planet which can really benefit from the power of free-as-in-freedom software. This is a sub-group of the wider BytesForAll Network. Currently some of the major FOSS Communities working in the group include BytesForAll[Fredrick Noronha], South Asia, FOSSFP Pakistan, Sarai India, NepaLinux, MPP, Nepal etc.

14 Localization Project Management Techniques, Experiences of Madan Puraskar Pustakalaya under the PAN Localization Project

14.1 Introduction

Project management is one of the crucial aspects of the Localization or any other project, this guide would have remained incomplete without discussing the managerial approaches,software development cycle and general strategies adopted.In this Chapter, we share some of the experiences of the Madan Puraskar Pustakalaya, the Nepal Component of the PAN Localization Project in terms of Localization Project Management(2004-2006).

14.2 Localization Project Management

For the Madan Puraskar Pustakalaya team the task of producing a Nepali Linux Distribution within a time span of two years was a big challenge, primarily because it was being done for the first time and everything had to be done almost from scratch. Achieving something on such a big scale, requires careful planning from the management side. In the following section, we describe the management approaches adopted in due course of the accomplishment of the Project.

Management Approaches

Project management is all about controlling the five variables: cost, time, scope, quality and risk. An assortment of the traditional approaches with flexible project management techniques are what must be followed for planning, execution and delivering outputs for such a project. According to the project deliverables, we divided the localization team in three units namely Linux unit, Translation unit and Natural Language Processing unit. The rest of the document will be devoted to the measures employed for accomplishing the deliverables in each unit.

The management of the project was divided into the following stages:

a) Preparing for the necessary prerequisites of localization;
b) Project Planning;
c) Project Execution;
d) Project Monitoring;
e) Project Completion.

Prerequisites for localization

One of the prerequisites of localization is Standardization as mentioned in the earlier chapters of this guide. A national level policy on standardization needs to be initiated if localization for a particular language has never taken place. In case of the Nepali language, attempts were made to localize smaller applications like date conversion and so on. However fonts and keyboard layout had become the major hurdle in the process. Fonts were all developed in ASCII encoding scheme with devastating. results. Data processing was impossible with such fonts, and more over each font had its own keyboard layout. This resulted in the confusion for the users as they had to learn different styles of typing corresponding to different fonts. Localization and language computing seemed to be a far fetched dream due to these hurdles.

A major change came through when Unicode consortium proposed an encoding scheme for all the languages. After Devanagari, the writing script of the Nepali language, found a place in the scripts supported by Unicode, the Nepali character set was standardized. Later, Nepali Unicode fonts were developed based on this scheme,

Unicode fonts brought a sense of excitement to the users. Now, searching, sorting, even calculation became possible. Keyboard drivers for Unicode font were developed in parallel with font development. Even though Unicode fonts have their own keyboard layouts, users became keen on using Unicode fonts because it incorporated both the traditional Nepali keyboard layout and the new Romanized keyboard layout. The Romanized keyboard driver became popular because as the name suggests, most the Nepali character were mapped to similar phonetic English characters.

With the development of Unicode fonts and keyboard layouts, MPP, along with other software corporate

houses initiated the works on localization working in close collaboration with the National level of committee for standardization, a sub body of the High Level Commission for Information Technology (HLCIT). The latter is headed by Hon. Minister of Nepal and is an apex government body formed with the objective of providing crucial strategies direction and helping to formulate appropriate policy responses for development of ICT sector in Nepal. A Steering committee "Nepali Language in Information Technology" was formed under the HLCIT to deal with the issues on standardization. This committee has already standardized the glossary of 2600 words, Nepali character set, keyboard layout, locale and collation sequence.

Project Planning

The MPP localization team was divided into three different working units respectively, the Linux Unit, Translation Unit and the Natural Language Processing Unit. A detailed overview of the work force division is presented in the figure below.

