The Asian Journal of TEX, Volume 2, No. 1, April 2008
Article revision 2008/3/15

Unicode Support in the CJK Package

Werner Lemberg
Municipal theatre of Koblenz, Germany
Kleine Beurhausstraße 1
44 137 Dortmund, Germany
wl@gnu.org

KEYWORDS Unicode, UTF-8, CJK, LATEX, CJKutf8, font encoding, character set, XETEX, PDF, bookmarks.

ABSTRACT This article describes how the CJK package for LATEX handles Unicode encoded characters. Using the `CJKutf8.sty` file, LATEX's native Unicode handling is combined with CJK's capabilities for CJKV scripts. Another recent extension is support for XETEX which is here described also.

1 Basics

I want to start this article with the introduction of some important concepts which are needed to correctly understand the sometimes tricky details of this topic. Many people use the terms described in this section quite sloppily; this often leads to heavy confusion, as can be easily seen, for example, in the archives of the CJK mailing list [1].

1.1 Characters and glyphs

Characters are entities which have a semantic meaning. Visual presentation forms of characters are called *glyphs*. A character can be represented by more than a single glyph — just think of an italic *A* and a sans-serif A. However, this mapping is not a one-to-one relationship; for example, the glyph A could be, among other characters, either a Latin, a Cyrillic, or a Greek uppercase letter.

TEX, as do basically all text formatting programs, expects characters on the input side and emits glyphs on the output side.

1.2 Character sets and input encodings

A *character set* is a collection of characters. An *input encoding* assigns code points to each element of a character set. A corollary of those separate definitions is that it is possible to have more than a single encoding for a given character set. For example, the Korean character set KS X 1001:1992 (formerly called KS C 5601-1992) can be used as the main encoding in an EUC (*Extended Unix Code*) encoding with code points in the range 0xA1A1–0xFDFE (with the second byte covering only the range 0xA1–0xFE), or it can be mapped onto Unicode (with many gaps for all Unicode characters not in KS X 1001).

An excellent reference for Asian character sets and encodings is Ken Lunde's book *CJKV Information Processing* [2].

1.3 Font encodings

Internally, each font, regardless of its format, uses either indices or names to access its glyphs. For some font formats, this is directly visible; below I discuss the most important ones of this kind, namely PostScript Type 1 and CID-keyed PostScript fonts.

1.3.1 Glyph names and indices

Type 1 fonts [3] can contain more than 256 glyphs, but the standard way to access them is using an *encoding vector* which maps at most 256 glyph names to codes 0x00–0xFF; the rest of them are unencoded. In case you want to access unencoded glyphs you have to change the encoding vector to make them encoded (and others unencoded).

Glyphs of CID-keyed PostScript fonts [4] do not have names. Instead, the developer of this font technique, Adobe, has defined unique *character collections* (which are ordered glyph collections actually, using the terminology introduced in subsection 1.2), giving each glyph a fixed index (the *CID*, character ID, which should rather be called a glyph ID). An example for such a collection is 'Adobe-GB1-0', which defines an ordering of glyphs for the Chinese GB 2312-80 character set. Note that collections for Asian scripts contain, among other things, some extra glyphs for vertical typesetting [5].

TEX fonts belong in this category too but their format is the most primitive of all of them: At most 256 glyphs are contained in a TEX font, to be accessed with indices only.

1.3.2 Character maps

Other font formats like OpenType [6] use a different route to access glyphs by providing *character maps* (also called 'cmaps'), depending on the used platform and encoding. A typical OpenType font to be used with MS Windows in Unicode encoding has a (3,1) cmap: Value 3 is the *PID*, the platform ID (Windows), and value 1 is the *EID*, the encoding ID (Unicode).

Character maps exist for CID-keyed fonts too (called 'CMaps' in Adobe's parlance). For example, the CMap 'KSC-V' maps the ISO-2022-KR input encoding to the 'Adobe-Korea1-0' collection, using vertical glyph representation forms [5].

While CMaps for CID-keyed fonts can select glyphs for either horizontal or vertical writing, this is not possible with OpenType cmaps. Instead, a different mechanism is used: the selection of *features*, based on scripts and languages. Assuming that an OpenType font supports vertical typesetting, it is possible, for example, to select the feature 'vrt2' (vertical representation forms[1]) for the script 'hani' (CJK ideographic script) and language 'dflt' (default, that is, to be applied for all languages belonging to the 'hani' tag). The sets of tags for feature, script, and language names are predefined.[2] XETEX supports selection of OpenType features, scripts, and languages.

