A glance at CJK support with XeTeX and LuaTeX

Antoine Bossard

Abstract

From a typesetting point of view, the Chinese and Japanese writing systems are peculiar in that the characters are concatenated without using spaces to separate them or the meaning units (i.e., "words" in our occidental linguistic terminology) they form. And this is also true for sentences: although they are usually separated with punctuation marks such as periods, spaces remain unused. Conventional typesetting approaches, TeX in our case, thus need to be revised in order to support the languages of the CJK group: Chinese, Japanese and, to a lesser extent, Korean. While more or less complete solutions to this issue can be found, in this article we give and pedagogically discuss a minimalistic implementation of CJK support with the Unicode-capable XeTeX and LuaTeX typesetting systems.

1 Introduction

The Chinese, Japanese and Korean writing systems are conventionally gathered under the CJK appellation. The Chinese writing system consists of the Chinese characters, which can be in simplified or traditional form, amongst other character variants [1]. The (modern) Japanese writing system is made of the Chinese characters and the kana characters. The Chinese and Japanese writing systems concatenate characters without ever separating them with spaces. The Korean writing system consists mainly of hangul characters, in principle together with the Chinese characters, but they are rarely used nowadays. Although modern Korean does separate words with spaces, traditionally, the Korean writing system does not (as an illustration, see, e.g., Sejong the Great's 15th century manuscript Hunminjeongeum[1]).

Notwithstanding other critical issues such as fonts (and to a lesser extent indexing [2]), by not relying on spaces between characters or words, the CJK scripts are a challenge to conventional typesetting solutions such as TeX. In fact, the algorithms for line-breaking, which conventionally occurs at spaces, and for word-breaking (hyphenation), become inapplicable.

On a side note, although we consider hereinafter only the CJK writing systems, this discussion can be extended to related scripts such as Tangut and Vietnam's Chữ Nôm.

[1] King Sejong (世宗) introduced hangul in the *Hunminjeongeum* (訓民正音) manuscript (1443–1446).

In this paper, we provide a glance at CJK support with XeTeX and LuaTeX by giving a minimalistic implementation for these East Asian scripts. This work is both a proof of concept and a pedagogical discussion on how to achieve CJK support as simply as possible with the aforementioned typesetting solutions. Both XeTeX and LuaTeX support Unicode, which enables us to focus on typesetting issues, leaving encoding and font considerations aside.

The rest of this paper is organised as follows. Technical discussion of the proposed implementation is conducted in Section 2. The state of the art and paper contribution are summarised in Section 3. The paper is concluded in Section 4.

2 A minimalistic implementation

We describe here the proposed minimalistic implementation of CJK support with XeTeX and LuaTeX step by step in a pedagogical manner:

- paragraph management (Step 1) is addressed in Section 2.1,
- Latin text mingling (Step 2) in Section 2.2,
- Latin text paragraphs (Step 3) in Section 2.3,
- Korean text paragraphs (Step 4) in Section 2.4,
- sophisticated line-breaking (Step 5) in Section 2.5.

"Latin text" here designates text written with the Latin alphabet, or similar; for instance English and French text.

A handful of TeX commands appear hereinafter without being detailed; see [5] for those that are not self-explanatory. The document preamble specifies nothing in particular. The `fontspec` package [12] is loaded for ease of font manipulation, and, as detailed in the rest of this section, since it is considered without loss of generality that the document consists of Chinese or Japanese paragraphs by default, the main font of the document is set accordingly (e.g., `\setmainfont{Noto Serif CJK JP}` [4]).

2.1 Paragraph management

A conventional approach to break long character sequences (i.e., Chinese or Japanese characters in our case) is to insert between each two glyphs a small amount of horizontal space so that TeX can split the sequence across multiple lines (see for instance [15]). Without such extra space, line breaks can in general still occur thanks to hyphenation, but this is not applicable in the case of CJK. We rely on a "scanner" macro to transform a paragraph by interleaving space between its characters. In practice, according to the TeX terminology, this extra space will be a horizontal skip of 0pt width and ±1pt stretch.

