Poor Man's Japanese in TEX

Thomas Ridgeway
Humanities and Arts Computing Center
University of Washington

This little document is intended as an illustration of and preliminary guide to *poor man's Japanese* [pmJ日本語]. pmJ日本語 is a temporary expedient for printing texts with Japanese via TEX until such time as we have a well-thought-out system (and fonts) for handling Japanese. @

pmJ日本語 has a small number of virtues; among these are:

- pmJ日本語 is available now
- pmJ日本語 is available free of charge
- pmJ日本語 works with standard TEX3.x
- pmJ日本語 is device-independent (although it must be admitted that you will get relatively more pleasing results on less sophisticated printers)
- pmJ日本語 is sufficiently simple-minded you will not need to be a rocket scientist to make modifications for your own use
- pmJ日本語 uses fonts mechanically faked-up from dot matrix quality fonts; if you want you can incorporate new characters of your own design using a bitmap font editor which *may* be easier to use than METAFONT.‡

  For those who might be previously unacquainted with TEX, let us put forth a few of the advantages of TEX which are inherited by pmJ日本語: it can run on almost any modern computer; it can print to any of a large number of printers/typesetters; it can print (if desired and suitably instructed) elaborate formatting of the text; it is available at very low cost, or free (which is an instance of very low cost).

The disadvantages of pmJ日本語 are at least equally compelling:

- pmJ日本語 is crude and unlikely to ever be greatly improved as regards the quality of its Japanese.
- pmJ日本語 uses emulation-in-METAFONT techniques to produce a dot-matrix font on your output device. No matter how talented your printer may be,

@ In JTEX there exists such a system, but for many individuals and institutions the relative cost of fonts/unavailability of free fonts has made installing JTEX not seem advantageous. It is also inconvenient to have to install another separate TEX system for the purpose unless you are doing Japanese in a big way.

‡ Need we point out that the presence of user-defined character codes in a document will make the .tex file non-standard? Such a document will not generate correct output on a system which does not have the user-defined characters defined (!or has *different* user-defined characters defined!).

the Japanese output by `pmJ日本語` will looked like it was stripped in from a dot-matrix printer.

- `pmJ日本語` uses a large number of fonts which occupy a finite and measurable quantity of diskspace for each font. Your screen previewer had *better* be able to use the same resolution your printer uses, or you may experience a little tightness around the disk. You probably will not want the fonts in multiple magsteps (to a not inconsiderable extent they will just get uglier if you enlarge them).
- `pmJ日本語` does not do slanted, bold, or other fancy Japanese although one could in principle. However, insofar as even a minimal installation requires 88 fonts *for Japanese alone*, we must at this point quote The Master:

  > the format of a *char_node* allows for up to 256 different fonts and up to 256 characters per font; ... TEX intended for oriental languages will need even more than 256 × 256 possible characters when we consider different sizes and styles of type.*

- `pmJ日本語` is based on TEX (hooray!), not on JTEX (hooray!) nor any not-known-to-me to-be-extant FooTEX (huh?) which might know how to set type vertically, handle thousands of fonts, make doughnuts, and do all the other things that we might want software to do. `pmJ日本語`, therefore, does not set type vertically, handle thousands of fonts, emulate Spads or Spitfires, or any of the other real neat stuff.
- `pmJ日本語` may take a long time to set up if you need to make your own fonts. Preparing *one* set of fonts on a NeXT computer for use at 400 dpi required a METAFONT run of over 37 hours.† You will not ordinarily wish to make up the fonts more than once, assuming a relative absence of masochism in your

---

* Donald E. Knuth, *TEX: The Program.* Addison Wesley, Reading MA: 1986. (Volume B of *Computers and Typesetting*). §134, p. 57.

To ease somewhat the problem of the number of fonts used, the wjisxx fonts which `pmJ日本語` uses are generated with a \magstep2 variant included in the same font. The enlarged version will be printed any time we have signalled \bigJtrue. I.e., ordinary 殴 versus 殴.