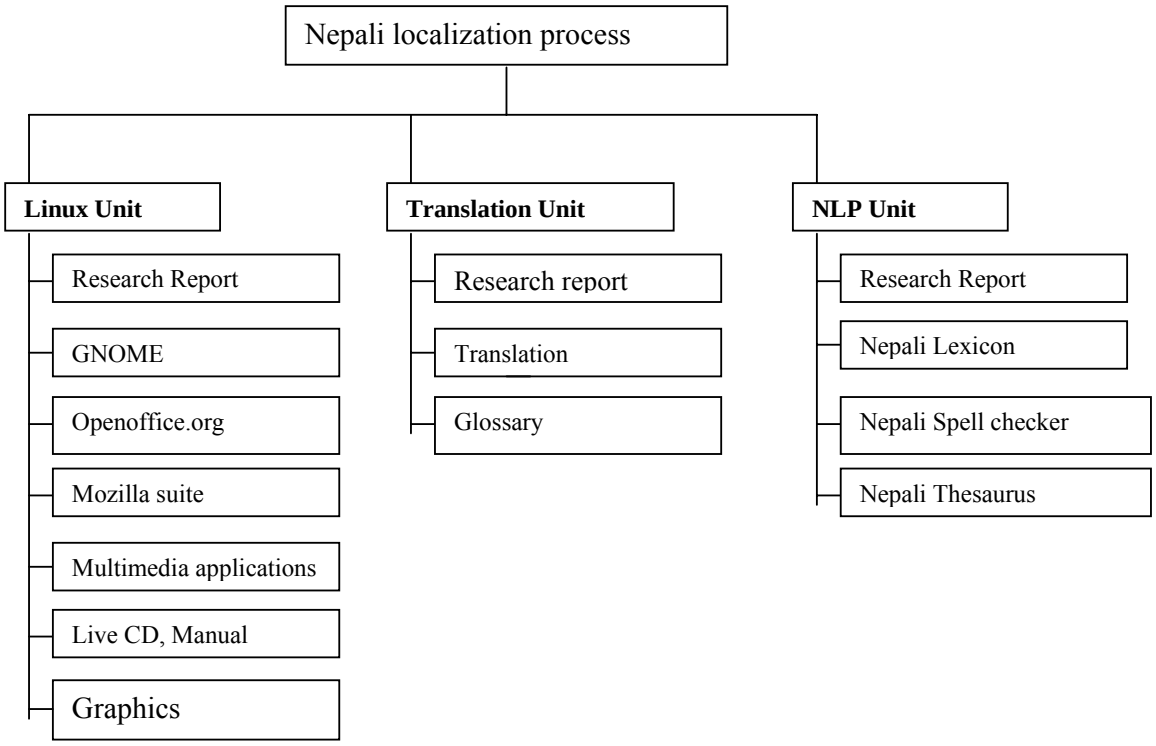

Figure 19. Nepali Localization work force division

Estimating the resources

One of the most important factors that influence the cost and time is the correct estimation of human resources. We cannot, however ignore the fact that with the first attempt at localization, correct estimation can not be done beforehand in the initial phase of the project. Accurate estimation can only be done after execution of all three units for at least a month. Moreover, correct estimation also depends on factors, discussed below.

For the Linux unit, defining the scope of the tasks will greatly help in providing a correct estimation of human resources required. Since the project intention was to produce Nepali Linux distribution for desktop users, we needed to provide the sources to the Desktop Environment (GNOME), office package (openoffice.org), internet (mozilla suite), graphics designing (GIMP) and other applications. Studying, building and testing of these various components requires at least three full time working persons.

For the Translation unit, number of translation strings defines the estimation of the translation man hour. Metrics used for calculating the human resource requirement for translation for the Nepali language were as follows:

Calculate the total strings to be translated = 105000strings

One string on an average contains 10 to 15 words.
Multiplying total strings with 10 will give the approximate total number of words to be translated = 105000*10=1050000 words.

According to our experience, a translator on an average, translates maximum 1060 words per day working seven hours a day.

Therefore one translator working 6 days a week can translate = 1060*26= 27,560 words per month.

Other factors governing the human resource estimation are:

- Level of experience of translators
- Familiarity with the Linux Operation system
- Deciding on the version of GNOME. As newer and stable version of GNOME gets released , on an average 2000 strings are added. So the translated files need to be revisited to accommodate the new strings in the files. Therefore, translation is an ongoing process unless the decision to stick to one particular version is finalized. We had started translation from GNOME 2.6 and we have come up all the way to GNOME 2.14.
- 20% time of translator effort goes on testing and quality checking,
- 10% effort getting used to the terms in the glossary and training with the translation tool

Translation team structure:

- Translation Manger – 1
- Translation Team leader -1
- Translators – 4

For the Natural Language Processing unit:
A full time linguist and a developer would be able to complete the Nepali Spell checker of 24,500 root words and Nepali thesaurus of about 6,000 words in about 6 months. However other issues that could affect are:

1. Lexicon Building
2. Whether an automated tool is available for lexicon building?
3. Whether the framework for the development of the Spell Checker and the Thesaurus for a particular language exist in openoffice?
4. The difficulty level of integrating spell checker and thesaurus for a particular language in Openoffice.org.

