English Notes on Typesetting Japanese with pTEX

ティム　エール　ptex(at)nihilist.org.uk

平成二十二年三月

Contents

1 Introduction

The program pTEX from ASCII Media Works is an effective tool for typesetting Japanese. Unfortunately I've never been able to find much in the way of English documentation for pTEX. This document gathers together the knowledge I've accumulated on pTEX through web-searching, inspired guesses, hair-pulling, inspecting code and doing my best to make sense of the Japanese documentation.

In this document I assume you are using Microsoft Windows. If you use Linux then you will be sufficiently computer-literate to apply what is written here to your enviroment. Macintosh users might also find some of the information here useful. I

have tried the Macintosh distribution of pTEX and it works well. I also assume that you are familiar with using the MS-DOS command-line interface and basic tools like gzip and tar.

2 Changelog

2010/03/25: Corrected capitalization of PostScript, corrected explaination of `H` for CMap resources, clarified Unicode's status as a character set rather than an encoding, corrected reference to Ken Lunde's book *CJKV Information Processing*, corrected use of term 'furigana' to the typographic term 'ruby'. [All these corrections were kindly pointed out by Ken Lunde]. Changed typographical presentation of names like Tex to use more typographically elaborate names like TEX. Removed dead links. Updated link to W32TeX site. Removed 'New Material' section. Various minor adjustments.
2006/11/05: Added note on ruby in plain non-pTEX. Tidied up the ruby section. Added a note on jTEX.
2006/10/29: Added note on ruby in plain pTEX.
2006/10/21: Added notes on *warichu*. Changed typewriter font so that the document looks prettier in Adobe Reader.
2006/06/08: Added notes on: Linux installation of pTEX; XƎTEX.
2005/11/20: Added notes on: use of alternative kanji fonts with LATEX font selection scheme; suppressing font shape warnings; 教科書 (schoolbook) fonts.
2005/11/11: Initial revision

3 Acquiring and installing pTEX

Point your web browser at `www.w32tex.org`[1]. This is the download site for W32TeX. There is an English version of the page. A good thing about this installation is that the maintainer updates it every few days. Download all the packages from the Basic and Standard Installation sections. If you fancy any of the packages in the Full Installation section then download those too.

One of the things I like about W32TeX is that the packages are just gzipped tar files. The installation includes an installer but you can just `gunzip` all the files and `tar -xvf` them yourself. My W32TeX installation takes up about 250MB of disk space.

Once you've done this you'll need to add `c:\usr\local\bin` to your path and modify the `texmf.cnf` file to reflect your system. Of course, the details of this are outside the scope of this document.

If you are reading this document then you are probably already familiar with TEX and therefore probably already have a TEX system installed. If you can get pTEX to work on your existing installation then I'm happy for you. I never managed to do it. For a while I had a TEX Live installation running alongside a pTEX installation. This worked fine; I just had to change my path when I was using Japanese so that I picked up the W32TeX binaries instead of the TEX Live binaries. Eventually I migrated to W32TeX. W32TeX is what most Japanese people seem to use. The good thing about W32TeX is that it handles Japanese without needing any extra configuration.

As an aside, pLATEX seems to be more popular in Japan than LATEX is in the West. In Japan I see a few books on pLATEX in most larger book shops.

3.1 Installation on Linux

You need to find the extra packages for your distribution that include pTEX. You'll also need Adobe Reader or xpdf with the Japanese support package; a Japanese font, such as `kochi-mincho.ttf`; and dvipdmx. Once all these are installed you can compile documents that are in Shift-JIS format by running

```
ptex -kanji=sjis myfile.sjs
```

This will probably work. However, when you `dvipdfmx myfile.dvi` you'll probably get a failure.

I fixed this by copying the contents of the `cmap` directory in my W32TeX installation to my Linux installation. Then I updated `texmf.cnf` via

```
/etc/texmf.d/<somefile>
```

and

```
update-texmf
```

(this seems to be a feature of teTEX) to point at the directory in my installation that contains my TrueType fonts (`/usr/share/fonts/truetype//`). Finally I updated `x-cid.map` to add the line

```
rml H kochi-mincho
```

`H` refers to something called a CMap resource. You'll find it in the `cmap` directory you copied over.

4 Entering Japanese text

4.1 Encodings

In the world of computers all data is stored as numbers. You will already know that the characters a–z, A–Z, 1–9 and some punctuation marks are represented by numbers between 0 and 127. The number used to represent each character is defined by the ASCII standard[1]. Because we only need the numbers between 0 and 127 to represent plain English we can store each character in an English text file as a byte. Languages such as French and Czech that include accented characters can also be represented by text files that use just one byte for each character. However, to represent the accented characters they also make use of the numbers between 128 and 255. You may already know that there is no one standard for the characters represented by the numbers between 128 and 255; the character that is represented by one of these numbers is defined by the *encoding* that is being used.

This state of affairs is reflected in the development of TEX. When Knuth first released TEX each font had 128 character slots. A later version gave each font 256 character slots, thus enabling people to use the full width of a byte to represent character.

Japanese has far more than 256 characters. Therefore we need to use bigger numbers to represent the characters. This is typically achived by using multiple bytes to represent each character. Using two bytes provides us with 65,536 slots to put characters in. This is enough even for Japanese. However, the details of representing complex writing systems in computers is rather more complex than this. Full details are beyond the scope of this article and can be found in [2].