1. An older version of this feature is called 'vert', which contains a subset of the glyphs in the 'vrt2' feature.
2. It is possible to extend these sets if necessary.

Unicode binary representation	UTF-8 binary representation
U+0000–U+007F	
00000000 00000000 0*xxxxxxx*	0*xxxxxxx*
U+0080–U+07FF	
00000000 00000*yyy yyxxxxxx*	110*yyyyy* 10*xxxxxx*
U+0800–U+FFFF	
00000000 *zzzzyyyy yyxxxxxx*	1110*zzzz* 10*yyyyyy* 10*xxxxxx*
U+10000–U+10FFFF	
000*uuuzz zzzzyyyy yyxxxxxx*	11110*uuu* 10*zzzzzz* 10*yyyyyy* 10*xxxxxx*

TABLE 1. The relationship between Unicode values and UTF-8 encoding.

1.4 Unicode and UTF-8

Unicode [7] is a very large character set; the current major version 5.0 contains 99 024 characters; 70 229 of them are CJKV ideographic characters. A Unicode value is a number between 0x0 and 0x10FFFF (comprising 1 114 112 code points); this means that about 9% of the available code space is already allocated, with plenty of space for future additions. Note that about 12% of the code space is reserved for private use.[3]

Using Unicode values directly within a data stream is not possible since they are, as mentioned earlier, numbers and nothing else. Various formats have been defined for data exchange; the most common encoding forms are UTF-16 (each Unicode character has a length of either two or four bytes), UTF-32 (all Unicode characters have a length of four bytes), and UTF-8 (Unicode characters have variable length). For TEX, only UTF-8 is of interest; the other two formats cannot be handled.[4] Table 1 shows how Unicode is represented as UTF-8.

The layout of UTF-8 is very clever, for a number of reasons:

1. The ASCII characters stay unmodified.
2. Looking at the leading byte of a UTF-8 byte sequence it is immediately known how many bytes follow;
3. All non-leading bytes have the same format, and can never be mistaken for leading bytes.

Property 1 guarantees that UTF-8 works within single-byte environments. Properties 2 and 3 allow easy resynchronization in case of a data stream error — most other multibyte encodings, in particular all EUC encodings, SJIS, and Big 5, are lacking this very useful feature: A single missing byte probably makes the rest of a document unreadable.

1.5 CJKV scripts and Unicode

As Unicode encodes characters, not glyphs, CJKV characters (this is, the ideographic script developed in China thousands of years ago which is or was in use for Chinese,

3. For the ordinary user, Unicode and the ISO/IEC 10 646 standard are the same. I will not go into more detail here.
4. However, XETEX can digest UTF-16 too.

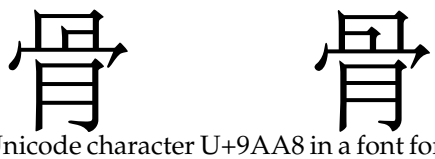

TABLE 2. The Unicode character U+9AA8 in a font for traditional Chinese (left) and simplified Chinese (right).

TABLE 3. The Unicode character U+9038 in a font for traditional Chinese (left), Korean (middle), and Japanese (right).

Japanese, Korean, and Vietnamese) are 'unified' if possible.[5] The example given in the Unicode standard book to demonstrate this unification process is character U+9AA8 (see table 2).

A more extreme example is character U+9038 (table 3), where even the writing order and stroke numbers differ.

Given these examples it is obvious that unification on the input side needs localization on the output side. In other words, you get ugly results if you use a Chinese font for Japanese and vice versa. It is thus a bad idea to use a single font like the notorious `cyberbit.ttf` to 'print CJK characters', as naïve users may believe — it can be used, say, as a fall-back font for Web browsers if no other font can handle a particular character, but for good typography it should be avoided.

2 LaTeX's Unicode handling

Within LaTeX, the standard inputenc mechanism is used to load support for UTF-8 encoding; the macros can be found in file `utf8.def`. The mechanism is quite simple: Assuming that we have input character Œ (which is U+0152, LATIN CAPITAL LIGATURE OE), `utf8.def` maps it to the macro `\u8:Œ`. Here, 'u8:Œ' is a single macro name defined with `\csname`; the 'Œ' in this name is encoded in UTF-8.