Project Execution
Every software corporate house implements this stage with some variations, however, the typical combination would be defining the problems, evaluating the alternatives, choosing a path, and then implementing it. Performing these project activities, requires a decision to be made about the framework for software developmental process. Localization, being different from conventional software development, does not necessarily follow the conventional method of software development. A bit of tweaking with the model may be required. Localization process requires iterative building and testing from the early phase. For this reason, spiral model seems to best fit the practice. Hence, the spiral life cycle model for software development was adopted. The spiral life cycle models for the translation and the Linux and NLP units are shown in figures 19 and 20 below respectively.

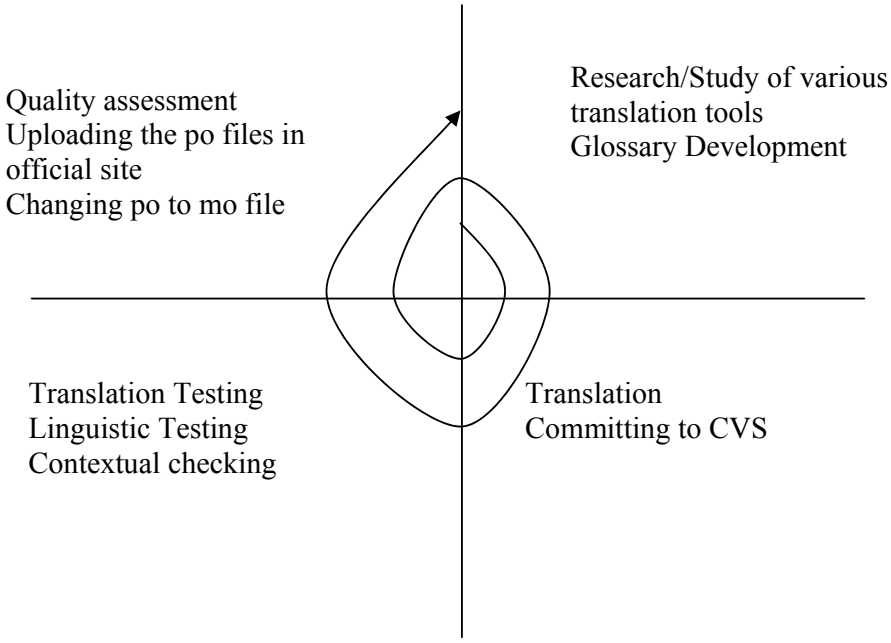

Figure 20. Spiral Lifecycle model for translation

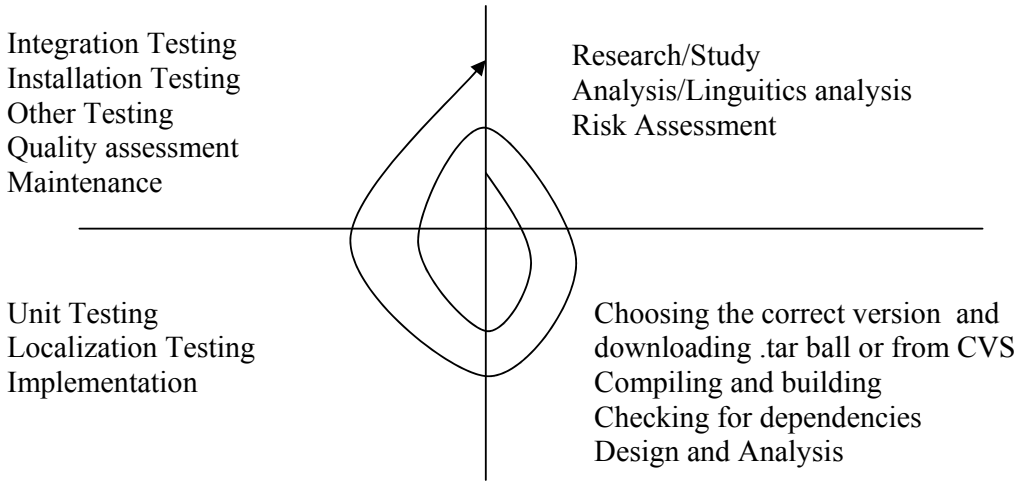

Figure 21. Spiral Life Cycle Model for Linux and NLP development

Creating an Environment

Translation Initiation:

Before initiation, the translation tools need to be installed and checked. All translation files need to be downloaded and all pot files need to be changed in the po files. For proper management of these files and the translation process, a CVS needs to be setup. The set up needs to be such that files are centrally located. General guidelines for translation need to be formed. And training document for the translators needs to be prepared.