One might think that using multiple bytes to represent the world's most bewildering writing system is complex enough. However, as is usually the case with software, we have another layer of complexity: there is no one standard encoding for representing Japanese characters. The most common ones are Shift-JIS, ISO-2022-JP ('JIS'), EUC-JP and the various encodings of the Unicode character set such as UTF-8, UTF-16 and UTF-32.

Unicode and its encodings are the closest we have to an industry-wide standard for encoding the written word. A disadvantage of the UTF-16 and UTF-32 encodings of Unicode is that if you send Japanese text encoded in one of these formats to a destination that can read ASCII but not Unicode then the recipient cannot read any of the text. This would be particularly unfortunate if there were only a few Japanese characters in the message. This is the advantage of the UTF-8 encoding; it keeps all the ASCII characters as single bytes of ASCII. The disadvantage of UTF-8 is that it

[1] IBM mainframes use a character encoding called EBCDIC, which does not represent consecutive letters by consecutive numbers. I've never seen EBCDIC used with TEX.

uses significantly more bytes than UTF-16 to encode the same number of non-ASCII characters. UTF-32 is inefficient in its use of space and is rarely used.

The JIS encoding was devised in Japan; it stands for *Japanese Industrial Standard*. This system has the same disadvantage as Unicode in that Western characters will not survive if the JIS text is displayed on a JIS-incapable device. Thus Shift-JIS (sometimes written S-JIS or SJIS) was devised. Confusingly, it was devised by a Japanese company called ASCII Media Works in collaboration with Microsoft. Microsoft adopted this encoding (in a slightly modified form) so it is widely used. ASCII Media Works also produced pTEX so, not surprisingly, the native encoding of pTEX is Shift-JIS. I always use Shift-JIS unless I have a good reason to do otherwise.

I don't know anything about the EUC-JP encoding except that it tends to be used on UNIX systems. For information on the EUC-JP encoding and comprehensive information on handling Chinese, Japanese, Korean and Vietnamese on computers see [2].

4.2 JWPce

Western versions of Microsoft Windows XP and above include a Japanese text entry system; you just need to fiddle with the settings in the Control Panel to get it working. It works with Notepad and if you're lucky it might work with your favourite text editor. Powerful though this input method is, it is more aimed at native Japanese speakers than students of the language.

Much better for people like me is JWPCE[3]. It's a free download and comes with plenty of help and documentation. Features that are useful for students include the built-in dictionary and the built-in kanji information look-up. It also has three Japanese fonts built into it.

Download JWPce from [3] and install it. When you save your TEX source use the Shift-JIS encoding (`.sjs`).

4.3 Adobe Reader

You will probably want to view your pTEX creations as PDF files. If you don't already have Adobe Reader installed, install the latest version. Also install the Japanese language pack. The fonts included in this language pack are enough to get you going with pTEX. You don't even need the Windows Japanese fonts.

In Japan people seem to use a DVI previewer called DVIout. It is also possible to run a Japanese-enabled version of dvips (see below) and view the results using Ghostview.

4.4 Japanese Fonts 日本語の字体

If you want to view your pTEX output as PostScript then you will need to install the Windows Japanese fonts. You can do this from the Control Panel. The fonts are called msmincho.ttc and msgothic.ttc. Yes, it is counter-intuitive to need TrueType fonts to view a PostScript document.

5 Other Japanese-Capable TEX Systems

There is a LATEX package called CJK that provides another way to typeset Japanese text in TEX. It allows you to typeset Korean and Chinese as well as Japanese. It has documentation in English.

The future of polyglot TEX typesetting appears to lie with XETEX. This system has now been ported from the Macintosh OS X platform to both Linux and Windows. The Windows installation is done as a bolt-on to W32TeX; the W32TeX download site includes a binary package and English installation instructions. I have got both these systems up and running. The Windows installation took a matter of minutes. XETEX is now also available with TEX Live.

There is a Japanese version of a program called Omega, a version of TEX that can handle 16-bit encodings. There seems to be little activity or documentation on this project.

jTEX is an early (c.1987) Japanese-enabled TEX variant created by NTT. It is still available for download but has been largely superseded by pTEX. An article on the development of this package has been published in TUGboat[4].

The UMS package allows you to put Japanese text in a file that is to be compiled by pdfTEX or pdfLATEX. You use it by producing a Shift-JIS source file, running this file through a program called topdftex and the sending the result to pdfLATEX with the UMS package included. One reason for doing this rather than using pTEX and dvipdfmx is that you might want to use some feature that is specific to pdfTEX.

