# Internationalization & Localization of OpenOffice.org - The Indian Perspective

*"Comprehensive Office Suite for Multilingual Indic Computing"*

Bhupesh Koli, Shikha G Pillai

<bhupesh@ncb.ernet.in>
<shikha@ncb.ernet.in>

The Supercomputing People

**CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING**

**(Formerly NCST)**
68, Electronic City, Hosur Road,
Bangalore 561 229, India.
Tel: +91 80 852 3300
Fax: +91 80 852 2590
http://www.cdacindia.com/

# Contents

# *Abstract*

*India supports a culturally and linguistically diverse population, majority of whom are excluded from the productive usage of information technology due to the lack of standardized and economical Indian language enabled software.*

*OpenOffice.org is the leading office productivity suite through open-source initiatives, available cross-platform with internationalization and localization support for major International languages. This paper examines the development aspects, usage and prospects of OpenOffice.org internationalized and localized to cater to the Indian market.*

*Most Indian scripts originate from the Brahmi script and follow complex rules of layout involving consonants, vowels, special symbols, conjuncts and ligatures. Unicode encoding for Indian languages establishes a similar pattern among the scripts. We will examine these orthographic rules and how Complex Text Layout algorithms can be developed for Indian scripts in OpenOffice.org. Also explored are storage and rendering aspects of Indian text, along with font technologies suitable for Indian scripts.*

*The Internationalization (i18n) and Localization (l10n) framework of OpenOffice.org sets guidelines for localization and internationalization work of the suite in other languages. The project "BharateeyaOO.o" (http://www.ncb.ernet.in/bharateeyaoo) commenced on the lines of these frameworks, to achieve Indian language support in OpenOffice.org. With initiatives for localizations in major languages of India, Complex Text Layout support, Indian locales, dictionary support and collation algorithms, the project aims at a completely "Indianized" Office suite packaged economically for the Indian user. This paper concludes with an insight into the development, implementation details and progress of this project.*

# 1. Introduction

India is a country having eighteen constitutional languages, six thousand dialects and an estimated eight hundred fifty languages in daily use, and supporting 15% of the world's population. The growth and penetration of Information Technology in this developing nation is however, restricted to only 10% of its population, since the other 90% are yet to be conversant in the *lingua franca* of computer mediated communication – English. This calls for development of software supporting local-languages, allowing creation and manipulation of regionally relevant content, along with computing in local mediums. Such software need to be economically viable, and compliant to international standards to allow information dissemination transparently across economic, cultural and social barriers. Here, we analyze the prospects and developmental aspects of one such software - the internationally acclaimed product OpenOffice.org - in Indian languages.

# 2. OpenOffice.org in India

OpenOffice.org is the open source project through which Sun Microsystems has released the technology for the popular StarOffice[tm] Productivity Suite. With a worldwide developer community, open access to a common source base for high-productivity office applications that runs on all major platforms, and extensible internationalization and localization frameworks for development in all international languages, OpenOffice.org is an unparalleled effort towards open technology access in this computing era. For India, OpenOffice.org provides not only economic viability, but also extensibility in terms of Indian language development and support achieved so far.

Indian language support in OpenOffice.org aims to fulfill the basic requirement of having the commonly used collection of office applications on each Indian desktop. With support added for Indian localized interfaces and technical processing of Indian languages, the software will be in the forefront of Indian language development and information propagation across all digital barriers.

# 3. Overview of Indian Scripts

Of the eighteen official Indian languages, fifteen are of common origin – the ancient "Brahmi" script of India [1]. The other three namely Kashmiri, Sindhi and Urdu are of Perso-Arabic origin. Based on the Brahmi script classification and syllabic features, the characters of the fifteen major languages have been classified as Consonants (C) and Vowels (V). These two basic groups in various orders, along with graphical symbols (G) of the script, combine to form the orthographic syllable.

The organization of consonants and vowels, as well as the possible combinations form the basics of each Indian language. This organization results in explicit rules for creation of characters and conjuncts. Indian scripts generally exhibit complex behavior in syllable composition, formation and positioning. There also exist many shaping rules for each character, depending on the syllable it forms with other characters. These shaping rules may vary for individual scripts, but on the whole, based on the organizational structure and the classification of characters within each script (explained below), all major Indian scripts display a similar behavioral pattern.