† If one looks at the simple-minded brute-force METAFONT code which the `pmJ日本語` fonts use, one may roughly estimate that a staggering number of computations (probably even more than eight) are needed to emulate a dot-matrix font through this method. There might, indeed, be a better way to do it. On the other hand, Pac-Man worldwide has consumed several hundred million times the number of CPU cycles consumed by compiling `pmJ日本語` fonts, so we are not exactly talking about wasting a scarce resource, are we?

Some of you thought I was going to slam NeXTs here, didn't you? I'm not that kind of guy.

personality makeup, nor will you probably want to keep the METAFONT source code permanently *since you can mechanically regenerate it.*

How to use pmJ日本語

Assume a text editor capable of editing text in Shift-JIS-encoded Japanese characters. Assume a text editor capable of editing text in "normal" Latin characters. Use the first text editor to prepare a plain text file containing the Japanese text; *plain text file* means that there are no tricky codes in the text file which are intended to be used only by the program that produced the file—"ASCII file" frequently conveys what we mean by "plain text file", but by virtue of having encoded Japanese, your file will not be ASCII. Use the second text editor to enter the Latin-letter strings providing instructions to TEX; (if your first text editor is o.k. for Latin too, you can continue using it for this step! — but watch out, the JIS and Shift-JIS encoding schemes have some Latin characters (graphemes) included in the two-byte code space: we need "normal" Latin (one-byte coded) text). On a separate line before the first line of Japanese, put a line saying "\input pmJ \beginJapanese".** pmJ.tex is intended to be the name of a file containing the basic set of instructions for pmJ日本語. "\beginJapanese" is a pmJ日本語-defined magic-word signifying that everything until an "\endJapanese" is to be interpreted as Japanese, if possible. Using whatever text editor you want, incorporate text in whatever other language you want. Other languages will conflict with Japanese in pmJ日本語 only if they use character codes in the range 160 to 254; such languages must be separated from Japanese by enclosing the Japanese in "\beginJapanese" "\endJapanese" structures. Create a new last line saying "\bye". Put a blank line between paragraphs in the text, and use any other TEX commands with which you may be familiar to further adorn your text. Save the file, and say the appropriate words to run TEX on your file, preview it, print it, or whatever.

What can go wrong

Well, for one, you may not have an editor for entering Japanese. [Sorry; pmJ日本語 has nothing to offer you if you have no way to compose your text]. Or, for another, your editor may not use Shift-JIS-encoding. [Easy: convert your text to Shift-JIS-encoding! (How? That's not my problem; or, to be a *little* more gracious about it, there are some converters out there, see if you can find one ...) There is a converter TO_SJIS to translate from NEC, NEW-JIS and OLD-JIS (as described below) to what we want; this converter should be included with pmJ. There are other encoding schemes, however, of which TO_SJIS knows nothing]. You may not

---

** We *really* do need to have the \beginJapanese on a line separate from and preceding the first Japanese character to be printed under its control. Absent strict adherence to this principle all sorts of unJapanese looking things will happen.

know "any other TEX commands with which you may further adorn your text"§ [So take a class]. Something else goes wrong. [Fix it]. You don't have a copy of pmJ日本語. [Get one].

How to get pmJ日本語

From a friend is good. If you have no friend who has pmJ日本語 and is prepared to make a copy for you ... (that was supposed to be phrased as delicately as possible) ... you may get it through an *Established Channel*. There are a number of "TEX distributions" which may elect to incorporate pmJ日本語 in their offerings of TEX and TEX-related software. You will very likely be able to get a copy from the NWCSC, distributors of TEX for Unix systems; they will have to charge you a fee, however, as even though they are non-profit they still have to pay their employees, pay phone bills, and all the rest of the drag with which I am sure you are familiar. You might also be able to get it electronically and free-of-charge via **ftp** from some place or other, but I couldn't tell you where. As I write, November 30, 1990, none of these alternatives which I have suggested will actually work, and there is no way that you can get pmJ日本語 short of enrolling in a course of study at the University of Washington, or you have a friend who has done so, or you make me a really attractive offer.