The scanner macro is a recursive process that takes one token (e.g., a character) as single parameter and outputs it with on its right extra horizontal space. The recursion stops when the parameter token is the stop signal (more on this later), in which case the macro outputs \par, thus triggering the end of the paragraph. The scanner macro \cjk@scan is defined as follows:

```
\def\cjk@scan#1{% #1: single token
 \ifx#1\cjk@stop% stop signal detected
  \par% so, complete the paragraph
 \else
  #1% display the current character
  \hskip 0pt plus 1pt minus 1pt\relax% space
  \expandafter\cjk@scan% recursive call
 \fi
}
```

This scanner is started by the \cjk@scanstart macro, whose primary objective is to append the stop signal \cjk@stop at the end of the paragraph that is about to be transformed. This initial macro takes one parameter: the paragraph to transform. In a pattern matching fashion, a paragraph is taken as a whole by setting \par as delimiter for the parameter of the \cjk@scanstart macro. This will require inserting \par once the paragraph has been transformed, since the \par command that ends the paragraph is treated as a delimiter by the macro and thus skipped. In addition, each paragraph needs to be ended by a blank line (or, equivalently, \par) for this pattern matching to work. The scanner starting macro is this:

```
\def\cjk@scanstart#1\par{% #1: paragraph
 \cjk@scan#1\cjk@stop% append \cjk@stop
}
```

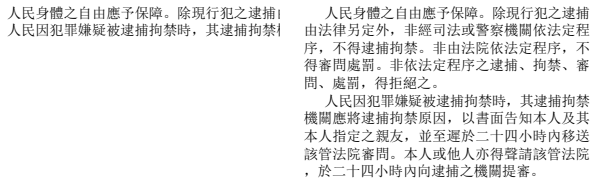
In this work, paragraphs are considered to be written in Chinese or Japanese by default. Hence, paragraph typesetting mode selection by means of a command such as \CHJPtext is not suitable. We rely on the \everypar token parameter to trigger the transformation of each paragraph with the scanner previously described. This is simply done with the following assignment:

```
\everypar={\cjk@scanstart}
```

or, in a safer manner [3]:

```
\everypar=\expandafter{\the\everypar
 \cjk@scanstart}
```

An illustration of the result of this paragraph transformation is given in Figure 1 with two traditional Chinese paragraphs.



(a) (b)

Figure 1: Before (a) and after (b) paragraph transformation: line breaking now enabled (traditional Chinese text example).

2.2 Latin text mingling

It is often the case that Latin text such as English words, expressions or sentences is mingled within Chinese or Japanese paragraphs. In the paragraph transformation method described so far, spaces, if any, are "gobbled" and never passed as parameters to the scanner macro \cjk@scan. This is not a problem for Chinese and Japanese text since, as explained, they do not rely on spaces. But now that we are considering Latin text mingling in such paragraphs, spaces need to be retained since Latin text, such as English, does rely on spaces to separate words, sentences, etc.

Without going too far into the details, to force TEX to also pass spaces as parameters to the scanner macro, spaces need to be made *active*, in TEX terminology. Hence, it suffices to call the \obeyspaces macro, whose purpose is exactly to make the space character active, at the beginning of the document. In addition, the scanner macro is refined to avoid adding extra space when the current character is a space:

```
\def\cjk@scan#1{%
 \ifx#1\cjk@stop
  \par
 \else
  #1%
  \if#1\space% no extra space if #1 is a space
  \else
   \hskip 0pt plus 1pt minus 1pt\relax
  \fi
  \expandafter\cjk@scan
 \fi
}
```

An illustration of the result of this refined paragraph transformation is given in Figure 2.

We conclude this section with the following two remarks. First, it should be noted that Latin text mingled within Chinese or Japanese paragraphs is treated just as Chinese or Japanese text: extra space is inserted between glyphs. Therefore, line- and

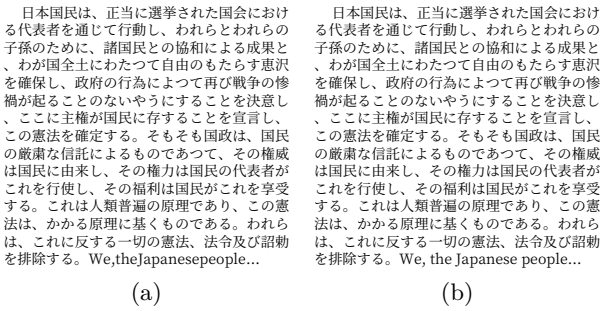


Figure 2: Before (a) and after (b) making spaces active: Latin text mingling now retains spaces (Japanese text example).

word-breaking for mingled Latin text can occur anywhere, and thus no word-breaking by hyphenation will happen. Second, even though no extra space is added after a space character, extra space is still added before a space character. This issue will be tackled in a subsequent section.

2.3 Latin text paragraphs

Because the `\obeyspaces` macro has been called so as to typeset Chinese and Japanese paragraphs, Latin text paragraphs would be typeset just as those, that is, with extra space added between consecutive glyphs (except after spaces). As a result, as explained above, line- and word-breaking would not be satisfactory.