## 3.1 Vowels – Independent and Dependent

Vowels in Indian languages have both independent and dependent forms. The independent form stands by itself, and emphasizes the sound that it represents. The dependent vowel, also known as **vowel signs/*matras***, is depicted in combination with a consonant or consonant cluster. The sound of this combination is the combined result of the sounds of the consonant/cluster and the vowel. Depending on the Indian script, these dependent vowels may attach itself to the any part of the consonant cluster: up, down, left or right. In certain scripts, combination of a consonant and dependent vowel may also change the inherent shape of both the vowel and consonant.

The Tamil script has been used here to highlight independent and dependent vowel combinations with a consonant:

| Independent Vowel Form | Dependent Vowel Form | Consonant | Combination |
|---|---|---|---|
| உ (U) | ◌ு | க (Ka) | கு |
| உ (U) | ◌ு | த (Ta) | து |
| உ (U) | ◌ு | ப (Pa) | பு |
| ஓ (Oo) | ◌ோ | க (Ka) | கோ |

## 3.2 Consonants – Full and Half Forms

Consonants are generally treated as the base of a syllable, and can be found having an inherent vowel, generally with the sound {a}. When combined with a dependent vowel, the sound of the inherent vowel is overridden with that of the dependent vowel.

A consonant may be visible in its full or half form, the latter depicting a consonant without the inherent vowel. This half form of a consonant is obtained by combining the consonant with a special character called a **virama/halant.** The halant kills the inherent vowel of the consonant and is used in forming consonant conjuncts.

Depicted below is an example of different conjuncts formed from two consonants in Hindi, using the halant ? and zero-width characters ZWJ and ZWNJ:

र + ् + क = क्क First consonant(RA) positioned above second (KA)

क + ् + र = क्र RA positioned below KA when it comes second

र + ् + ZWJ + क = क्‍क ZWJ added after halant creates half form of RA, which is then combined with KA

र + ् + ZWNJ + क = र्‌क ZWNJ prevents RA from combining: halant is visible here

### 3.3 Graphical Symbols – Modifier marks

Each Indian script also has modifier marks and diacritics that add to the consonant to produce some modified phonetic behavior. There also exist special symbols all having different combination roles, and Indian numerals within the script.

The following showing consonant, vowel and special character combination logic in Hindi; the boundaries of each cluster have been marked:

|क| + |ी| = |की|, |क| + |्| = |क्|,
|क| + |ी| + |ँ| = |कीँ|,
|क| + |्| + |र| + |ं| = |क्रं|,
|ष| + |्| + |ट| + |्| + |र| = |ष्ट्र|,

Graphical Symbols

# 4. Issues In Indic Computing

Enabling Indian languages in software requires focusing on character encoding, font, character input, rendering and shaping technology for each script. Described below are some of these issues:

### 4.1 Unicode for Indian scripts

Unicode encoding (16 bit) for Indic scripts is based on the 1988 draft of ISCII (Indian script code for Information Interchange), brought out by the Indian Department of Electronics (DoE), Government of India. The ISCII code set was evolved to exploit the common phonetic structure of Indian scripts, as an 8-bit coded character set. The lower 128 characters are from ASCII, while the top 128 are for one Indian script.
This arrangement sought to cater to all ten main Brahmi-based Indian scripts, with an INSCRIPT keyboard overlay providing a logical arrangement of the characters.

Nine main Indian scripts viz. Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada and Malayalam have been encoded in Unicode in the South and Southeast Asian script range [2]. The encodings are used in conjunction with the 8-bit transformation format UTF-8 to effect storage and rendering of Unicode text effectively. UTF-8 is compatible to ASCII ranges and transforms each Unicode Indian character to three byte representations.