The various `\u8:`*x* macros are defined in so-called *Unicode definition files* (which have the extension `.dfu`) using the `\DeclareUnicodeCharacter` macro; for each LaTeX font encoding a DFU file should be provided. For example, the first few data lines of `t1enc.dfu` look like this:

```
\DeclareUnicodeCharacter{00A1}{\textexclamdown}
\DeclareUnicodeCharacter{00A3}{\textsterling}
\DeclareUnicodeCharacter{00AB}{\guillemotleft}
\DeclareUnicodeCharacter{00BB}{\guillemotright}
\DeclareUnicodeCharacter{00BF}{\textquestiondown}
```

5. With 'possible' it is meant that round-trip compatibility to major Asian character code sets like JIS X 0208 or GB 2312-80 has preference — if two characters would be unified by the Unicode principles but one of those major character sets has two code points for them, they are retained as two different code points. Details to the unification guidelines can be found in the Unicode standard [7].

```
\DeclareUnicodeCharacter{00C0}{\@tabacckludge‘A}
\DeclareUnicodeCharacter{00C1}{\@tabacckludge’A}
\DeclareUnicodeCharacter{00C2}{\^A}
\DeclareUnicodeCharacter{00C3}{\~A}
...
```

Using a Unicode character in the document which has not been defined in a DFU file causes an error.

It is highly recommended to use the babel package or a similar mechanism to switch between languages so that the proper font encodings (which in turn cause the loading of the DFU files) are selected. For correct hyphenation this tagging is needed anyway.

3 The Unicode handling of the CJK package

UTF-8 support within the CJK package[6] is straightforward: Similar to the inputenc package, all possible first bytes of the UTF-8 encoding have been made active characters. Then a proper subfont is selected for typesetting the particular character, after converting the UTF-8 byte sequence to a Unicode value. For example, to typeset U+3456 for font family ‘foo’, the glyph with index 0x56 in subfont ‘foo34’ is selected. Obviously, a single CJK subfont covers exactly 0x100 (256) glyphs. Since there are not ligatures or kernings between CJKV ideographic characters, this approach is sufficient for most East-Asian scripts.[7] However, to typeset a language like French which needs glyphs from different Unicode subfonts — just think of the Œ example given above — this solution is not sufficient because TEX does not support kerning between different fonts.

A recent addition to the CJK package is the support of the whole Unicode range; versions older than 4.7.0 support only the *BMP* (Basic Multilingual Plane, U+0000–U+FFFF). Subfont names for character ranges above U+FFFF have four lowercase hexadecimal digits appended (for example ‘foo025e’ which covers range U+25E00–U+25EFF). Subfont names within the BMP use only two lowercase hexadecimal digits.

Within a CJK environment, LATEX’s input encoding mechanism is disabled.

4 The CJKutf8 package

There are at least three serious disadvantages if using the CJK environment as-is for UTF-8:

1. LATEX’s input encoding mechanism does not work; in particular, only glyphs from the selected CJKV font are used for characters larger than U+00FF which usually gives ugly results for all glyphs not related to CJKV typesetting.

6. A general description of the CJK package has been given elsewhere [8].
7. This is not completely correct since, especially for vertical typesetting in Japanese newspapers, glyphs like — (U+4E00) could be kerned to compress the text vertically.

2. Non-CJKV scripts which need glyphs from more than a single subfont probably miss kerning and ligature data. Since only a small percentage of languages in the world can be typeset with ASCII alone[8]), all other languages using the Latin script are affected.

3. The probably most serious problem is that TEX's hyphenation mechanism will fail too because hyphenation is restricted to words which are typeset without a (TEX) font change.

Now enters the CJKutf8 style file. It loads the CJK package and activates the UTF-8 handling of LATEX's inputenc package, then it modifies the internal font macros of the CJK package to first check whether a `\u8:`*x* macro exists for a particular character, and to use it if it is available. Doing so, CJKutf8 combines LATEX's Unicode definitions with CJK's handling of CJKV ideographs. This immediately resolves items 2 and 3, and item 1 is now a non-issue since those LATEX macros map to the proper glyphs of the current LATEX font.

The document you are reading right now uses CJKutf8; it has the following structure:

```
\documentclass[12pt, DIV10]{scrartcl}

\usepackage{cmap}
\usepackage[T1]{fontenc}
\usepackage{CJKutf8}

% We need this for the word Hawai‘i
\DeclareUnicodeCharacter{02BB}{‘}
...
\newenvironment{TChinese}{%
  \CJKfamily{bsmi}%
  \CJKtilde
  \CJKnospace}{}
\newenvironment{Japanese}{%
  \CJKfamily{min}%
  \CJKtilde
  \CJKnospace}{}
...
\begin{document}
\begin{CJK}{UTF8}{}
...
\end{CJK}
\end{document}
```

8. You can find a list of them at http://blogamundo.net/unsorted/asciilangs.txt; among those 60 or so of about 6900 languages in the world you find, for example, Basque, English, Indonesian, Latin, Swahili, and Tagalog, but not Hawai‘ian: The ‘ character in the word Hawai‘i should be U+02BB, MODIFIER LETTER TURNED COMMA, not U+0060, GRAVE ACCENT.

