Guide to pTeX for developers unfamiliar with Japanese

Japanese TeX Development Community*

version p3.8.3, March 21, 2020

pTeX and its variants, upTeX, \(\varepsilon\)-pTeX and \(\varepsilon\)-upTeX, are all TeX engines with native Japanese support. Its output is always a DVI file, which can be processed by several DVI drivers with Japanese support including dvips and dvipdfmx. Formats based on LaTeX is called pLaTeX when running on pTeX/\(\varepsilon\)-pTeX, and called upLaTeX when running on upTeX/\(\varepsilon\)-upTeX.

Purpose of this document

This document is written for developers of TeX/LaTeX, who aim to support pTeX/\(\varepsilon\)-TeX and its variants upTeX/upLaTeX. Knowledge of the followings are assumed:

- Basic knowledge of Western TeX (Knuthian TeX, \(\varepsilon\)-TeX and pdfTeX),
- ... and its programming conventions.

Any knowledge of Japanese (characters, encodings, typesetting conventions etc.) is not assumed; some explanations are provided in this document when needed. We hope that this document helps authors of packages or classes to proceed with supporting pTeX family smoothly.

Note: This English edition (ptex-guide-en.pdf) is not meant to be a complete translation of Japanese edition (ptex-manual.pdf). For example, this document does not cover the following aspects of pTeX:

- Typesetting conventions of Japanese characters
- Details of vertical writing

For beginners of writing Japanese texts, please refer to the Japanese edition.

*https://texjp.org, e-mail: issue(at)texjp.org
# Contents

1 Brief introduction 4

1 \textit{pT\TeX} and its variants 4

2 Compatibility with Western \LaTeX 4

3 $\varepsilon$\LaTeX on \textit{p\LaTeX}/up\LaTeX — $\varepsilon$\textit{p\LaTeX}/up$\varepsilon$\LaTeX 4

4 Eminent characteristics of \textit{p\LaTeX}/up\LaTeX 5

II Details 6

5 Output format — DVI 6

5.1 Extensions of DVI format in \textit{p\LaTeX} family 7

5.2 DVI drivers with Japanese support 8

5.2.1 Using \textit{dvipdfmx} 8

5.2.2 Using \textit{dvips} 8

6 Programming on \textit{p\LaTeX} family 8

6.1 Number of registers and marks 8

6.2 Number of math families 9

6.3 Additional primitives and keywords 9

6.3.1 Sync\LaTeX additions (available in \textit{p\LaTeX}, up\LaTeX, $\varepsilon$-\textit{p\LaTeX}, $\varepsilon$-up\LaTeX) 9

6.3.2 \textit{p\LaTeX} additions (available in \textit{p\LaTeX}, up\LaTeX, $\varepsilon$-\textit{p\LaTeX}, $\varepsilon$-up\LaTeX) 10

6.3.3 up\LaTeX additions (available in up\LaTeX, $\varepsilon$-up\LaTeX) 11

6.3.4 $\varepsilon$-\textit{p\LaTeX} additions (available in $\varepsilon$-\textit{p\LaTeX}, $\varepsilon$-up\LaTeX) 12

6.3.5 $\varepsilon$-up\LaTeX additions (available in $\varepsilon$-up\LaTeX) 13

6.4 Omitted primitives and unsupported features 13

6.5 Behavior of Western \TeX primitives 13

6.5.1 Primitives with limitations in handling Japanese 13

6.5.2 Primitives capable of handling Japanese 14

6.6 Case study 14

6.6.1 Detecting \textit{p\LaTeX} 14

6.6.2 Detecting up\LaTeX 15

6.6.3 Defining large integer constants 15

6.6.4 Creating a Japanese character token with a specified code 16

6.7 Difference from pdf\LaTeX in DVI mode 18
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.8</td>
<td>Recommendation for file encoding</td>
<td>18</td>
</tr>
<tr>
<td>6.9</td>
<td>Input handling</td>
<td>18</td>
</tr>
<tr>
<td>6.10</td>
<td>Japanese tokens</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>Basic introduction to Japanese typesetting</td>
<td>18</td>
</tr>
<tr>
<td>7.1</td>
<td>Automatic insertion of glue and penalties</td>
<td>18</td>
</tr>
<tr>
<td>7.2</td>
<td>Japanese fonts</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>Other strange beasts</td>
<td>19</td>
</tr>
<tr>
<td>8.1</td>
<td>Internal kanji encodings</td>
<td>19</td>
</tr>
</tbody>
</table>
Part I