After development of the Unicode standard, many issues have been brought up by Indic computing societies, about the incompleteness of the Unicode standard with regard to proper representation of all the Indian scripts. These are mainly issues regarding misrepresentation of some characters, missing characters and incorrect shape or annotation. The Government of India has proposed some changes to the existing encoding for Indian scripts, to bring out the best possible representation [3]. Apart from Unicode, there are also other encoding standards in India, like ISCII and some proprietary/font encodings for Indian languages. However, since Unicode is an international multilingual standard compatible across software platforms, it can be seen as a better option for most Indian languages.

## 4.2 OpenType

The OpenType specification[4] uses Unicode for character encoding and allows development of fonts containing large glyph sets and glyph variants. It allows font designers to map single characters to multiple glyphs and vice-versa, a single glyph for multiple characters.

OpenType tables like GSUB (glyph substitution) and GPOS (glyph positioning) allows complex positioning, definition of conjuncts and cluster formations, and encoding of specific script features. With such explicit script information, text-processing applications need only provide the processing details than concentrate on linguistic specifications. For Indian languages, OpenType fonts provide the best solution, and has proven results with respect to support for complex features within each script.

## 4.3 Indian Language Input Methods

Language Input in a software depend on the recognition of the language in the software as well as methods to input characters within the language. Input methods may be keyboard-based, through speech recognition, or dictionary/vocabulary dependent. Most common ways of Indian language input in software is through usage of logical keyboards based on mapping the standard keyboard. Since Indian language input in itself is complex in nature, as opposed to Latin scripts, keyboard layouts for Indian languages also require much deliberation before implementation and usage. The INSCRIPT keyboard, an initiative of the Department of Electronics, maps the standard QWERTY keyboard for Indic scripts.

The keyboard overlay exploits the logical ordering of Indian scripts, bringing about a common structure among all the scripts, with the vowels on the left hand side, and consonants on the right. The Windows platform Indian locales uses INSCRIPT keyboard layout for Indian character entry. On Linux, there are keymaps developed, used in conjunction with xkb or xmodmap, that use the INSCRIPT layout. There are also a variety of phonetic, regional and romanized keyboards available, however, some are proprietary and based on various other standards.

# 5. Indian Language Support in OpenOffice.org

## The BharateeyaOO.o project

The BharateeyaOO.o project (http://www.ncb.ernet.in/bharateeyaoo) was commenced at the National Centre for Software Technology (officially CDAC from April 2003) in 2001, to initiate development in OpenOffice.org through localizations and internationalization support within the suite for major Indian languages, on the prominent software platforms. Such an initiative was brought about to compensate for the lack of Indian language support in key software like office suites, which are almost every computer user's requirement.

The project targets implementation of the following features in OpenOffice.org for Indian languages

**Complex Text Layout:** Enabling CTL support in the suite, for all major Indian languages based on a generic framework, on Windows and Linux. This also involves development of Indian language solutions such as fonts, and input methods, according to requirement of the platform.

**Indian Locales:** Enabling locale data in the suite, for effecting currency, calendar, dictionary and collation specifications for each Indian language.

**Localization:** Translation and localization of the suite on Windows and Linux for the Indian languages – Hindi, Tamil, Kannada, Telugu, Punjabi, Marathi, Gujarati, Bengali and Malayalam.

The development of the project was based on the i18n (Internationalization) and l10n (Localization) frameworks of OpenOffice.org. The sections described below, highlight the developmental aspects of BharateeyaOO.o with insights into the solutions provided and observations made during implementation.

# 6. Internationalization (i18n)

The i18n framework of OpenOffice.org offers internationalization functionality within the suite. Encapsulated within two modules of the source i18n and i18npool, the framework designates internationalization requirements for western, CJK and complex script based languages. Addressing these requirements for Indian languages, addition of complex text layout support for major Indian languages, and locale data support was taken up.

## 6.1 Complex Text Layout

The presentation of Indian Language scripts requires contextual processing for output and display, due to its complex nature. Main issues that need to be tackled are those of conjunct formation from more than one character, and processing of such clusters within a document. For the requirements of CTL, i.e. text rendering, processing logic for arrow keys, positioning of caret and cursor, backspace/delete processing etc. character clusters lend a new dimension and each CTL event must be able to accommodate clusters and behave accordingly.