Translation tool: Kbabel

Document Library:

There is a need of a Master library for all the programs, documents, reports, manual, specifications, reference documents and correspondence for the project. Documents for tracking and monitoring and controlling during all the phases of the software development process need to be kept. Templates, forms, and minutes need a place to be stored. Software documents files need to be stored. A document identifying

conventions needs to be formalized.

Document tools: openoffice.org.

Developing environment:
The developer establishes a software development environment in accordance with the requirement specification. If a system is developed in multiple builds, establishing the software development environment in each build should be interpreted as a means to establishing the environment needed to complete that build. Standardization of character set, keyboard layout, locale, fonts are mandatory hence these need to be prepared and must be readily available before the development starts. Similarly requirements of testing tools, compilers need to be identified before hand.

Project Management Environment
A yearly schedule of meetings and review dates can be decided on, at the initial stage. Milestones need to be set. Proper scheduling of task deliverables for completing certain amount of work to producing reports need to be decided upon. A baseline document on each of the unit needs to be laid down. In other words, it would be fitting to have individual project management activities for Linux Unit, Translation Unit and NLP Unit respectively. Moreover, it would be convenient to have different meetings and review schedules for each of these units. However, one monthly meeting of all the units would result in rendering effective communication and appropriate solutions. Proper documentation of assigning of works, work breakdown structure and organizational chart has to be produced. Even though man hour estimation may not be accurate, it is always good to have some metrics to measure it with.

An important aspect of project management is to track changes in the requirements specifications and the resulting changes in strategies of management activities. A proper change control management needs to be constructed. Our experience has shown that it is wise to be flexible with the dates in the research phase. For example, with the Nepali spellchecker component, it was decided in the initial phase that a new application would be built and integrated in openoffice.org. However, during the research phase, we realized that integrating a separate Spell Checker in the openoffice.org would consume more time than anticipated. Therefore, we had to search for other alternatives and it was the Hunspell which saved us a great deal of time. Consequently, the strategy for Nepali spell checker development had to be changed.

Project Monitoring
Project activities need close monitoring on weekly or adhoc basis. Once the baseline document is ready for each activity, progress can be measured by comparing, the work completed against the plan. Identifying and resolving risks at the earliest possible, prioritizing the tasks and, evaluating the work load of each team member and other related developmental and translation actions and schedules. An update form can be used to record the progress on these activities.

Joint technical and management reviews can be conducted for research, development and testing phases. The purpose of these reviews is to provide management with the progress of the development and translation activities, moreover it checks the fulfillment of requirements. Timely, technical and management reviews can help the team to take corrective action in time. Development review has to be done by the technical team and translation review should be done by translation manager. The review will have to focus on in-process and final software products, even for the report that will be generated after completion of each phase of the development cycle. In the review process, technical solutions need to be proposed identifying the short and long term risks. Experience with the translation shows that time management is the main risk factor as the strings with each new release of Linux component increase by 20%. Hence, keeping the size of the translation team the same and expecting to complete the translation on time without considering the above is very risky.

Project Completion
This stage covers the preparation of the deliverables such as software application, reports, papers, manuals. Making the software application ready for the user needs qualification testing of the software. Preparing the environment for the testing needs to be established. For these, developers should participate in developing and recording test preparation, test cases and test procedures to be used for system quality check.

The testing has to be conducted in the host organization. Testing should fulfill all the requirements provided in the requirement specification document. Following are the different types of testing employed while qualifying the end product, NepaLinux in our case:

1) Usability testing:
Since this Nepali Linux distribution has targeted the desktop users, making this product user friendly was the first priority. Features like auto mount, easy installation were added and tested thoroughly.

2) Hardware detection testing
This distribution was tested on different configurations of hardware.

3) Localization testing:
This testing was performed in order to check whether the system fails after loading the translated file.

Preparing the build of the software
Localization includes choosing the application, library routines etc. Hence, it becomes necessary to keep track of the correct version of these applications, libraries and routines that will be packaged in the final product. Even the correct versions of manuals, reports, translations files need to be recorded.

Preparing the user manual
The developers and the translators prepare the user manual. A framework needs to be worked out for the manual development. The user manual may also include installation manual.