Brief introduction

1 pT\LaTeX{} and its variants

There is no advantage to choose \texttt{pT\LaTeX{}/upT\LaTeX{}/\epsilon-pT\LaTeX{}/\epsilon-upT\LaTeX{}, so we focus mainly on \texttt{\epsilon-pT\LaTeX{}/\epsilon-upT\LaTeX{}}.

2 Compatibility with Western T\LaTeX{}

\texttt{pT\LaTeX{}/upT\LaTeX{}} are almost upward compatible with Knuthian T\LaTeX{}, however, they do not pass the TRIP test. In \texttt{pT\LaTeX{}/upT\LaTeX{}}, input handling is different from Knuthian T\LaTeX{}; if a pair of two or more 8-bit codes matches Japanese character code, it is regarded as one Japanese character. There is no difference in handling 8-bit TFM font.

\texttt{\epsilon-pT\LaTeX{}/\epsilon-upT\LaTeX{} are almost upward compatible with \epsilon-TEX}, however, input handling is similar to \texttt{pT\LaTeX{}/upT\LaTeX{}}. It does not pass the e-TRIP test. That said, please note that “raw \epsilon-T\LaTeX{}” is unavailable anymore in T\LaTeX{} Live and derived distributions; they provide a command \texttt{etex} only as “DVI mode of pdfT\LaTeX{}.” We should note that \texttt{\epsilon-pT\LaTeX{}/\epsilon-upT\LaTeX{}} are not upward compatible with DVI mode of pdfT\LaTeX{}, which will be discussed later in section 6.7.

3 \LaTeX{} on \texttt{pT\LaTeX{}/upT\LaTeX{}} — \texttt{pLAT\LaTeX{}/upLAT\LaTeX{}}

Formats based on \LaTeX{} is called \texttt{pLAT\LaTeX{} when running on \texttt{pT\LaTeX{}/\epsilon-pT\LaTeX{}}}, and called \texttt{upLAT\LaTeX{} when running on \texttt{upT\LaTeX{}/\epsilon-upT\LaTeX{}}. In recent versions (around 2011) of T\LaTeX{} Live and its derivatives, the default engines of \texttt{pLAT\LaTeX{}} and \texttt{upLAT\LaTeX{}} are \texttt{\epsilon-pT\LaTeX{}} and \texttt{\epsilon-upT\LaTeX{}}. That is, the command \texttt{platex} starts \texttt{\epsilon-pT\LaTeX{}} (not \texttt{pT\LaTeX{}}) with preloaded format \texttt{platex.fmt}.

In the kernel level (\texttt{platex.ltx} and \texttt{uplatex.ltx}), \texttt{pLAT\LaTeX{}} and \texttt{upLAT\LaTeX{}} adds some additional commands related to the followings:

- Selection of Japanese fonts
- Crop marks (called “tombow”) for printings
- Adjustment for mixing horizontal and vertical texts

For authors, \texttt{pLAT\LaTeX{}/upLAT\LaTeX{}} are almost upward compatible with original \LaTeX{}, except for the followings:

- Order of float objects; in \texttt{pLAT\LaTeX{}/upLAT\LaTeX{}}, \langle bottom float \rangle is placed above \langle footnote \rangle. That is, the complete order is \langle top float \rangle \rightarrow \langle body text \rangle \rightarrow \langle bottom float \rangle \rightarrow \langle footnote \rangle.
For developers, additional care may be needed, for changes in the kernel macros and/or absence of \pdfTeX features.

4 Eminent characteristics of \TeX/\upTeX

The most important characteristics of \TeX/\upTeX can be summarized as follows:

- Japanese characters are interpreted and handled completely apart from Western characters.