Some typical requirements of Indian script CTL processing are that the arrow and mouse events should result in the caret being positioned at cluster boundaries, and not within the cluster. Deletion should remove the entire cluster containing more than one code point, while a backspace operation should be allowed to remove characters one by one. Combining characters also need to be considered while selection, cut/copy/paste and text break issues.

### Finding a combination for Indian Languages

A text-processing engine for Indian scripts should be able to know character behavior, as in whether it can be involved in a combination with its neighbors. This involves logic processing for finding a combination in a specific language.

In OpenOffice.org, such logic can be developed within the i18n implementation of BreakIterator. The BreakIterator is used to find a character boundary, based on logic provided on the particular script. This character boundary logic, accessed through interfaces, can aid the text-processing engine in finding valid character combinations.

All major Indian languages that originate from the “Brahmi” script follow similar rules of combination. This feature can be used to develop a generic algorithm for character combinations. However, there are some exceptions in certain scripts, which may require specific implementation.

### Character combination affects Text width

Combination often results in a modification of a character's appearance or even a completely different character substituted. This leads to the cluster having a new width, rather than a width equal to the sum of individual width of characters involved in combination. If the text-processing engine places the caret according to the sum of the widths, this may lead to an error. Thus along with combination character processing, determination and calculation of text width is also necessary.

Combination width is not equal to the total of individual characters widths

| | | | |
|---|---|---|---|
| क+ी = की | क+् = क् | க+ி = கி | [2 characters in cluster] |
| क+ी+ँ = कीँ | क+्+क = क्क | | [3 characters in cluster] |
| क+्+र+ी = क्री | ഒ+ി+ഗ+ി = ഗ്രീ | | [4 characters in cluster] |

### Implementation of Indian script – CTL

Implementation of CTL for Indian scripts, based on Character combinations and their corresponding text width, is resolved to the following three major feature additions:

### Additional support required for Complex Script characters.

OpenOffice.org Visual class library [VCL] module provides access to different GUI systems. This module also defines the APIs to render multilingual text.
An array [WidthInfoArray] of type - structure [ImplWidthInfoData] has been defined to store the width of each unique character occurring in a document. If the character is used in the document once again, its width will be looked up in the array, instead of repetitive computation.

```
struct ImplWidthInfoData{
USHORT      mnChar;
Long        mnWidth;
};
```

For multilingual Indian text, comprising of complex characters, this becomes a limitation, since, there is no provision to store the width of the character cluster, which in itself has a unique value dissimilar to its components. CTL Processing is constrained here, as the exact widths of combinations are not stored.

To counter with this limitation, we propose a new layout of the structure for complex scripts. The member variable mnChar should be a string to store complex characters (consisting of more than one code-point).

Following is a possible definition:

```
struct ImplWidthInfoData{
USHORT      mnChar;      string  maChar;
long        mnWidth;
};
```

### Abstract interface to get a width of combination characters

Complex text may contain single character, combining characters or even multilingual text. Width needs to be calculated correctly based on each feature within the text. If characters are single (not combining), width can be calculated individual. In case of combining characters, we need to have an API that abstracts system level functionality to get exact width of the string.

The [VCL] module implementation contains a SalGraphics (System Abstraction Layer Graphics) class, where a function to get width of characters is defined. Here we propose to have a function, which can return width of strings (character cluster) passed as arguments.

```
Class SalGraphics {
...
GetCharWidth (nChar, nChar, nLen);
...
// It also requires an interface to compute string width

GetTextWidth (String, nLen);

};
```

