Babel speaks Hindi

Zdeněk Wagner
Vinohradská 114
13000 Prague 3
Czech Republic
zdenek dot wagner (at) gmail dot com
http://icebearsoft.euweb.cz

Abstract

Babel provides a unified interface for creation of multilingual documents. Unfortunately no Indic languages are currently supported, so typesetting in Indic languages is based on specialised packages. The most advanced of these is Velthuis Devanāgarī for TEX, because it already provides Hindi values for language-dependent strings as well as a macro for a European-style date. A language definition file for plugging Hindi into Babel has therefore been recently developed.

The second part of the paper explains differences between Unicode and Velthuis transliteration. This is important for understanding the tool that can convert Hindi and Sanskrit documents from MS Word and OpenOffice.org into TEX via an XSLT 2.0 processor and a Perl script, as well as a method of making the PDF files searchable.

Finally the paper discusses some possibilities of further development: the advantages offered by XETEX and the forthcoming integration of Lua into pdfTEX.

1 Introduction

Packages for typesetting in various Indic languages in both plain TEX and LATEX have been available from CTAN for a long time. The authors of these packages have made substantial efforts to support the Indic scripts, which present difficulties that cannot be solved by TEX itself. For example, although generation of conjuncts and placing dependent vowels (matras) to subscripts and superscripts could be handled as ligatures in TFM files, the form of a conjunct depends also on language. While conjuncts क्त (kta) and न्न (nnna) are used in Sanskrit and traditional Hindi, they are nowadays often replaced with the half forms क्त and न्न, respectively. Such matters can be solved only by using a preprocessor bound to a (LA)TEX macro package.

The existing packages are used by indologists from all over the world as well as by people from India. It is therefore unfortunate that the packages support only the script and lack features to support the language. An exception is the Velthuis Devanāgarī for TEX package [1]. Since version 2.13, it contains definitions for language-dependent strings as well as a European-style date, and macros for switching between them and an English version. It therefore provides a kind of a mini-Babel. The further natural development step was to prepare a language definition file which will integrate Hindi into the Babel system.

2 Birth of the Language Definition File

The aim of our work was to enable transparent use of Hindi in multilingual documents by means of the standard Babel invocation:

```
\usepackage[hindi]{babel}
```

Preparing a language definition file for Babel is not a difficult task. It involves defining language-dependent macros such as `\chaptername`, and the date macro `\today`. These definitions were already present in the Devanāgarī package and they even properly handled switching between the Bombay, Calcutta and Nepali variants of the Devanāgarī fonts. However, placing them into the language definition file is not enough. Rendering these words in the Devanāgarī font requires a lot of special macros. Moreover, as mentioned previously, the Hindi text cannot be fed directly to TEX — preprocessing is mandatory.

It therefore seemed useless to copy the macros from the Devanāgarī package to the language definition file and follow the same development in two different places. It was therefore decided just to load `devanagari.sty` and redefine its options as language attributes. As a matter of fact, the language

definition file itself without the fonts and the preprocessor would not work at all. Thus the requirement to have the Devanāgarī package installed does not pose a real limitation.

The `devanagari.sty` package already contains captions and date macros for English. These definitions must not be removed because they are already documented and their removal might damage legacy documents. On the other hand these definitions collide with the Babel core.

The package was therefore modified so that the macros are declared by means of `\providecommand` and the definition delayed via `\AtBeginDocument`. The macros are thus guaranteed to exist and the Babel definitions have higher preference no matter in which order the files are loaded. The language definition file checks the package version and will complain if an old version of `devanagari.sty` is installed.

As already mentioned, it is not sufficient to activate the Hindi language either as the main language or by any of the Babel language switching environments. The text must be enclosed within a `{\dn ...}` group, or the preprocessor will not find it.