A sample input file is as follows:

```
\documentclass[12pt]{article}
\usepackage{ums}
\begin{document}
私は魚に興味があります。
\end{document}
```

The commands you need to run to obtain a PDF document from a shift-JIS format file using pdfLATEXare as follows.

```
topdftex source.sjs tmp.sjs
```

```
pdflatex tmp.sjs
```

When you run topdftex, the resulting file tmp.sjs should look like this:

```
\documentclass[12pt]{article}
\usepackage{ums}
\begin{document}
\UMS{79C1}\UMS{306F}\UMS{9B5A}\UMS{306B}\UMS{8208}...
\end{document}
```

To set up the UMS package you need to run the batch jobs in the following two directories

```
C:\usr\local\share\texmf\fonts\type1\public\omegaj\msmin
C:\usr\local\share\texmf\fonts\type1\public\omegaj\msgoth
```

to create all the `.pfb` files. This in turn requires you to intall the W32TeX Omega packages.

pTEX is the most popular solution in Japan and, as such, has plenty of Japanese-specific macros available. Judging from the questions on the TEX newsgroup, `comp.text.tex` the CJK package is the most popular solution outside Japan. XƎTEX describes itself as experimental software whereas pTEX has had many years of field hardening. Both the CJK package and the XƎTEX system support writing systems other than Japanese whereas pTEX only supports Japanese in addition to those supported by ordinary TEX.

6 Creating a document

6.1 Plain pTEX

A remarkable feature of pTEX is that you can enter Japanese text in-line with Western text without any extra markup. pTEX handles all the font switching internally. Here is a simple document in plain pTEX. Save the file in Shift-JIS (`.sjs`) format.

```
The Japanese symbol for fish is 魚.
\bye
```

6.2 pLATEX

Using pLATEX is no more complex; again pLATEX handles everything for you. The only difference is that if your document is intended to be read as being mainly Japanese you should use

```
\documentstyle{jarticle}
```

instead of

```
\documentstyle{article}
```

This makes the output caption figures with 図 instead of *Figure* and so on. Here is a simple example:

```
\documentclass{jarticle}
\begin{document}
鯨は魚ではありません。
\end{document}
```

7 Viewing documents

Once you have written your pTEX or pLATEX source file you compile it in the obvious way:

```
c:\work>ptex my_document.sjs
```

or

```
c:\work>platex my_platex_document.sjs
```

The resulting file is called `my_document.dvi`. However, the file format is not standard DVI so the standard versions of dvips and dvipdfm will not be able to convert it into a viewable format. Because of this pTEX is not strictly speaking a version of TEX at all. pTEX does not pass the `trip.tex` test either[5]; this disqualifies it from being a true TEX. However, you are unlikely to notice any problems in practice.

7.1 dvipdfmx

To convert your `.dvi` file into PDF format, run it through dvipdfmx. This program comes with the W32TeX installation and does not need any configuration. Run the following two commands and, assuming you have bound `.pdf` files to Adobe Reader, your document should appear on the screen.

```
c:\work>dvipdfmx my_platex_document
c:\work>start my_platex_document.pdf
```

7.2 dvipsv

The dvipsv program is a version of dvips enhanced to handle the pTEX `.dvi` format and embed the TrueType fonts in the document. If you want PostScript output then this is probably the one to use. It produces large output files because of the embedding. Obviously, you need Ghostscript and Ghostview installed to view the output.

```
c:\work>dvipsv my_platex_document
c:\work>start my_platex_document.ps
```

7.3 dvipsk

There is a bit of naming convention confusion here. Radical Eye now call dvips dvipsk. However W32TeX calls the executable for the standard, non-pTEX version of dvipsk `dvips.exe`. The executable for the version of dvipsk that can handle pTEX output is called `dvipsk.exe`.

The advantage `dvipsk.exe` has over `dvipsv.exe` is that it produces smaller output files and runs more quickly. The disadvantage is that it does not embed the fonts in the output so you need to have the fonts installed on the system where you are going to view the PostScript file. Furthermore, if you install a new Japanese font on your system then you need to modify your Ghostscript configuration files before you can view your new document. This is covered in detail in a later section.

W32TeX also includes a program called `udvips`. It appears to produce output identical to `dvipsk`.

8 PDF bookmark entries

You have to do a bit of extra work to get PDF bookmarks to work in Japanese script. The PDF special `tounicode` is the key. For plain pTEX the source would look like this:

```
\def\bookmark#1{\special{pdf: out 1 << /Title (#1) /Dest
                              [ @thispage /FitH @ypos ] >>}}%
\special{pdf:tounicode 90ms-RKSJ-UCS2}
\bookmark{日本語 1}
\bye
```

and in in pLATEX

```
\documentclass{jarticle}
```

```
\def\bookmark#1{\special{pdf: out 1 << /Title (#1) /Dest
             [ @thispage /FitH @ypos ] >>}}%
\AtBeginDvi{\special{pdf:tounicode 90ms-RKSJ-UCS2}}
\begin{document}
\bookmark{日本語 1}
Hello
\end{document}
```

This technique works for annotations (sticky notes) in dvipdfm too. For plain pTEX source it would look like this

```
\special{pdf:tounicode 90ms-RKSJ-UCS2}
話話話話話話話話
\special{pdf: ann width 3.0in height 36pt
<< /Type /Annot /Subtype /Text
/Contents (日本語) >>}
Blah blah blah
\bye
```

There should be an annotation just here on this page.