### Modification in the text processing in source

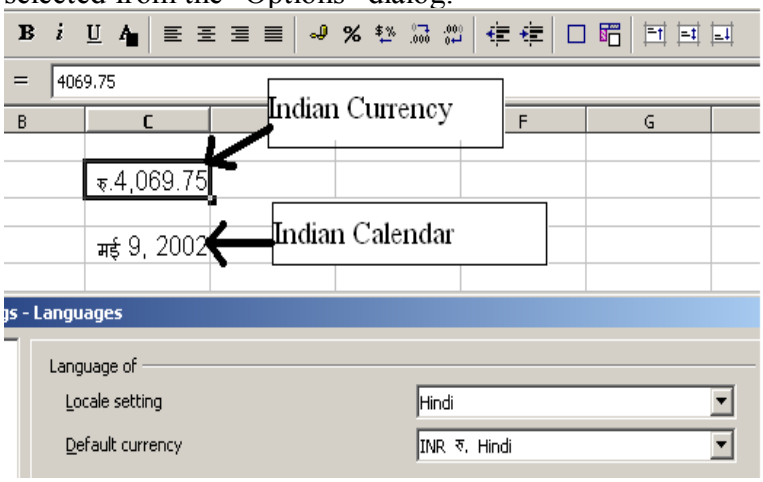
All width processing, spacing array and text break implementations have to be modified to accommodate the changes mentioned above. One such implementation can be as follows. The example is that of a function in the [VCL] layer definition of the OutputDevice class.

```
long OutputDevice::GetTextArray( String& , long* , nIndex, nLen ) const
{
...
for ( i = 0; i < nLen; i++ )
{
/* Check the character to see if it is from a complex script.  Call the BreakIterator APIs for
that language, to find out the character boundaries and their widths. According to whether
the character has combined with another one or not, width can be stored in the modified
array along with the single/combination character*/
}
}
```

All other CTL implementations like arrow processing, delete/backspace, etc. inherently uses the BreakIterator implementation, and hence will be resolved to work for the complex script text.

### 6.2 Locale Data

Indian Locale Data has been implemented to provide calendar and currency formats in Hindi. This involves adding a locale file (XML) to the i18npool module, from which it is parsed and compiled into a shared library. At runtime, the locale can be selected from the "Options" dialog.




# 7. Localization (l10n)

The Localization project of OpenOffice.org, residing at http://l10n.openoffice.org/ hosts a Localization (l10n) framework based on the multi-platform environment of the suite. The Framework[5] stipulates the method to build localized workspaces, with the introduction of a new language to the suite and clearly demarcates the necessary steps in terms of

- Getting Milestone workspaces
- Localizing and Building OpenOffice.org
  - Representing the language in the build environment
  - Extraction of strings from the source
  - Merging back translated strings
  - Rebuilding the localized code
- Switching to the newer workspaces

Localization of OpenOffice.org was planned with the two major languages: Hindi and Tamil. While the former is the national language and widely spoken in Northern India, the latter is a language spoken in Southern India. After successful localization in these two languages, other language localizations were to be started.

Translation within OpenOffice.org is required for strings composing the resources of the suite, definitions of the setup program and configuration files. These files have to be translated in the target language and rebuilt in a source which has the target language already added to the environment.

## 7.1 Localization Tools

The suite also provides localization tools such as transex3, lngex, cfgex and localized that parse the resource files to produce tab-separated files for translation, and allow merging back of the translated strings within these files to the source. The tabs are crucial for the merging back process. If the translation results in incorrect tab spacings then the file will be corrupted, and may hamper correct merging back of the strings.

While transex3, lngex and cfgex work for specific files, the localize tool extracts and merges strings from the entire source. These tools are located in the transex3 module and require the target language to be added to the transex3 source and rebuilt, for extraction and merging to be possible.

## 7.2 Glossary and Translation

The OpenOffice.org glossary, composing about 7000 strings, contains most of the words within the 21,000 actual resource strings to be translated. Using the glossary as a reference for translation, consistency across translations can be assured, as well as decreasing time required for reference.

Translation of the strings is to be done within the tab-separated files produced by the localization tools, in a simple editor, and saved in UTF-8 format. The font used for entry of translations need to be a Unicode font so that the UTF-8 values generated are correct. Only this can ensure that the application renders the correct strings on the user interface after localization. For Indian languages, OpenType fonts can produce best results.

Resource translations need to be apt and context sensitive as far as possible. For this, the context information of the strings within the files, i.e. the module information and user-interface element that holds the string, can be used to guess out the context during translation.

## 7.3 Representing the language in the build environment

Language support in OpenOffice.org is demarcated in six different modules of the source i.e. solenv, tools, svx, rsc, transex3 and the installation project comprising the modules scp and setup2.