3 Unicode vs. Velthuis transliteration

Devanāgarī originates in an ancient Brāhmī script, and is an abugida. Each consonant (vyanjana) also represents an inherent following vowel — *a* in the case of Devanāgarī. Other vowels are added as diacritics in the vicinity of the consonant. Groups of consonants often form conjuncts, with only a minority of them written using a virama sign. Initial form of vowels have a different shape than vowel diacritics (matras).

Unicode is based on characters. The inherent *a* is not encoded. Thus U+0915 represents the क (ka) syllable (akshara). The syllable कि (ki) is represented by the two Unicode characters U+0915 and U+093F. Reversing the order of glyphs in displayed output is left to the rendering engine [2].

Independent vowels (initial forms) have distinct codes, e. g. the code of इ (i) is U+0907. The three characters U+0915 U+094D U+0924 denote a Sanskrit conjunct क्त (kta). However, such a glyph may not be present in modern Hindi fonts. The rendering engine will then display क्त. In order to force the rendering engine to display the latter form even though the Sanskrit ligature is present in the font, zero-width-joiner must be inserted. The Unicode encoding will then be U+0915 U+904D U+200D U+0924.

In contrast, the transliteration scheme developed by Frans Velthuis tries to be as close to scholarly practice as possible. Devanāgarī is traditionally transliterated into the Roman alphabet where long vowel, retroflex consonants, etc., are marked with diacritics [10]. The practice varies slightly between different textbooks and dictionaries. The Velthuis transliteration is a 7-bit encoding so the text can be input on a standard US keyboard. Transliteration is based on pronunciation, although in Hindi a short *a* in the middle of words, as well as at the end of words, is very often not pronounced. The inherent *a* in the middle of the word must always be written but the final short *a* can be omitted. Thus करना must be input as *karanaa* while घर can be written just *ghar*.

Important aspects of the differences between these two approaches will be shown in the following subsections.

3.1 Conversion into the Velthuis transliteration

Book production is often a collaborative work of author(s), editor(s), and a compositor. The authors rarely supply the texts in TEX; they commonly use other text editors, mostly MS Word. The first task is therefore conversion of the supplied manuscript into TEX. Almost all markup must be removed and replaced in order to achieve a particular graphical design.

We have found it is advantageous to open the manuscript in OpenOffice.org, and then save in its native format, which is XML. Although various tools are freely available, they still retain too much of the author's markup. Use of TEXML will also require much work which can hardly be reused in other books. Since the OpenOffice.org file format is XML, conversion can be performed by XSLT. A simple stylesheet can remove nearly all markup, keeping just boldface, italics and footnotes.

Another difficulty is that some Unicode characters are not directly available in TEX. Fortunately, Saxon 8.x [5], which implements XSLT 2.0, offers character maps, which allow replacement of such characters with TEX control sequences. However, this is not enough for the texts in the Devanāgarī script. If we just convert Unicode characters to corresponding Roman letters, all inherent *a*'s would be lost.

Although the result of the XSLT transformation can be fed into TEX, some postprocessing is recommended. Sometimes bold text is entered such that each character is emboldened separately. Merging

these into a single `\textbf` command is more easily done later in a Perl script than directly in the stylesheet. Also, the whole OpenOffice.org document is output on a single line. We need to edit the resulting file in order to wrap it to a reasonable width. Without any programming, this can be done by the Perl `Text::Wrap` module.

In addition to such formatting issues, we also split the main job of conversion between XSLT and Perl. First of all, long vowels can be encoded in the Velthuis transliteration either by doubling or by uppercasing. The stylesheet always uses uppercase for long vowels in order to prevent ambiguities. For instance, कई will be converted into *kaI* because *kaii* will be rendered as कैइ, which is wrong. Empty braces would also solve the problem, but using uppercase is easier. The independent vowels are transformed directly into the corresponding letter(s) in the Velthuis transliteration. The dependent vowels (matras) are preceded with an equal sign. The consonants are followed by equal signs and the virama is transformed into an underscore.