Determning whether `tounicode` works for other dvipdfmx contructs too is left as an exercise for the reader. The following standard hyperref package code placed in the preamble to a document will produce PDF bookmark entries:

```
\special{pdf:tounicode 90ms-RKSJ-UCS2}
\usepackage[dvipdfm,bookmarks=true,%
bookmarksnumbered=true,bookmarkstype=toc,%
colorlinks,linkcolor=blue,urlcolor=blue]{hyperref}
```

9 Installing new kanji fonts

The default installation of W32TeX appears to use MS Mincho and MS Gothic as its only fonts. However, if you use dvipdfmx you actually see the Adobe Reader fonts; dvipdfmx writes the PDF file specifying the Adobe Reader fonts as substitutes. To get the real MS Mincho and MS Gothic fonts you need to run

```
dvipdfmx -f msembed.map file.dvi
```

This results in a larger PDF file because the font is now embedded within it.

When I first started using pTEX I was grateful to be able to typeset Japanese at all; it seemed greedy to want to use other fonts. However, after using pTEX for a while you might want to use a completely different font. It is possible to install new Japanese TrueType fonts into W32TeX. This section explains how.

9.1 Available fonts

There are dozens of free Kanji fonts out there. Do a web search to find them. Epson in particular have a bundle of several Japanese fonts that they give away. Try searching for `epkyouka.ttf`. One of the most famous free TrueType fonts comes from Netscape and is called Cyberbit.

If you are learning kanji then it's worth looking at the kyoukasho 教科書(きょうかしょ) fonts. These are the Japanese equivalent of the Western 'Schoolbook' fonts and are designed explicitly for teaching Japanese. A Japanese calligraphy teacher recommended the commercial Iwata Gakusen Kyoukasho (Gーイワタ中太教科書体) font to me. This font costs about 12,000 円.

9.2 Installing into pTEX

First install the font in windows. Let's call it epkyouka.ttf. You should have a file called `c:\windows\fonts\epkyouka.ttf` on your system. In your local `texmf` tree (such as `c:\work\texmf}` copy
`fonts\tfm\dvips\rml.tfm` to `fonts\tfm\dvips\epk.tfm`
copy
`fonts\tfm\ptex\min10.tfm` to `fonts\tfm\ptex\schoolbook.tfm`
and copy
`fonts\vf\ptex\min10.vf` to `fonts\vf\ptex\schoolbook.vf`.

Open `fonts\vf\ptex\schoolbook.vf` in a text editor and change the three letters `rml` to `epk`.

What we've done here is to create the `.tfm` files that pTEX uses for a new font and to create a virtual font so we can view it.

Run `mktexlsr` (or equivalent) so that KPSE knows about your new files. You should now be able to run a file like the following through pTEX.

```
\font\schlbk=schoolbook at 12pt
\tenmin 鮭は魚です。

\schlbk 鮭は魚です。
\bye
```

9.3 dvpdfmx

These metric files are not much use unless you can view the output. To do this you must tell `dvipdfmx` about the new font. The best way to do this is to modify `msembed.map`. Copy it into your local `texmf` tree and add the following line

```
epk  H  :0:epkyouka
```

Run `mkstexlsr` again and then do

```
dvipdfmx -f msmebed.map test.dvi
```

You should get the PDF file. When you open it with Adobe Reader, the document properties should tell you that the MS Mincho and Epson Kyoukasho fonts are both present in the document.

There are some advanced options you can put in the msembed.map file. For example

```
epk  H  :0:epkyouka,Bold
```

gives you a bold version of the font. BoldItalic and Italic are also valid keywords here. My experiments indicate that this doesn't work very well; the fonts appear in the modified form in Adobe Reader but do not come out on the printer. This is no great loss; ransom-note typography is best left alone. Some TrueType fonts contain multiple versions of themselves in the same file. You can access the different versions by changing the number between the colons:

```
xyz  H  :1:complexfont
```

The `H` refers to whether the font is for horizontal or vertical typesetting. I haven't tried installing a vertical version of a font.

9.4 dvips

It is also possible to use TrueType kanji fonts with dvipsv and dvipsk.

For dvipsv, locate `psfontsv.map`, take a copy into your local `texmf` tree and add the following line.

```
ekn       r-epson-kyoukasho              <‘r-epson-kyoukasho
```

Next locate `vfontcap` in the main `texmf` tree, save off a backup and modify it where it is (KPSE doesn't seem to find it if I put it in my local `texmf` tree) to add the following lines.

```
r-epson-kyoukasho:\
:ft=freetype:\
:ff=c\:/windows/fonts/epkyouka.ttf:
```

Run `mktexlsr` and then `dvipsv test.dvi` and you should get a PostScript version of your document.

If you want a PostScript file that does not embed the kanji font then you can also configure `dvipsk` to use a new TrueType font. First update `psfonts.map` to include the line

```
ekn          epson-kyoukasho-H
```

Then update the file `cidfmap` in your Ghostscript installation (try looking for `c:\gs\gs8.51\lib\cidfmap`) to include the following line (split into two lines here so it will fit on the page)

```
/epson-kyoukasho << /FileType /TrueType /SubfontID 0 /CSI
    [(Japan1) 3] /Path (C:/WINDOWS/fonts/epkyouka.ttf) >> ;
```

I find that documents created this way take a long time to open in Ghostview. Furthermore, one document with dozens of different fonts in it that I tried crashed Ghostscript. Therefore I can't recommend this method.

10 Vertical typesetting

Traditionally Japanese is written from top to bottom and from right to left. One of the strengths of pTEX is that it has native support for this format.