Are we really going to show any Japanese or what?

Well, o.k.! Included below are extracts from *Electronic Transfer of Japanese* by Ken R. Lunde [klunde@vms.macc.wisc.edu or klunde@wiscmacc]. It discusses some of the varieties of encoding Japanese as alluded to above. If for some reason discernable to neither of us you are reading this and are as unfamiliar with computer representations of Japanese as I am, the text will be informative in its own right, entirely apart from serving as an example of Japanese.

PART 3: DIFFERENCES BETWEEN THE THREE 7-BIT JAPANESE CODES

(KANJI-IN will be abbreviated KI; KANJI-OUT, KO)

	KI	KO (JIS-ROMAN)	KO (ASCII)
NEW-JIS (1983)	<ESC>$B	<ESC>(J	<ESC>(B
OLD-JIS (1978)	<ESC>$@	<ESC>(J	<ESC>(B
NEC CODE	<ESC>K	<ESC>H	n/a

NOTE #1: The difference between KO (JIS-ROMAN) and KO (ASCII) is very minor. Also, most terminals can emulate only one of the KO character sets no matter which one it receives. For example, NINJA TERM (PDS) by Michiharu Ariza from SRA only emulates KO-ASCII.

NOTE #2: The difference between NEW-JIS and OLD-JIS is not significant.

The most commonly used code in Japan for communication is the NEW-JIS. NEC CODE is the least used.

§ Careful study will reveal this phrase to be a direct quotation of nothing except itself.

The KI escape sequence tells Japanese terminals to treat what follows as two-bytes per character. KO, on the other hand, tells Japanese terminals to treat what follows as one-byte per character (back to JIS-ROMAN or ASCII).

A two-byte per character coding system using 7-bit bytes (ASCII) can encode up to 16,384 characters (128 by 128); however, the Japanese only use the 94 printable ASCII codes in their matrix, so it can only encode a maximum of 8,836 characters (94 by 94).

The NEW-JIS Japanese code includes 6,877 standard characters; 6,353 KANJI in 2 levels (level 1: 2,965 KANJI arranged by ON reading; level 2: 3,388 KANJI arranged by radical), 86 KATAKANA, 83 HIRAGANA, 10 numerals, 52 English characters, 147 symbols, 66 Russian characters, 48 Greek characters, and 32 line elements (for making charts). This was established in 1983.

PART 4: 7-BIT JAPANESE CODE REPRESENTATIONS

(This section contains Japanese text which can be read using appropriate software and hardware.)

```
JAPANESE TEXT: これは日本語の文章です。
(ROMANIZED: KO RE WA NI HON GO NO BUN SHOU DE SU .
RAW CODE (NO ESCAPE CODES) $3$1$OF|K\81$NJ8>O$9!#
NEW-JIS: <ESC>$B$3$1$OF|K\81$NJ8>O$G$9!#<ESC>(J
OLD-JIS: <ESC>$@$3$1$OF|K\81$NJ8>O$G$9!#<ESC>(J
NEC CODE: <ESC>K$3$1$OF|K\81$NJ8>O$G$9!#<ESC>H
```

Notice the correspondences between the ASCII characters and Japanese characters, namely that two ASCII characters represent one Japanese character; hence, Japanese characters consist of two bytes. For example, the HIRAGANA sysmbol for “RE” is represented by the two ASCII characters “$” and “l”.

The following paragraphs are Japanese text represented in each of the three major Japanese codes. If you are using a Japanese terminal, then you can view the Japanese text using the procedures found in later sections of this report.