Hence, we next enable the proper typesetting of Latin text paragraphs, that is, paragraphs that include spaces between words. To this end, we define the `\iflatin` conditional statement that will be used to distinguish Latin text paragraphs from others. The flag command `\latinfalse` is called at the beginning of the document to reflect that Chinese and Japanese paragraphs are the norm. Latin text paragraphs are marked as such by calling the flag command `\latintrue` at the beginning of the paragraph. The scanner starting macro `\cjk@scanstart` is adjusted so as to not start the scanner in case the Latin flag is set.

Since the `\obeyspaces` macro has been previously called, spaces are active characters; this setting needs to be reverted in the case of a Latin text paragraph in order to have proper line- and word-breaking. Hence, the scanner starting macro in addition reverts spaces from the active state back to their default state in the case of a Latin text paragraph. The refined code is given next:

```
\newif\iflatin % flag to detect whether to scan
\latinfalse % flag initially set to false
```

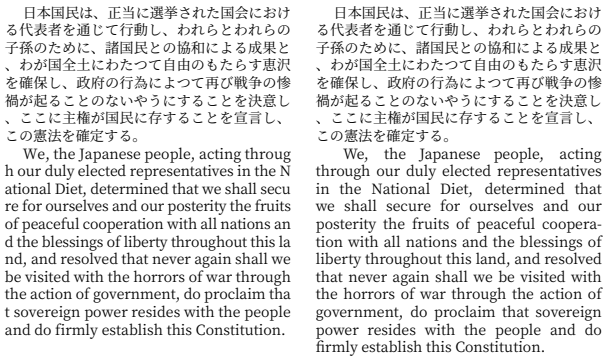


Figure 3: Before (a) and after (b) Latin mode enabling: Latin text now properly typeset (Japanese and English text example).

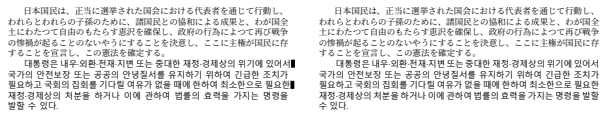
```
\def\cjk@scanstart#1\par{%
 \iflatin% if Latin text paragraph, don't scan
  \catcode`\ =10% revert \obeyspaces
  #1\par% display the paragraph normally
  \latinfalse% back to default
 \else
  \cjk@scan#1\cjk@stop
 \fi
}
```

An illustration of the result of this refined paragraph transformation is given in Figure 3.

2.4 Korean text paragraphs

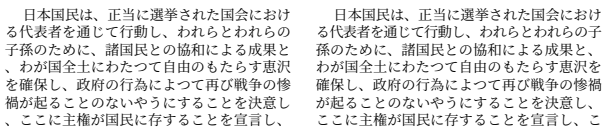
Let us now discuss the case of Korean text paragraph typesetting. As mentioned in the introduction, modern Korean relies on spaces to separate words. Hence, Korean text paragraphs are treated as Latin text paragraphs, concretely marked with the `\latintrue` flag. Yet, because Korean glyphs (i.e., hangul or hanja) are wider than Latin ones, the width of spaces is adjusted. In addition, a font switch is also used to select a Korean font since it is common that Korean glyphs are not included in the default font used for Chinese and Japanese paragraph typesetting.

Such settings need to be applied at the beginning of the paragraph, so we need to embed the paragraph into a group for font selection and the adjusted space setting. Therefore, the paragraph starts with a '`{`' token, and thus it is required to leave vertical mode for proper parsing of the paragraph when it is used as the parameter of our macro `\cjk@scanstart` which starts the scanner. Specifically, the problem with starting the paragraph with a command like `{\malgun` (e.g., a font switch) is that TEX is still in vertical mode when it is pro-



(a) (b)

Figure 4: Before (a) and after (b) space width adjustment for Korean text: no more overfull horizontal boxes (Japanese and Korean text example).



(a) (b)

Figure 5: Paragraph transformation by the original (a) and the new (b) scanner macro: no more line break before a comma (Japanese text example).

cessed. Switching to horizontal mode starts a new paragraph and thus triggers `\everypar`, but then with an unmatched '`}`' remaining (i.e., the one corresponding to, say, the font switch) at the end of the paragraph, and thus the parsing error.