To typeset a document vertically in plain pTEX use `\tate` at the start of the document and declare the font you want to use (`\tentmin` is the only one I know works):

```
\tate\tentmin
私はイギリス人です。
\bye
```

Then convert the `.dvi` file to a landscape PDF as follows

```
dvipdfmx -l sample
```

In Japanese *tate* 縦(たて) means *vertical*.

As you would expect from plain TEX, the rest of the document formatting needs work before you can use this method for a real document. However, using pLATEX you get everything done for you. All you have to do is change the

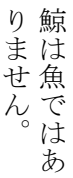

Figure 1: Vertical Japanese.

```
\documentstlye{jarticle}
```

in the preamble to

```
\documentstlye{tarticle}
```

For example

```
\documentclass{tarticle}
\begin{document}
鯨は魚ではありません。
\end{document}
```

gives you something like Figure 1.
Again you need to convert the `.dvi` file to a landscape PDF as follows

```
dvipdfmx -l sample
```

That's right, you can take your horizontally-orientated document and convert it to vertical format by changing just one character.

11 Ruby

Ruby is the typographical name for furigana. These are small kana characters written near a kanji to clarify its reading, like this: 魚(さかな).

11.1 Ruby in pLaTeX

To use ruby in your pLaTeX document include

```
\usepackage{sfkanbun}
\usepackage{furikana}
```

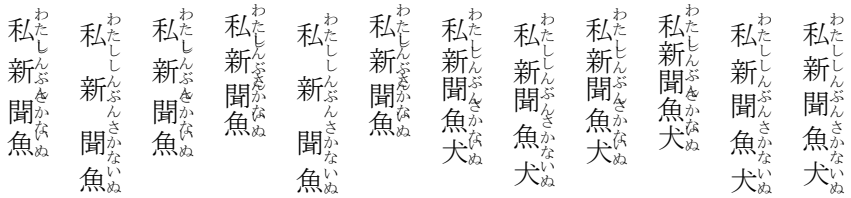

Figure 2: Demonstration of ruby macro syntax

in your pLaTeX document's preamble. There are two macros and a variety of options. The following code yields the example in Figure 2.

```
\documentclass[12pt]{tarticle}
\usepackage{sfkanbun}
\usepackage{furikana}
\begin{document}
\kana{私新聞魚犬}{わたししんぶんさかないぬ}\par
\kana[0]{私}{わたし}\kana[0]{新聞}{しんぶん}\kana[0]{魚}{さかな}%
\kana[0]{犬}{いぬ}\par
\kana[1]{私}{わたし}\kana[1]{新聞}{しんぶん}\kana[1]{魚}{さかな}%
\kana[1]{犬}{いぬ}\par
\kana[2]{私}{わたし}\kana[2]{新聞}{しんぶん}\kana[2]{魚}{さかな}%
\kana[2]{犬}{いぬ}\par
\kana[3]{私}{わたし}\kana[3]{新聞}{しんぶん}\kana[3]{魚}{さかな}%
\kana[3]{犬}{いぬ}\par
\kana[4]{私}{わたし}\kana[4]{新聞}{しんぶん}\kana[4]{魚}{さかな}%
\kana[4]{犬}{いぬ}\par
\Kana{私, 新, 聞, 魚, 犬}{わたし, しんぶん, さかな, いぬ}\par
\Kana[0]{私, 新, 聞, 魚, 犬}{わたし, しんぶん, さかな, いぬ}\par
\Kana[1]{私, 新, 聞, 魚, 犬}{わたし, しんぶん, さかな, いぬ}\par
\Kana[2]{私, 新, 聞, 魚, 犬}{わたし, しんぶん, さかな, いぬ}\par
\Kana[3]{私, 新, 聞, 魚, 犬}{わたし, しんぶん, さかな, いぬ}\par
\Kana[4]{私, 新, 聞, 魚, 犬}{わたし, しんぶん, さかな, いぬ}\par
\end{document}
```

11.2 Ruby in plain pTeX

It is possible to modify the file furikana.sty to allow you use its ruby macros in plain pTeX. Here is a summary of what you need to do (I don't recommended this for

beginners):
– Copy `furikana.sty` to `plain_furikana.tex`. Do the edits that follow on the latter file;
– Remove the lines that start `\typeout`;
– Remove the paragraph before the line that reads `\let\rubykatuji=\tiny`;
– Change `\@s@sf` to `\asasf` throughout;
– Remove the `\kana` macro;
– Change the `\k@na@` macro so that it always takes five parameters and rename it to `\kana`. Parameter `#1` is the ruby style, parameters `#4` and `#5` define the font and size used for the ruby. Add

```
\font\tiny=#4 at #5pt\def\@rubykatuji{\tiny}\def\rubykatuji{\tiny}
```

to the macro after the line `\xkanjiskip=0pt`; and
– Remove `\endinput` at the end of the file.
You can then use ruby in a plain pTEX document by adding

```
\input plain_furikana.tex
```

near the start of the document and entering things like

```
\kana{1}{私}{わたし}{epkyo}{6}
```

This works in both horizontal and vertical modes. You may want to define your own two-parameter macro that sets the the other parameters automatically. `\Kana` remains a pLATEX-only macro.