Any language within the source is represented by six different parameters:

**Locale Identifier**: Reserved Hexadecimal LANGID defined in Windows. Can also be specified by a user, for languages that do not have an existing definition.

**Numerical Code:** Decimal equivalent of the LANGID.

**Language Identifier:** A unique two-digit identifier that represents the language.

**ISO-Code:** A short string, which indicates the language.

**Symbolic Code:** Name of the language. If the language is spoken in different regions, then a 2-character representation can be also added for that region along with the language.

**Environment Variable:** A variable to be set to TRUE to indicate that the build should be performed for that language.

The following table gives the values adopted for Indian localizations:

| Language | Locale Identifier | Numerical Code | Language Identifier | ISO Code | Symbolic Code | Environment Variable |
|---|---|---|---|---|---|---|
| Hindi | 0x0439 | 1081 | 91 | hi | “hindi” | RES_HINDI |
| Tamil | 0x0449 | 1097 | 92 | ta | “tamil” | RES_TAMIL |

The values assigned for each language across all the files should be uniform and consistent, for the user interface to be completely localized. The ISO-code may be predefined in the file isolang.cxx in the tools module, and if so, it would be a better approach to adopt the same value for all further modifications. The encoding format for the language also has to be specified within the tools module. For Indian languages, UTF8 encoding was used. (Specified as RTL_TEXTENCODING_UTF8 in the file l2txtenc.cxx).

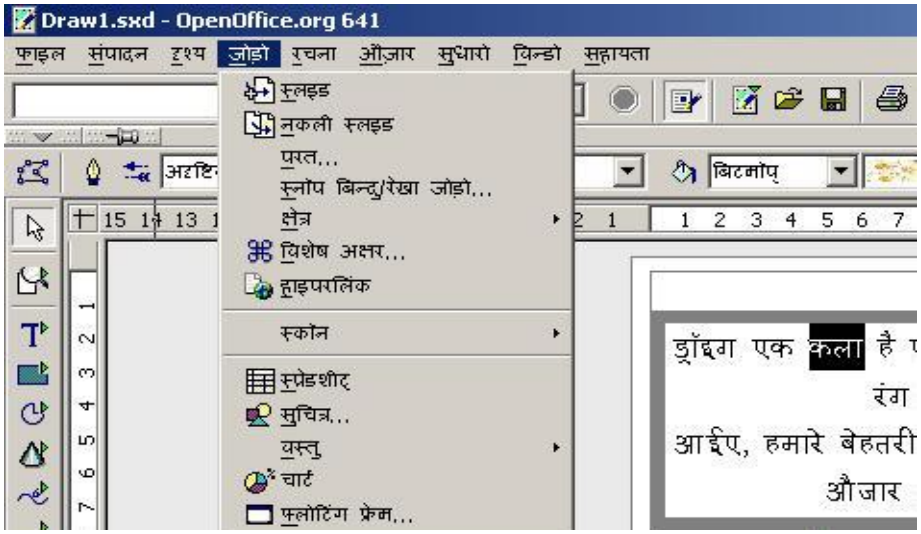
## 7.4 Building the installation set

A rebuild of the source with the localization modifications added will produce the new installation set. For the rebuild, the environment variable should be set to TRUE within the environment file (*.bat on Windows and *.set on Linux) and LANGEXT=## provided as argument to the dmake/build command. The installation sets after building are created in instsetoo/*.pro/##/normal. (*.pro maybe wntmsci7.pro or unxlngi3.pro on Windows and Linux respectively. ## is the Language identifier).