For convenience, these Korean text paragraph settings are gathered in a `\korean{}` macro as defined below.

```
\def\korean#1{%
 \latintrue% activate the Latin mode
 \leavevmode% leave the vertical mode
 {% Adjust the space size:
  \spaceskip=\fontdimen2\font plus
    3\fontdimen3\font minus
    3\fontdimen4\font% ×3 stretch and shrink
  \malgun #1% Korean font switch
 }
}
```

Note that this redefinition of `\spaceskip` for the current paragraph would also be applied to Latin text mingled within a Korean paragraph. Furthermore, this font selection process — without necessarily activating the Latin mode and adjusting the space width — could also be used in the case where distinct fonts for Chinese and Japanese text are required.

An illustration of the result of this paragraph typesetting is given in Figure 4. One should note the overfull horizontal boxes which are shown by the two black boxes in the left-hand example, when the space width adjustment has not been applied yet.

2.5 Sophisticated line-breaking

Just as, say, in French, where line breaks are not allowed before the punctuation marks ':', ';', '!' and so on — even though these need to be preceded by a space and are thus typical usages of non-breaking spaces — CJK typesetting forbids breaking lines before punctuation marks such as commas and periods.

We derive in this section a new scanner macro, `\cjk@scanbis`, to address this remaining problem. The approach is simple: refrain from adding extra space after the current character when the next one is a punctuation mark. At the same time, this new scanner allows us to solve the aforementioned incongruity of extra space being added before a space character in Latin text paragraphs.

To implement this, the new scanner takes two tokens as parameters instead of one: the first parameter is the currently processed token and the second one is the next token in line. The recursive call is also updated since it is now expecting two tokens as parameters instead of one; here it is:

```
\def\cjk@scanbis#1#2{% two tokens passed
 #1%
 \ifx#2\cjk@stop
  \par
 \else
  \if#2、% no extra space before character `、'
  \else\if#2。% idem before character `。'
  \else\if#2\space% idem before a space
  \else\if#1\space% idem after a space
  \else\hskip 0pt plus 1pt minus 1pt\relax
  \fi\fi\fi\fi
  \expandafter\cjk@scanbis\expandafter#2%
 \fi
}
```

Similar additional conditions for other CJK punctuation marks can easily be appended if needed.

One other change is needed: in the scanner macro `\cjk@scanstart`, the initial expression
`\cjk@scan#1\cjk@stop`
is modified to
`\cjk@scanbis#1\cjk@stop`.

An illustration of the effect of this new scanner is shown in Figure 5.

3 State of the art and contribution

Early solutions for supporting the CJK writing systems within the TEX ecosystem include the `CJK` package [6] and the Japanese TEX system pTEX [8]. Although the former provides some support for Unicode, the latter does not. Notably, pTEX supports vertical typesetting [10], while the `CJK` package only partially supports it. Based on the `CJK` package,

the `BXcjkjatype` package [16] provides some support for Japanese typesetting with pdfLaTeX (UTF-8 files). Regarding Korean, the `hlatex` package [14] enables the processing by LaTeX of KS X 1001 encoded files, and of UTF-8 files via the obsolete TeX extension Omega [11]. Omega also has some support for multi-directional CJK typesetting.

More recent solutions include the `xeCJK` package [7], which is dedicated to XeTeX (i.e., no LuaTeX support). This package is very large, consisting of more than 14,000 lines of macro code. As of summer 2019, it is only documented in Chinese. Another extensive package, `luatex-ja` [13], is available, this time restricted to support for Japanese with LuaTeX. Finally, up(La)TeX [9], another system dedicated to Japanese, can also be cited; it is based on p(La)TeX, but unlike its predecessor supports Unicode.

Even if the above are more or less complete solutions to the CJK typesetting issue with TeX, we have presented in this paper a very simple solution, which requires neither a separate TeX system such as pTeX nor advanced TeX capacities such as `xtemplate`, LaTeX3, etc., unlike, for instance, `xeCJK`. With only a few lines of macro code, we have described how to add basic yet arguably competent support for CJK to both XeTeX and LuaTeX, without differentiation. The XeTeX, LuaTeX flexibility has been retained: no extra layer has been piled on as, for instance, with `xeCJK` (e.g., the `\setCJKmainfont` command). Moreover, the complexity induced by packages such as `xeCJK` is likely to be a threat to compatibility with other packages, as well as with online compilation systems such as those employed by scientific publishers.

4 Conclusions

It is well known that the Chinese, Japanese and Korean writing systems are challenging for typesetting programs such as TeX that were originally designed for Latin text. Various extensions and packages have been proposed to support CJK in TeX, with uneven success. Such solutions are in most cases, if not all, extensive — not to say invasive — additions to the TeX ecosystem. In this paper, relying on the Unicode-capable XeTeX and LuaTeX systems, we have presented and pedagogically discussed a minimalistic solution to this CJK typesetting issue. With only a few lines of macro code, we have shown that satisfactory CJK support can be achieved: paragraph management, Latin text mingling and sophisticated line-breaking are examples of the typesetting issues addressed.