11.3 Ruby in plain non-pTEX

The plain pTEX version of the `furikana.sty` macro described in the previous section requires that you use pTEX. The following macro allows users to use ruby in any version of TEX. An obvious application is to typeset ruby when using XETEX. The `furikana.sty` macro does not work in pLATEX `\section{}` headings, so this macro could also be useful when typesetting with pLATEX. However, this macro results in corrupted PDF bookmarks when combined with the `\hyperref` package. This macro could even be used with jTEX or plain TEX (fonts permitting).

```
\font\tinyjapanese=min10 at 6pt%
\def\furigana#1#2{\leavevmode%
\setbox0=\hbox{#1}\setbox1=\hbox{\tinyjapanese#2}%
\ifdim\wd0>\wd1\dimen0=\wd0\else\dimen0=\wd1\fi%
\hbox{\vbox{\hbox to\dimen0{\tinyjapanese\hfil#2\hfil}%
\nointerlineskip\hbox to\dimen0{\hfil#1\hfil}}}}
```

The `furikana.sty` macros produce more elegant output and provide more formatting flexibility than this macro:

No ruby: 日本で働いた
`furikana.sty`: 日本(にほん)で働(はたら)いた
`\furigana{}{}`: 日本(にほん)で 働(はたら) いた

You are likely to want to tweak the macro described in this section to suit your particular application.

12 Circled characters

Japanese text (especially reference works) sometimes makes use of characters with circles around them. The PSTricks macros `\psCirclebox` and `\pscirclebox` work well for producing Japanese characters with circles around them. The disadvantage is that you then have to produce your document as a Postscript file rather than a PDF file. The best way to handle this is to use Ghostscript to convert your document to PDF like this:

```
gswin32c -sDEVICE=pdfwrite -sOutputFile=circle.pdf
                           -dNOPAUSE -dBATCH -q circle.ps
```

The output looks bad in Adobe Reader but looks fine when you print it.

13 dvipdfmx and PSTricks effects

The fancy text colouring and rotation dvipdfm(x) and PSTricks provide work just as well with Japanese text as they do with Western text. The following sample code produces the garish example in Figure 3

```
\documentclass[11pt]{article}
\usepackage{pstricks}
\usepackage{pst-grad}
\usepackage{pst-plot}
\usepackage{pst-text}
\usepackage{pst-char}
\begin{document}
\begin{pspicture}(0,-1)(8,2)
\pscharpath[linecolor=Yellow,%
fillstyle=gradient,%
gradbegin=Yellow,%
```

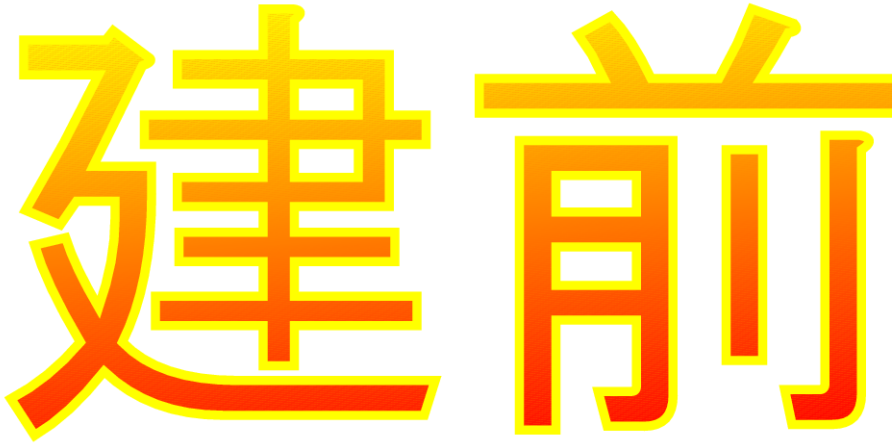

Figure 3: Using PSTricks on Kanji.

```
gradend=Red,%
gradmidpoint=1,%
gradangle=5]%
{\font\tmp=goth10 at 1.5cm\tmp 建前}
\end{pspicture}
\end{document}
```

14 Context

I have not been able to persuade pTEX to work with the ConTEXt macro package. However, ConTEXt does provide support for XƎTEX.

15 Mixing vertical and horizontal text

This might seem like an exotic requirement and I have to admit I'm unlikely to use it myself. However, Japanese newspapers often mix vertical and horizontal text. You can do this in pTEX using `minipage`:

```
\documentclass{jarticle}
\usepackage{plext}
\begin{document}
私は魚が好きです。私は魚が好きです。私は魚が好きです。
私は魚が好きです。私は魚が好きです。私は魚が好きです。
```