UTF-8 binary representation	UTF-16 binary representation
11110*uuu* 10*uubbbb* 10*bbcccc* 10*dddddd*	110110*aa aabbbbbb* 110111*cc ccdddddd*

TABLE 4. The surrogate pair mechanism of the UTF-16 encoding. In the above table, $aaaa = uuuuu - 1$, $0 < uuuuu \leq 10000_2$ ($10000_2 = 16_{10}$). Example: U+10302 is equivalent to the UTF-8 byte sequence 0xF0 0x90 0x8C 0x82, which is equivalent to the UTF-16 surrogate pair 0xD800 0xDF02.

4.1 Bookmark support for pdfTEX

The hyperref package provides, among many other things, support for PDF bookmarks [9], together with hook macros to adapt it as necessary. The encoding for bookmark strings is either a special encoding which is called `PD1` in the hyperref package or UTF-16,[9] called `PU`. Since the CJK package bypasses the standard character handling of LATEX for CJKV characters, we must indeed use the hook `\pdfstringdefPreHook` to provide special routines which convert the CJKV characters to UTF-16 entities. A small complication is that Unicode values above U+FFFF are emitted as *surrogate pairs* (see table 4).

The CJKutf8 package does this conversion using standard TEX features only; it cleverly applies large `\ifcase` structures to extract the needed data. hyperref itself has bookmark support for LATEX's UTF-8 input encoding too; however, it uses ε-TEX features to implement it (and it does not work with the CJK package).

4.2 Cut and paste support for pdfTEX

As soon as TEX has converted characters to glyphs, the information content of the input has been lost. Sometimes, however, applications need this information, for example, to search text in a PDF document or to cut and paste data into another application. In case the fonts used in the PDF contain glyph names following the *Adobe Glyph List* [10] or use one of a predefined set of known PostScript CMaps or character collections (see section 5.9.1 in [9] for the complete algorithm), the extraction of the information content works out of the box. Otherwise, *ToUnicode* CMaps are needed which provide the proper glyph-to-character mapping.[10] Especially for the Type 1 versions of the CM fonts you need this because the glyph names used in those fonts predate the AGL and are thus non-standard.

Currently, there are two competing possibilities to create ToUnicode CMaps with pdfTEX. The first and older one uses `\pdffontattr` to add PDF code which directly attaches a ToUnicode map to a font. The second one, introduced in 2006, uses the (currently completely undocumented) `\pdfgentounicode` primitive which does approximately the same but with a TEX-like syntax. However, both methods have limitations: The former does not support virtual fonts, and the latter works with Type 1 fonts only.

9. To be more precise, bookmarks use UTF-16BE; this is big-endian UTF-16 encoding. *Big-endian* means that within a byte stream the byte holding the higher eight bits of a 16-bit number are emitted first, followed by the byte with the lower eight bits.

10. The PDF standard also defines an *ActualText* structure element which makes it possible to directly embed the input data into a PDF file in a hierarchical manner. As far as I know, pdfTEX does not support automatic generation of this.

Владимир Волович (Vladimir VOLOVICH) has written the cmap package which provides automatic ToUnicode CMap support for many common LaTeX encodings; the CJKutf8 package does the same for CJK fonts in Unicode encoding.[11]

5 The XeTeX extensions of the CJK package

In 2007, 孙文昌 (SŪN Wén-Chāng) started work on an extension to the CJK package to make it support XeTeX. After presenting it on the CJK mailing list, Wén-Chāng and I have improved it further in a collaborative effort, and it is now available in the git repository of the CJK package (http://git.sv.gnu.org/gitweb/?p=cjk.git;a=summary).

The idea behind xCJK, as he calls this extension, is to provide the following features:

- Use XeTeX's font support for system-wide fonts. This basically means that you no longer have to fiddle with CJKV subfonts. However, the separate font handling of CJKV and non-CJKV fonts should be retained; this is achieved by implementing `\setCJKmainfont` (in analogy to `\setmainfont` as defined in the standard fontspec style file for XeTeX).
- Provide style options `BoldFont` and `SlantFont` to select fake bold and slanted fonts.
- Provide CJK punctuation character support as with the CJK package.
- Provide bookmarks for PDF similar to CJKutf8.