As for future work, given its still rather frequent usage, right-to-left horizontal typesetting would be a useful addition to this discussion of CJK typesetting. Furthermore, although it is a complex issue for TeX, right-to-left vertical typesetting is another meaningful objective as it is ubiquitous for the CJK writing systems.

Acknowledgments

The author is grateful to Takeyuki Nagao (Chiba University of Commerce, Japan) and Keiichi Kaneko (Tokyo University of Agriculture and Technology, Japan) for their insightful advice. This research project is partly supported by The Telecommunications Advancement Foundation (Tokyo, Japan).

References

[1] A. Bossard. *Chinese Characters, Deciphered*. Kanagawa University Press, Yokohama, Japan, 2018.

[2] A. Bossard and K. Kaneko. Experimenting with makeindex and Unicode, and deriving kameindex. In *Proceedings of the GuIT meeting 2018, ArsTeXnica 26*, pp. 55–61, Rome, Italy, October 2018. `https://www.guitex.org/home/images/ArsTeXnica/AT026/kameindex.pdf`

[3] S. Checkoway. *The everyhook package*, November 2014. Package documentation. `https://ctan.org/pkg/everyhook` (last accessed August 2019).

[4] Google. Google Noto fonts, 2017. `https://google.com/get/noto` (last accessed August 2019).

[5] D. E. Knuth. *The TeXbook*. Addison-Wesley, Boston, MA, USA, 1986.

[6] W. Lemberg. *CJK*, April 2015. Package documentation. `https://ctan.org/pkg/cjk` (last accessed August 2019).

[7] L. Liu and Q. Lee. *xeCJK 宏包 (in Chinese)*, April 2018. Package documentation. `https://ctan.org/pkg/xecjk` (last accessed August 2019).

[8] K. Nakano, Japanese TeX Development Community, and TTK. *About pLaTeX 2ε*, September 2018. Package documentation. `https://ctan.org/pkg/platex` (last accessed August 2019).

[9] K. Nakano, Japanese TeX Development Community, and TTK. *About upLaTeX 2ε*, April 2018. Package documentation. `https://ctan.org/pkg/uplatex` (last accessed August 2019).

[10] H. Okumura. pTeX and Japanese typesetting. *The Asian Journal of TeX* 2(1):43–51, April 2008. `http://ajt.ktug.org/2008/0201okumura.pdf`

[11] J. Plaice and Y. Haralambous. The latest developments in Ω. *TUGboat* 17(2):181–183, June 1996. https://tug.org/TUGboat/tb17-2/tb51plaice.pdf

[12] W. Robertson. *The fontspec package — Font selection for XeLaTeX and LuaLaTeX*, July 2018. Package documentation. https://ctan.org/pkg/fontspec (last accessed August 2019).

[13] The LuaTeX-ja project team. *The LuaTeX-ja package*, November 2018. Package documentation. https://ctan.org/pkg/luatexja (last accessed August 2019).

[14] K. Un. 한글라텍 길찾이 *(in Korean)*, April 2005. Package documentation. https://ctan.org/pkg/hlatex (last accessed August 2019).

[15] B. Veytsman. *Splitting Long Sequences of Letters (DNA, RNA, Proteins, etc.)*, August 2006. Package documentation. https://ctan.org/pkg/seqsplit (last accessed August 2019).

[16] T. Yato. *BXcjkjatype package*, August 2013. Package documentation. https://ctan.org/pkg/bxcjkjatype (last accessed August 2019).

Permissions

The placeholder text used in the various illustrations of this article is in the public domain as detailed below.

Figure 1: the placeholder text is the two first paragraphs of Article 8 of the Chinese constitution (1947), written in traditional Chinese.

Figure 2: the placeholder text is the first paragraph of the Japanese constitution (1946), followed by the first few words of the corresponding official English translation.

Figure 3: the placeholder text is the first sentence of the first paragraph of the Japanese constitution (1946), followed by the corresponding official English translation.

Figure 4: the placeholder text is the first sentence of the first paragraph of the Japanese constitution (1946), followed by the first paragraph of Article 76 of the South Korean constitution (1988).

Figure 5: the placeholder text is the first sentence of the first paragraph of the Japanese constitution (1946).

◇ Antoine Bossard
Graduate School of Science
Kanagawa University
2946 Tsuchiya, Hiratsuka
Kanagawa 259-1293
Japan
abossard (at) kanagawa-u dot ac dot jp