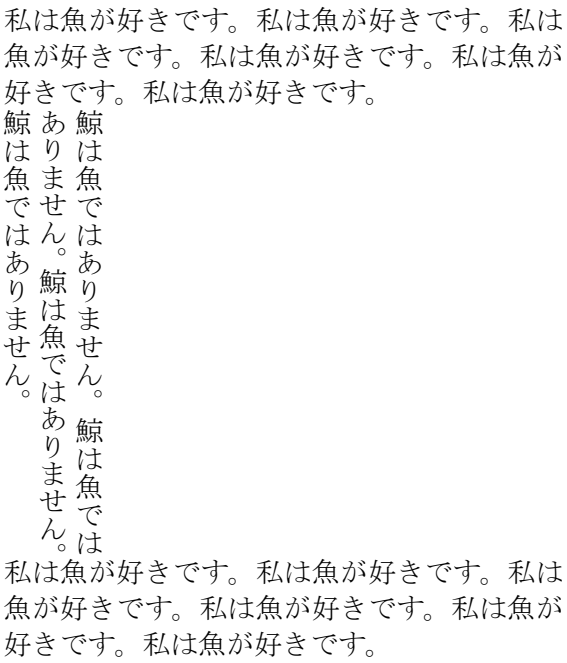

Figure 4: Vertical Japanese within horizontal

```
\begin{minipage}<t>{16zw}
鯨は魚ではありません。鯨は魚ではありません。
鯨は魚ではありません。鯨は魚ではありません。
\end{minipage}

私は魚が好きです。私は魚が好きです。私は魚が好きです。
私は魚が好きです。私は魚が好きです。私は魚が好きです。
\end{document}
```

Which yields something like Figure 4. The full syntax for `minipage` is described in the `platex.tex` format file as follows:

```
\begin{minipage}<dir>[pos]{width}...\end{minipage}
  dir: t ... force tate mode.
       y ... force yoko mode.
       z ... rotate 90 degree (ignored at yoko mode).
```

Yoko means horizontal mode. This syntax encourages the following test:

```
\documentclass{tarticle}
\usepackage{plext}
\begin{document}
```
私は魚が好きです。私は魚が好きです。私は魚が好きです。
私は魚が好きです。私は魚が好きです。私は魚が好きです。

```
\begin{minipage}<y>{16zw}
```
鯨は魚ではありません。鯨は魚ではありません。
鯨は魚ではありません。鯨は魚ではありません。
```
\end{minipage}
```

私は魚が好きです。私は魚が好きです。私は魚が好きです。
私は魚が好きです。私は魚が好きです。私は魚が好きです。

```
\begin{minipage}<z>{16zw}
```
鯨は魚ではありません。鯨は魚ではありません。
鯨は魚ではありません。鯨は魚ではありません。
```
\end{minipage}
```

私は魚が好きです。私は魚が好きです。私は魚が好きです。
私は魚が好きです。私は魚が好きです。私は魚が好きです。
```
\end{document}
```

Which yields something like Figure 5.
Note that a `zw` is a new unit of width introduced by pTEX.

16 Kanji font selection in LATEX

One way to make the default kanji text font in a LATEX document be a new one is brute force: `\font\normal=epkyo at 13pt\normal`. However, LATEX has a system for defining font sizes consistently and it is more architectural to use that. I think what follows is pretty much the same as you'd do for a new Western font except that `\rmdefault` is replaced by `\mcdefault`.

What you need to do is copy `jy1mc.fd` and `jt1mc.fd` from your installation pTEX tree to your local texmf tree (I put mine in `ptex\platex\base`) and rename them as `jy1ep.fd` and `jt1ep.fd`, where `ep` represents your new font. Change all the instances of `mc` in the files to `ep`. Then you need to change the following code

```
\DeclareFontShape{JY1}{mc}{m}{n}{<5> <6> ... <10> sgen*min
    <10.95><12><14.4><17.28><20.74><24.88> min10
```

私は魚が好きです。私は魚が好きです。私は魚が好きです。私は魚が好きです。私は魚が好きです。

鯨は魚ではありません。鯨は魚ではありません。鯨は魚ではありません。鯨は魚ではありません。

私は魚が好きです。私は魚が好きです。私は魚が好きです。私は魚が好きです。私は魚が好きです。

鯨は魚ではありません。鯨は魚ではありません。

Figure 5: Horizontal Japanese within vertical.

```
<-> min10
}{}
```

to

```
\DeclareFontShape{JY1}{ep}{m}{n}{<-> s*[1.3] epkyo}{}
```

Obviously you should replace `ep` here with your own font's name. The `[1.3]` is the magnification over the design size of the font (deliberately set high here at 130% because the Epson Kyoukasho font is rather small). The `s*` means, 'I don't care what size this ends up being so LaTeX should not warn me when it sees sizes it does not recognize'. The `epkyo` is the name of your font, which should exist as a `.tfm` file. All those numbers in pointy brackets and the `sgen*min` in the original are to do with fized sizes of the font and appear to be unneccesary in modern installations.

Once you've done this, create a file called `myfont.sty` in your local texmf tree and put something like the following code in it.

```
\ProvidesPackage{epkyo}
\renewcommand{\mcdefault}{ep}
\endinput
```

Then run `mktexlsr`, put `\usepackage{epkyo}` in the preable to your document and you should get your new font. If you want to change your gothic font rather than your mincho font then the above should work substituing *gt* for *mc* throughout. I haven't tried it.