Afterwards the Perl script takes its turn. In the input, each paragraph appears on a single line to be processed. The first task is to form conjuncts. This will work for Sanskrit words as well as modern Hindi, where some ligatures are not used, e. g. in अड्डा. The virama will be added by the preprocessor. Conjuncts are created by

```
while (s/(\{\\dn [^}]*)=_/$1/) {}
```

In the next step matras are handled. Double equal signs are simply removed and lone equal signs denote missing inherent *a*'s to be added. This is achieved by the following lines.

```
while
 (s/(\{\\dn [^}]*)==([aAiIuU.eo])/$1$2/) {}
while (s/(\{\\dn [^}]*)=/$1a/) {}
```

Next, we remove the final inherent *a* unless we are converting a Sanskrit document.

```
while (!$opt_sanskrit &&
 s/(\{\\dn [^}]*)a([^.a-zA-Z])/$1$2/) {}
```

Finally the line is wrapped.

```
eval {
  print wrap('', '', $_);  1;
} or do {
  warn "Warning: $@";  print;
};
```

3.2 Searchable PDF files

PDF files play an important role as online documents. Although the chapter and section names may be placed into outlines and related parts may be found by hyperlinks, it is also desirable to be able to search for words and sentences. Here, Indic scripts present a big problem. PDF, similar to PostScript, deals with glyphs, but the field in the search dialogue accepts Unicode characters. Files generated by the Devanāgarī package are therefore unsearchable. XeTeX [13] can use OpenType fonts, but it turns out that a PDF file created by XeTeX is not searchable either, although the situation is not simple. XeTeX can use several engines for PDF creation and the results differ. OpenOffice.org is slightly more successful because all simple words which do not contain conjuncts are searchable.

The key problem stems from the distinction between glyphs and characters. Mappings between glyphs and characters can be inserted into so-called 'ToUnicode' maps. This feature is already implemented in the `cmap.sty` package. An experimental ToUnicode map was therefore developed for the Velthuis Devanāgarī package. Since pdfTeX inserts these maps according to the font encoding, each Indic script will require its own encoding name for LaTeX. Currently Devanāgarī, Bengali, and Gurmukhi use encoding U, and all of them rely on preprocessors with analogous functionality. In the present experimental project, encoding X0900 was used in order to refer to the corresponding Unicode block.

The `dvng` fonts contain single characters, ligatures and pieces which are glued together by TeX macros. Single characters and ligatures are directly mapped to Unicode characters. Half forms of the consonants are mapped to the corresponding characters followed by a virama. Vattu is added to a full consonant, therefore it is mapped to the र (ra) consonant preceded by virama. This makes words with conjuncts searchable.

Unfortunately it is not possible to solve all problems. Since PDF works with glyphs, i-matras always precede the consonants. When searchng such words, it is necessary to type them into the search field as they appear, i. e. to write the i-matra in front of the consonant. Acrobat Reader is also confused by placing vowel diacritics below or above consonants as well as by vattus. An extra word boundary is created. Thus when searching for कुल्लू, two words कु ल्लू must be typed into the search field. The word ड्राइवर must be entered in a way impossible in the Velthuis transliteration, namely as two words ड् राइवर. Unicode allows starting a word with a dependent vowel which is, of course, incorrect. This misfeature enables making such words searchable. Extra word boundaries are not formed if the akshara is drawn as a glyph in the `dvng` font. The words रुकना and मात्रा can be searched for as such. More detailed

information is present in the documentation of the experimental ToUnicode map [12].

The ToUnicode map is also used when copying and pasting the text from a PDF file to another application. It is no surprise that we get into the same problems. The pasted text will contain extra spaces after vattu and both short and long u-matras. Words with i-matras will look visually correct but their Unicode representation will be wrong. If a word such as दिल्ली is copied and pasted from PDF and then sent to a sorting algorithm, it will appear in a nonsensical place, because the program will see a word starting with i-matra which is not allowed by Hindi orthography.