- Texts can be aligned vertically, called “tate-gumi” (縦組). The horizontal alignment of texts is called “yoko-gumi” (横組), and both “tate-gumi” and “yoko-gumi” can be mixed even within a single document.
Part II
Details

5 Output format — DVI

The output of p\TeX family is always a DVI file. This is in contrast to the mainstream of pdf\TeX in the Western \TeX world.

In case you are not familiar with DVI output processing, first we give some general notice on how to get a “correct” output using \LaTeX in DVI mode.

- The DVI format is, as its name suggests, inherently driver-independent. However, some \LaTeX packages (\texttt{graphicx}, \texttt{color}, \texttt{hyperref} etc.) embed some \texttt{\special} commands into the DVI, which can be interpreted later by some specific DVI driver. Such a DVI is no longer driver-independent, thus those are called driver-dependent packages.

- In almost all major \TeX distributions (of course including \TeX Live), the default DVI driver is set to \texttt{dvips}. When you choose to process the resulting DVI file with a driver other than \texttt{dvips} (e.g. \texttt{dvipdfmx}) after running \LaTeX, you need to pass a proper driver option (e.g. \texttt{[dvipdfmx]}) to all driver-dependent packages.

Now, let’s move on to the situation in Japan, which is slightly complicated due to historical reasons but may also apply to other countries:

- There are two major conventions to pass a proper driver option to all driver-dependent packages:
  1. To give a driver option to each driver-dependent package:

```
\documentclass{article}
\usepackage[dvipdfmx]{graphicx}
\usepackage[dvipdfmx]{color}
```

  2. To have a driver option as global:

```
\documentclass[dvipdfmx]{article}
\usepackage{graphicx}
\usepackage{color}
```

The former convention has been used for many years since 1990s when the number of driver-dependent packages was limited. But in recent years (around 2010–), there are much more driver-dependent packages available. Thus we (Japanese \TeX experts) advise
a global driver option rather than individual package options for simplicity, but not yet fully widespread.\footnote{The fact that there had been a mismatch in option names ([dvipdfm] vs. [dvipdfmx]) between packages may also have been part of it; geometry did not understand [dvipdfmx] option until 2018!}

- Many people still see driver options as “optional”; they do without driver options unless really needed. For example, the convention of having a global driver option does no harm even when no driver-dependent package is used, but some users choose to omit a driver option to avoid a warning\footnote{Since LaTeX 2e 2020-02-02, this warning is effectively gone. This is due to preloading of expl3 into the format, and the driver-dependent code of expl3 interprets the global driver option.}:

\begin{verbatim}
\LaTeX Warning: Unused global option(s): [dvipdfmx].
\end{verbatim}

5.1 Extensions of DVI format in \texttt{pT\TeX} family

The DVI format output by \texttt{pT\TeX} family is fully compatible with Knuthian \texttt{T\TeX}, as long as the following conditions are met:

- No Japanese characters are typeset.
- There is no portion of vertical text alignment.

However, some additional DVI commands, which are defined in the standard [\cite{LaTeX2013}] but never used in \texttt{T\TeX}82, can come out.

- \texttt{set2} (129), \texttt{put2} (134): Appears in both \texttt{pT\TeX} and \texttt{upT\TeX} DVI. Used to typeset a Japanese character with 2-byte code.
- \texttt{set3} (130), \texttt{put3} (135): Appears in only \texttt{upT\TeX} DVI. Used to typeset a Japanese character with 3-byte code.

When \texttt{pT\TeX} is going to typeset a Japanese character into DVI, it is encoded in JIS, which is always a 2-byte code. For this purpose, \texttt{set2} or \texttt{put2} are used. When \texttt{upT\TeX} is going to output a Japanese character into DVI, it is encoded in UTF-32. If the code is equal to or less than \texttt{U+FFFF}, the lower 16-bit is used with \texttt{set2} or \texttt{put2}. If the code is equal to or greater than \texttt{U+10000}, the lower 24-bit is used with \texttt{set3} or \texttt{put3}.

In addition, \texttt{pT\TeX}/\texttt{upT\TeX} defines one additional DVI command.