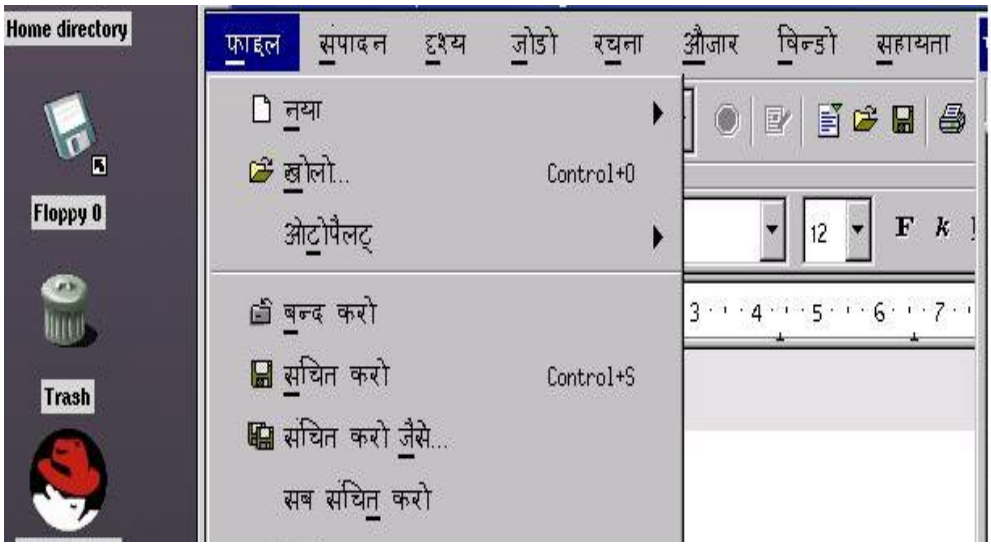
Following are some screenshots for the localization work in Hindi on Windows and Linux.

Localized strings on both platforms can be rendered using any OpenType font for that language. The preferences for font can be set within the source or at run-time.

**Localization on Windows – Draw (font is Mangal)**




**Localization on Linux – Text Document (font is Raghu)**




# 8. Future of BharateeyaOO.o Project

The Internationalization framework of OpenOffice.org can be extended to provide dictionary support, and locale data support for Hindi and other Indian languages. Implementation of sorting/collation for Indian languages also can be developed. Collation is the linguistic and culturally sensitive sort order of strings in a language. It is dissimilar to encoding order, esp. in the case of Unicode, since collation is language dependent while encoding is script based. Hence, for an Indian script like Devanagari, which is the base for Hindi, Marathi, Sanskrit and Konkani, collation[6]

orders have to be defined for each of the languages, all of which have different sorting requirements. For Indian languages, collation order should also be sensitive to multiple code points which form conjuncts, and "weights" given to consonant modifiers and special characters in the script. OpenOffice.org defines collator components that can be developed to produce collation algorithms for a particular language.

# 9. Conclusion

The availability of Indian language support in OpenOffice.org can be viewed as a major boon to Indian language computing and development. The multi-faceted features of this leading suite coupled with its availability in the open source domain, makes it an appealing and highly sought-for technology solution for Indian users. With this paper, we aim to introduce the possibilities of this suite in Indian languages and deliver an insight into the subsequent impact of its availability, on the propagation of Indian languages in the world of Information Technology.

# 10. References

[1] SP Mudur et al. Computers & Graphics 23 (1999) 7-24 "An architecture for the shaping of Indic Texts"
[2] http://www.unicode.org/uni2book/ch09.pdf: South and Southeast Asian script encoding in Unicode
[3] http://tdil.mit.gov.in/pchangeuni.htm: Proposed changes in Indian scripts given by the Department of Information Technology, Ministry of Communications and Information Technology, India
[4] http://www.microsoft.com/typography/otfntdev/indicot/default.htm: OpenType Indic script support
[5] http://l10n.openoffice.org/L10N_Framework/index.html: L10N Framework
[6] www.microsoft.com/middleeast/msdn/Indic_collation-DC.pdf : Issues in Indic collations

# 11. Acknowledgement

We would like to thank all the past members of this project as well as our other colleagues who have encouraged us in our work. We are indebted to members of our organization, NCST, for the pioneering research and development they have done in Indian Language Technology.

We would also like to extend our sincere gratitude to Dr. S.P Mudur, for initiating this work and inspiring us to do our best.

Finally, we thank all the OpenOffice.org team members for the wonderful support they gave us, and for providing assistance and answers to all our queries.

# 12. Copyright