4 Future development

The greatest disadvantage of contemporary Velthuis Devanāgarī for TEX is the necessity of using the preprocessor. It does not seem so uncomfortable if a single language document is being prepared. However, if a trilingual document in Hindi, Bengali and Panjabi is being typeset, the source file has to be run through three preprocessors in series. The `devnag` preprocessor can handle a few Bengali or Panjabi words within a Hindi paragraph using angle brackets but not all preprocessors are that advanced. The necessity of using the preprocessor brings about difficulties with index creation which have not been solved yet. Replacing the preprocessor with another mechanism is thus an important step forward.

The first idea was to reimplement the preprocessor in encTEX [6]. It hooks into TEX's mouth and converts input characters to arbitrary tokens according to the conversion table. Conversion can be switched on and off by changing the value of `\mubytein`. The conversion table can also modify the file output of `\write` according to value of `\mubyteout`. However, since the conversion acts upon characters in the TEX's mouth, it is not possible to distinguish between characters within words and characters within control sequences. Reimplementation of the preprocessor in encTEX would thus be very difficult and the resulting code inefficient.

More promising is integration of the Lua scripting language [3] into pdfTEX. The preprocessor can be reimplemented in Lua and moreover new features can be added. It will be possible to read texts both in the Velthuis encoding and in Unicode. It will also be possible to process documents already run through the preprocessor so that compatibility will not be lost. When and if Lua is integrated into XETEX, it will be possible to choose between `dvng` and OpenType fonts. Having hooks to TEX's mouth as well as output in Lua, it will be easier to implement indexing software in Indic languages.

5 Requirements for multilingual environment

Multilingual support is required not only in TEX, but also in the whole operating system. First, it is necessary to display the Unicode characters correctly. Groups of consonants with viramas must be properly combined into conjuncts and matras moved to the correct place. It is achieved in Linux by one of the following libraries: ICU [4] and Pango [8]. However, these libraries are not yet used by all programs. Even Firefox does not use Pango by default, it must first be activated by setting `MOZ_PANGO_ENABLE=1`. MS Internet Explorer displays Hindi texts correctly.

The text must also be prepared using Unicode. There remain problems with editors under Linux. OpenOffice.org, gEdit, and <oXygen/> (XML editor) [7] work well; yudit is also said to work, but I have not tried it. Support for Indic scripts in other editors is still missing.

Input in Indic scripts must, of course, be accepted in all applications and displayed correctly. This is a problem with Adobe Acrobat Reader under Windows. I did not manage to enter anything into the search dialogue directly from the Hindi keyboard in a usable way (it displayed something but did not search). It is possible to copy and paste the text from another application but it is displayed in Unicode sequences. A comparison of Linux and Windows versions is shown in Figure 1. The Linux version is more comfortable for users but both search equally well.

The search facility and copying texts from PDF require modification in the CMap encoding. Currently 1:1 and 1:many mappings are available. In order to be able to reverse the order of glyphs and handle two part vowels in Dravidian languages, a many:many mapping is needed. For better understanding, a few Devanāgarī and Malayālam sylables are shown in Table 1.

Table 1: Selected Devanāgarī and Malayālam syllables.

Meaning	**Devanāgarī** देवनागरी	**Malayālam** മലയാളം
ma	म	മ
maa	मा	മാ
mi	मि	മി
mii	मी	മീ
me	मे	മെ
mo	मो	മൊ