- \texttt{dir} (255): Used to change directions of text alignment.

The DVI format in the preamble is always set to 2, as with \texttt{T\TeX}82. On the other hand, the DVI ID in the postamble can be special. Normally it is set to 2, as with \texttt{T\TeX}82; however, when \texttt{dir} (255) appears at least once in a single \texttt{pT\TeX}/\texttt{upT\TeX} DVI, the \texttt{post_post} table of postamble contains ID = 3.
5.2 DVI drivers with Japanese support

There are some DVI drivers with Japanese support. The most eminent drivers are dvips and dvipdfmx. Nowadays most of casual Japanese users are using dvipdfmx as a DVI driver. On the other hand, users of dvips are unignorable, especially those working in publishing industry. In recent years, most of major driver-dependent packages support both two drivers.

5.2.1 Using dvipdfmx

A DVI file which is output by pT\TEX can be converted directly to a PDF file using dvipdfmx. For Japanese fonts to be used in the output PDF, dvipdfmx refers to kanjix.map generated by the command updmap. You can use the script kanji-config-updmap to change font settings; please refer to its help message or documentation.

5.2.2 Using dvips

A DVI file which is output by pT\TEX can be converted to a PostScript file using dvips. For Japanese fonts to be used in the output PostScript, dvips refers to psfonts.map generated by the command updmap. You can use the script kanji-config-updmap to change font settings; please refer to its help message or documentation.

The resulting PostScript file can then be converted to a PDF file using Ghostscript (ps2pdf) or Adobe Distiller. When using Ghostscript, a proper setup of Japanese font must be done before converting PostScript into PDF. An easy solution for the setup is to run a script cjk-gs-integrate developed by Japanese T\TEX Development Community.

6 Programming on pT\TEX family

We focus on programming aspects of pT\TEX and its variants.

6.1 Number of registers and marks

pT\TEX and upT\TEX have exactly the same number (= 256) of registers (count, dimen, skip, muskip, box, and token) as Knuthian T\TEX. e-pT\TEX and e-upT\TEX in extended mode have more registers; there are 65536, which is twice as many as 32768 of e-T\TEX. Similarly e-pT\TEX and e-upT\TEX have 65536 mark classes, which is twice as many as 32768 of e-T\TEX.

The following code presents an example of detecting the number of registers and mark classes available:

\ifx\eTeXversion\undefined
% Knuthian TeX, pTeX, upTeX:
% 256 registers, 1 mark
Here a primitive \omathchar, which is derived from Ω, is used as a marker of a change file fam256.ch.³

### 6.2 Number of math families

In \pTeX{} and up\pTeX{}, the number of math fonts is restricted to 16, each of which can contain 256 characters (same as Knuthian \TeX{}). In ε\pTeX{} and ε-up\pTeX{}, a change file fam256.ch, which is derived from Ω, extends the upper limit to 256. As a consequence, ε\pTeX{} and ε-up\pTeX{} allows 256 math fonts, each of which can contain 256 characters.⁴

For \pLaTeX/up\LaTeX{} users to use more than 16 math fonts, it is necessary to use macros which exploit Ω-derived primitives such as \omathchar. Recent (u)\pLaTeX{} (since 2016/11/29) partially supports this, and the maximum number of math alphabets that can be defined by \DeclareMathAlphabet is extended to 256 (\e@mathgroup@top) without needing any extension package. However, symbol fonts are restricted to 16 as \DeclareMathSymbol etc still use the standard \mathchar etc. A simple solution to use more symbol fonts as well as math alphabets is to load a package mathfam256⁵ though it’s still preliminary.

### 6.3 Additional primitives and keywords

Here we provide only complete lists of additional primitives of \pTeX{} family in alphabetical order. The features of each primitive can be found in Japanese edition.

#### 6.3.1 Sync\TeX{} additions (available in \pTeX{}, up\TeX{}, ε\pTeX{}, ε-up\pTeX{})

In the standard build of \TeX{} Live, Sync\TeX{} extension is unavailable in Knuthian \TeX{}; however, it is enabled in \pTeX{} family.