17 Missing font shapes

You may find that pLaTeX complains about missing font shapes when compiling documents. The messages are benign because the the pLaTeX font code sensibly substitutes other fonts in their place. You can prevent these warnings by adding the missing shapes to the files `jt1ep.fd` and `jy1ep.fd` described in section 16 so that they look like this:

```
\DeclareFontShape{JT1}{ep}{m}{n}{<->s*[1.3] epkyo}{}
\DeclareFontShape{JT1}{ep}{m}{sc}{<->ssub*ep/m/n}{}
\DeclareFontShape{JT1}{ep}{bx}{n}{<->ssub*gt/m/n}{}
```

and this:

```
\DeclareFontShape{JY1}{ep}{m}{n}{<-> s*[1.3] epkyo}{}
\DeclareFontShape{JY1}{ep}{m}{sc}{<-> ssub*ep/m/n}{}
\DeclareFontShape{JY1}{ep}{bx}{n}{<-> ssub*gt/m/n}{}
```

It's the middle line in each of these that is new. pLaTeX also complains about the gothic kanji font shapes that one might have thought it would have defined itself (the file that does this seems to be plfonts.dtx). The `myfont.sty file` is a convenient (albeit unarchitectural) place to fix this up. You can do so by adding the following two lines so it looks something like this:

```
\ProvidesPackage{epkyo}
\renewcommand{\mcdefault}{ep}
\DeclareFontShape{JT1}{gt}{m}{it}{<->ssub*gt/m/n}{}
\DeclareFontShape{JY1}{gt}{m}{it}{<->ssub*gt/m/n}{}
\endinput
```

If you are not setting up your own fonts and just want to suppress warnings in a document compiled using the default fonts then another approach (the one used in this document) is to define a new style file that declares the font shapes that pLaTeX is complaining about. Mine is called `noswarn.sty`:

```
\ProvidesPackage{noswarn}
\DeclareFontShape{JT1}{gt}{m}{it}{<->ssub*gt/m/n}{}
\DeclareFontShape{JY1}{gt}{m}{it}{<->ssub*gt/m/n}{}
\DeclareFontShape{JT1}{mc}{m}{sc}{<->ssub*gt/m/n}{}
\DeclareFontShape{JY1}{mc}{m}{sc}{<->ssub*gt/m/n}{}
\endinput
```

Put `\usepackage{noswarn}` in your document's preamble and run `mktexlsr`. Next time you compile your documents the warnings should have stopped. If you still get some warnings then try adding the shapes that pLaTeX is complaining about to the style file.

18 Underlined Japanese text

Though underlining text is considered bad typography, it is easy to do in pTeX if you are in horizontal mode. Just use

```
{$\underline{\hbox{鯨を食べないで下さい}}$}
```

to get <u>鯨を食べないで下さい</u>.

I have seen pLaTeX packages that do underlining but all the documentation was in Japanese. I think they handle vertical format text.

19 Warichu

Warichu or 割り注 (わりちゅう) is a form of inserted notes within Japanese text. The characters are half the height of the main text. This facility is available in pLaTeX using the `warichu.sty` package. Figure 6 shows an example of text that includes *warichu*. I generated Figure 6 from the following code:

```
田中先生は\warichu{本}{教科書だけです。小説がきらいです。}が好きです。
```

In general, the syntax is

```
\warichu{X}{Y}Z
```

where `X` denotes the last ordinary character before the notes, `Y` denotes the notes themselves and `Z` denotes the characters that come after the notes.

This typographical device is more commonly used in vertical format. In *tate* mode one uses the macro `\twarichu` rather than `\warichu`. I generated the example in Figure 7 using the above code with `\twarichu` substituted for `\warichu`.

The `warichu.sty` macros `\warigaki` and `\twarigaki` are similar to `\warichu` and `\twarichu` except that they only take one parameter and they omit parentheses from the result. See Figures 8 and 9 for examples. I generated Figure 8 from the following code:

```
田中先生は\warigaki{本}{教科書だけです。小説がきらいです。}が好きです。
```

There are two limitations to the `warichu.sty` package. First, it does not support line-breaking for *warichu*; all your notes have to be on a single line. Only painstaking manual tweaking could produce multi-line *warichu*. Second, by default the font used for the *warichu* text is simply a scaled-down version of the main text font. Ideally the weight of the strokes in the *warichu* text would match that in the main text. A good knowledge of LaTeX and pLaTeX font management would probably allow an expert user to get around this problem. This could be done by switching to a weightier version of the main text font for the *warichu* text. がんばって。

20 References

[1] W32TeX download page in English: http://www.w32tex.org

[2] Ken Lunde. *CJKV Information Processing.* Second Edition, O'Reilly Media, 2009.

[3] http://www.physics.ucla.edu/~grosenth/jwpce.html

[4] Yasuki Saito. *Report on jTeX: A Japanese TeX.* TUGboat Volume 8, Number 2, July 1987.

[5] http://oku.edu.mie-u.ac.jp/~okumura/texfaq/japanese/ptex.html

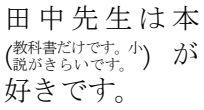

Figure 6: An example of horizontal *warichu*.

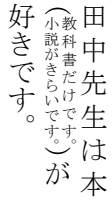

Figure 7: An example of vertical *warichu*.

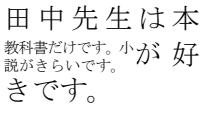

Figure 8: An example of horizontal *warigaki*.

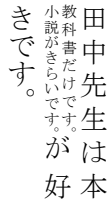

Figure 9: An example of vertical *warigaki*.