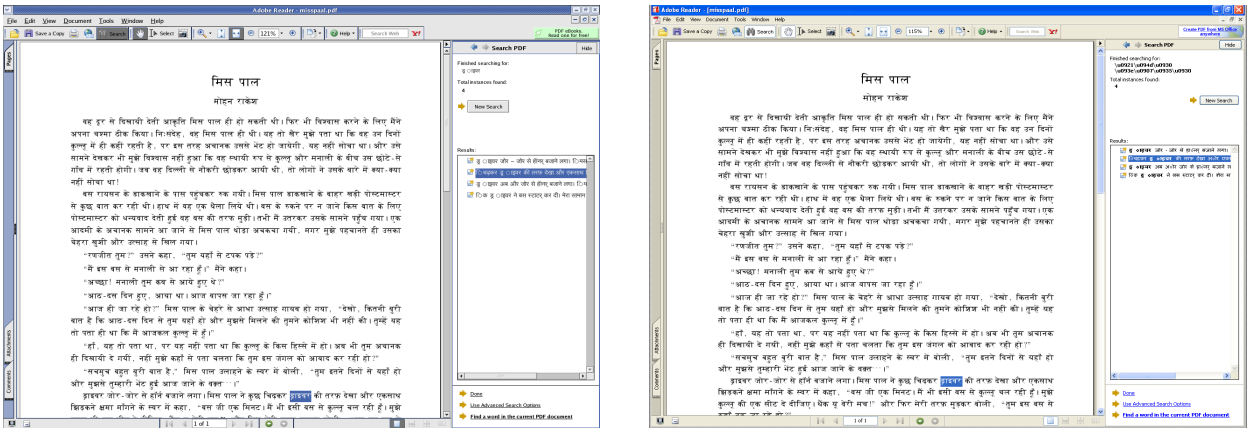

Figure 1: Comparison of search for ड्राइवर (input as ड् राइवर) in Adobe Acrobat Reader 7 in Linux (left) and Windows XP (right).

6 Conclusion

The paper describes how the Babel module for Hindi was made. It further presents thoughts concerning the use of Indic texts as online documents. Tools for conversion from MS Word and OpenOffice.org into TEX and for making PDF files with Devanāgarī texts searchable are presented. These tools are available from the author's web page [11].

7 Acknowledgement

The author would like to acknowledge the work of other developers of the Velthuis Devanāgarī for TEX, as the package is the base for this project, as well as Karel Píška for converting the Indic fonts to the Type 1 format [9]. Special thanks belong to Anshuman Pandey for translating the language-dependent strings into Hindi, and John Smith and Arnošt Štědrý for providing test files created by XETEX. The author would also like to acknowledge Alexandr Babič for running the test under Ubuntu and Petr Tomášek for explanation of topics related to font rendering in X. Finally, the author would like to acknowledge financial support of his attendance of the TUG 2006 conference by CSTUG and TEX Users Group.

References

[1] Devanāgarī for TEX. http://devnag.sarovar.org/.

[2] Joan Aliprand et al. *The Unicode Standard*, chapter South Asian Scripts. The Unicode Consortium, 2003. http://www.unicode.org/faq/indic.html#5.

[3] Hans Hagen. LuaTEX: Howling to the moon. *TUGboat*, 26(2):152–157, 2005. http://www.tug.org/TUGboat/Contents/contents26-2.html.

[4] International components for Unicode. http://icu.sourceforge.net/.

[5] Michael Kay. Saxon, the XSLT and XQuery processor. http://saxon.sourceforge.net/.

[6] Petr Olšák. encTEX. http://www.olsak.net/enctex.html.

[7] <oXygen/> XML editor. http://www.oxygenxml.com/.

[8] Pango. http://www.pango.org/.

[9] Karel Píška. Indic Type 1 fonts for TEX. CTAN:fonts/ps-type1/indic.

[10] Transliteration pages. http://homepage.ntlworld.com/stone-catend/translit.htm.

[11] Zdeněk Wagner. My free software. http://icebearsoft.euweb.cz/sw.php.

[12] Zdeněk Wagner. Searchable PDF with Devanāgarī texts. http://icebearsoft.euweb.cz/dvngpdf/.

[13] The XETEX typesetting system. http://scripts.sil.org/xetex.