---

³There is another \pTeX-derived engine named \pTeX-ng (or Asiatic \pTeX) https://github.com/clerkma/ptex-ng, it is based on ε-\TeX{} and up\TeX{}, but currently does not adopt fam256.ch so it has the same number of registers and mark classes as ε-\TeX{}.

⁴Ω allows 256 math fonts, each of which can contain 65536 characters.

⁵https://www.ctan.org/pkg/mathfam256
6.3.2 \TeX additions (available in p\TeX, up\TeX, \varepsilon-p\TeX, \varepsilon-up\TeX)

\texttt{\synctex}

\begin{itemize}
\item \texttt{\autospacing}
\item \texttt{\autoxspacing}
\item \texttt{\disinhibitglue} \textendash \textendash \texttt{New primitive since p3.8.2 (\TeX Live 2019)}
\item \texttt{\dtou}
\item \texttt{\euc}
\item \texttt{\ifdbox} \textendash \textendash \texttt{New primitive since p3.2 (\TeX Live 2011)}
\item \texttt{\ifddir} \textendash \textendash \texttt{New primitive since p3.2 (\TeX Live 2011)}
\item \texttt{\ifjfont} \textendash \textendash \texttt{New primitive since p3.8.3 (\TeX Live 2020)}
\item \texttt{\ifmbox} \textendash \texttt{New primitive since p3.7.1 (\TeX Live 2017)}
\item \texttt{\ifmdir}
\item \texttt{\iftbox}
\item \texttt{\iftfont} \textendash \texttt{New primitive since p3.8.3 (\TeX Live 2020)}
\item \texttt{\ifybox}
\item \texttt{\ifydir}
\item \texttt{\inhibitglue}
\item \texttt{\inhibitxspcode}
\item \texttt{\jcharwidowpenalty}
\item \texttt{\jfam}
\item \texttt{\jfont}
\item \texttt{\jis}
\item \texttt{\kanjiskip}
\item \texttt{\kansuji}
\item \texttt{\kansujichar}
\item \texttt{\kcatcode}
\item \texttt{\kuten}
\item \texttt{\noautospacing}
\end{itemize}

10
\noautoxspacing
\postbreakpenalty
\prebreakpenalty
\ptexminorversion — New primitive since p3.8.0 (\TeX\ Live 2018)
\ptexrevision — New primitive since p3.8.0 (\TeX\ Live 2018)
\ptexversion — New primitive since p3.8.0 (\TeX\ Live 2018)
\scriptbaselineshiftfactor — New primitive since p3.7 (\TeX\ Live 2016)
\scriptscriptbaselineshiftfactor — New primitive since p3.7 (\TeX\ Live 2016)
\showmode
\sjis
\tate
\tbaselineshift
\textbaselineshiftfactor — New primitive since p3.7 (\TeX\ Live 2016)
\tfont
\xkanjiskip
\xspcode
\ybaselineshift
\yoko
\H
\Q
\zh
\zw

6.3.3 up\TeX\ additions (available in up\TeX, \textbackslash{\epsilon}-up\TeX)
\disablecjktoken
\enablecjktoken
\forcecjktoken
\kchar
\kchardef
\ucs
\uptexrevision — New primitive since u1.23 (\TeX\ Live 2018)
\uptexversion — New primitive since u1.23 (\TeX\ Live 2018)
6.3.4 \(\varepsilon\text{-}\LaTeX\) additions (available in \(\varepsilon\text{-}\LaTeX, \varepsilon\text{-up\LaTeX})