From a technical point of view, xCJK disables XeTeX's native UTF-8 handling by setting both `\XeTeXdefaultencoding` and `\XeTeXinputencoding` to the value 'bytes'; the standard CJK macros are used to parse UTF-8, and which have been redefined to output real Unicode characters.

Here an example skeleton which demonstrates the usage of xCJK.

```
\documentclass{article}
\usepackage{xCJK}
\setmainfont[Mapping=tex-text]{Times New Roman}
\setCJKmainfont{FZKaiTi}

\begin{document}

\begin{CJK*}{UTF8}{}
... some Chinese text ...
\end{CJK*}

\end{document}
```

11. The code for this feature is based on the file `cjk-unicmap.sty` which was then part of the hangul-ucs package written by 김도현 (KIM Dohyun) and 김강수 (KIM Kangsoo).

Note, however, that this emulation mode is only suitable for CJK characters; due to the computation of subfont offsets, kerning (which comes from TFM files) is lost. Additionally, more recent versions of XeTeX provide far better support for CJK scripts than previously, thus xCJK can be considered as an interim solution.

6 Conclusion

Features introduced recently in the CJK package make it easier than ever to write LaTeX documents encoded in Unicode. Unfortunately, it will never be able to support certain typographic elements needed for proper CJKV typesetting due to limitations in LaTeX and XeTeX themselves. About ten years ago, Γιάννης Χαραλάμπους (Yannis HARALAMBOUS) and John PLAICE caused great excitement in the TeX world by introducing Ω (Omega), a TeX extension. It seemed that Ω finally solved many problems which could not be handled before.[12] However, for various reasons, the development of Ω (as we know it from, say, CTAN) has stopped. Meanwhile, Taco HOEKWATER is working on LuaTeX which tries to take the best features of Ω, ε-TeX, pdfTeX, and XeTeX, and which is using Lua [12] as a built-in scripting language. Hopefully, this time the project will succeed!

Many thanks to Barbara BEETON and Gernot HASSENPFLUG for corrections to this article.

References

1. The mailing list for the CJK package. http://lists.ffii.org/mailman/listinfo/cjk
2. Ken LUNDE, *CJKV Information processing: Chinese, Japanese, Korean & Vietnamese Computing*, O'Reilly, 1998, ISBN 1-56592-224-2.
3. *Adobe Type 1 Font Format*, Adobe Systems, 1990. http://partners.adobe.com/public/developer/en/font/T1_SPEC.PDF
4. *Adobe CMap and CIDFont Files Specification*, Technical Specification #5014, version 1.0, Adobe Systems, 1996. http://partners.adobe.com/public/developer/en/font/5014.CIDFont_Spec.pdf
5. *Adobe CJKV Character Collections and CMaps for CID-Keyed Fonts*, Technical Note #5094, Adobe Systems, 2004. http://partners.adobe.com/public/developer/en/font/5094.CJK_CID.pdf
6. *OpenType specification*, version 1.4, Microsoft Corporation, 2001. http://www.microsoft.com/typography/otspec/
7. The Unicode Standard. http://unicode.org
8. Werner LEMBERG, *The CJK package for LaTeX2ε — Multilingual support beyond babel*, Proceedings of the 1997 Annual Meeting, TUGboat **18** (1997), no. 3, 214–224. http://www.tug.org/TUGboat/Articles/tb18-3/cjkintro600.pdf
9. PDF *reference*: Adobe portable document format version 1.6. Adobe Systems, 3rd ed., 2005. http://partners.adobe.com/public/developer/en/pdf/PDFReference16.pdf

12. For CJKV languages, the most important element was the introduction of *OTPs* (Ω translation processes) to manipulate input characters before the TeX engine begins to digest them.

10. *Unicode and Glyph Names*, version 2.4, Adobe Systems, 2003. http://www.adobe.com/devnet/opentype/archives/glyph.html

11. Werner LEMBERG and Frédéric LOYER, *The ttf2pk package*. Contains the `ttf2pk` and `ttf2tfm` binaries together with a bundle of SFD files. Currently, the most recent version can be found in TeXLive. URL of `ttf2tfm`'s man page, specifying the SFD syntax: http://www.tug.org/svn/texlive/trunk/Master/texmf/doc/man/man1/ttf2tfm.1. URL of the SFD directory: http://www.tug.org/svn/texlive/trunk/Master/texmf/fonts/sfd/.

12. The Lua programming language. http://www.lua.org