NEW-JIS: (KANJI-IN: “<ESC>$B”; KANJI-OUT: “<ESC>(J”)

日本語の一文字を７ビット×２バイトのコードで表現する方法には新ＪＩＳ、旧ＪＩＳ、ＮＥＣ漢字の三種類があります。これらのコードを用いた文章では日本語の前後に漢字イン、漢字アウトという二つのエスケープ・シーケンスを使用することにより、その中が日本語であることを示します。

OLD-JIS: (KANJI-IN: “<ESC>$@”; KANJI-OUT: “<ESC>(J”)

日本語の一文字を７ビット×２バイトのコードで表現する方法には新ＪＩＳ、旧ＪＩＳ、ＮＥＣ漢字の三種類があります。これらのコードを用いた文章では日本語の前後に漢字イン、漢字アウトという二つのエスケープ・シーケンスを使用することにより、その中が日本語であることを示します。

NEC CODE: (KANJI-IN: “<ESC>K”; KANJI-OUT: “<ESC>H”)

日本語の一文字を７ビット×２バイトのコードで表現する方法には新ＪＩＳ、旧ＪＩＳ、ＮＥＣ漢字の三種類があります。これらのコードを用いた文章では日本語の前後に漢字イン、漢字アウトという二つのエスケープ・シーケンスを使用するこ とにより、その中が日本語であることを示します。

PART 5: 8-BIT JAPANESE CODES

These codes cannot be used reliably in the US since 7-bit paths will strip off the 8th bit leaving garbage. These codes are used primarily for internal processing of Japanese. The names in the parentheses below are other names for the same code (i.e., SHIFT-JIS is also called MS-KANJI).

EXAMPLES: SHIFT-JIS (MS KANJI) EUC (AT&T JIS)

SHIFT-JIS:

The two-byte 8-bit Japanese code's implementation is quite unlike that for the two-byte 7-bit code as was described above. I will explain how the two-byte 8-bit code works for the sake of completeness. The two-byte-per- character mode is initiated when the Japanese terminal receives a character from the ASCII 8-bit extension (i.e., any printable from 128-255). This character will be treated as the first byte of an expected two-byte sequence. The following character, which may be ANY printable ASCII character (including those in the 8-bit extension), is treated as the second byte to complete the two-byte sequences. In summary, the first byte in the two-byte 8-bit Japanese code must be a character which is part of the 8-bit extension (value 128-255), and the second byte can be ANY character. The main difference is that the two-byte 8-bit Japanese code does not use KANJI-IN or KANJI-OUT escape sequences to shift to and from the two-byte-per-character mode; the two-byte 7-bit code as was discussed above does implement KANJI-IN and KANJI-OUT escape sequences.

[End of Lunde extract.]

If a little is good, let's try some more!

This text is "JTeX Note" from the NTT JTeX distribution.

桜井貴文

TeX では <return> と <space> の扱いについて次の様な規則があります。

1. 空白行でない行の間の<return>は<space>とみなす。

2. 2つ以上の<space>は1つの<space>と同じ。 これだけ知っていればほぼ十分なのですが、空白行(<space>のみからなる 行も含む)も考慮してもう少し厳密に述べると、次の順に処理されます。

1. 行末の<space>は無視する。

2. 次の文字も<return>である<return>以外の<return>は<space>とみなす。

3. 2つ以上の<space>は1つの<space>と同じ。 これから派生規則として

4. 行頭の <space> は無視する。

しかし、 JTeX の場合漢字の前後の<return>を無視したいので、上記の 規則は以下の様に変更されています。

1. 行末の<space>は無視する。

2. 次の文字も <return> である <return> 以外の <return> のうち、 前後に漢字がある場合はその <return> は無視し、 そうでない場合はその <return> は <space> とみなす。

3. 2つ以上の <space> は1つの <space> と同じ。 従って、派生規則4は必ずしも成り立たない事に注意して下さい。

ただし、この変更は端末からテキストが入力されるときやファイル から \read で読むときには適用されません。従って、大抵の場合には 適用されることになります。

なお、この様に変更した結果、行の最後でエラーが起きた場合 (例えばその行を10行とすると)、 TeX では必ず

l.10 ...xxx xxx xxx

と表示されますが、 JTeX では

l.11

yy yyy yyyy

と表示される事があります。これは11行の先頭の前、即ち10行の最後で エラーが起きた、と読めるので表示が間違っているというわけでは ありません。