- \texttt{\currentspacingmode} — New primitive since 191112 (\LaTeX~Live 2020)
- \texttt{\currentxspacingmode} — New primitive since 191112 (\LaTeX~Live 2020)
- \texttt{\epTeXinputencoding} — New primitive since 160201 (\LaTeX~Live 2016)
- \texttt{\epTeXversion} — New primitive since 180121 (\LaTeX~Live 2018)
- \texttt{\expanded} — New primitive since 180518 (\LaTeX~Live 2019)
- \hfi
- \texttt{\ifincsname} — New primitive since 190709 (\LaTeX~Live 2020)
- \texttt{\ifpdfprimitive} — New primitive since 150805 (\LaTeX~Live 2016)
- \texttt{\lastnodechar} — New primitive since 141108 (\LaTeX~Live 2015)
- \texttt{\lastnodesubtype} — New primitive since 180226 (\LaTeX~Live 2018)
- \odelcode
- \odelimiter
- \omathaccent
- \omathchar
- \omathchardef
- \omathcode
- \oradical
- \pagefistretch
- \texttt{\pdfcreationdate} — New primitive since 130605 (\LaTeX~Live 2014)
- \texttt{\pdfelapsedtime} — New primitive since 161114 (\LaTeX~Live 2017)
- \texttt{\pdffiledump} — New primitive since 140506 (\LaTeX~Live 2015)
- \texttt{\pdffilemoddate} — New primitive since 130605 (\LaTeX~Live 2014)
- \texttt{\pdffilesizex} — New primitive since 130605 (\LaTeX~Live 2014)
- \texttt{\pdflastxpos}
- \texttt{\pdflastypos}
- \texttt{\pdffivesum} — New primitive since 150702 (\LaTeX~Live 2016)
- \texttt{\pdfnormaldeviate} — New primitive since 161114 (\LaTeX~Live 2017)
- \pdff pageheight
- \pdfpagewidth
\pdfprimitive — New primitive since 150805 (\TeX~Live 2016)
\pdfrandomseed — New primitive since 161114 (\TeX~Live 2017)
\pdfresettimer — New primitive since 161114 (\TeX~Live 2017)
\pdfsavepos
\pdfsetrandomseed — New primitive since 161114 (\TeX~Live 2017)
\pdfshellescape — New primitive since 141108 (\TeX~Live 2015)
\pdfstrcmp
\pdfuniformdeviate — New primitive since 161114 (\TeX~Live 2017)
\readpapersizespecial — New primitive since 180901 (\TeX~Live 2019)
\Uchar — New primitive since 191112 (\TeX~Live 2020)
\Ucharcat — New primitive since 191112 (\TeX~Live 2020)
\vfi
\fi

6.3.5 \textit{\textasciitilde-up\TeX} additions (available in \textit{\textasciitilde-up\TeX})

\currentcjktoken — New primitive since 191112 (\TeX~Live 2020)

6.4 Omitted primitives and unsupported features

Compared to Knuthian \TeX and \textit{\textasciitilde-up\TeX}, some primitives are omitted due to conflict with Japanese handling. One is enc\TeX extension, such as \texttt{\mubyte}. The ML\TeX extension, such as \texttt{\charsubdef}, is included but not well-tested.

6.5 Behavior of Western \TeX primitives

Here we provide some notes on behavior of Knuthian \TeX and \textit{\textasciitilde-up\TeX} primitives when used within p\TeX family.

6.5.1 Primitives with limitations in handling Japanese

Each of the following primitives allows only character codes 0–255; other codes will give an error “! Bad character code.”

\texttt{\catcode, \sfcode, \mathcode, \delcode, \lccode, \uccode}.

Each of the following primitives has \texttt{\ldots char} in its name, however, the effective values are restricted to 0–255.
6.5.2 Primitives capable of handling Japanese

The following primitives are extended to support Japanese characters:

- **\char ⟨character code⟩**, **\chardef ⟨control sequence⟩ = ⟨character code⟩**
  
  In addition to 0–255, internal codes of Japanese characters are allowed. For putting Japanese characters, a Japanese font is chosen. More information can be found in 6.6.3.

- **\font**, **\fontname**, **\fontdimen**

- **\accent ⟨character code⟩ = ⟨character⟩**

- **\if ⟨token1⟩ ⟨token2⟩, \ifcat ⟨token1⟩ ⟨token2⟩**

  Japanese character token is also allowed. In that case,
  
  - **\if** tests the internal character code of the Japanese character.
  - **\ifcat** tests the \kcatcode of the Japanese character.

  \TeXbook describes the behavior of \if and \ifcat as follows;

  If either token is a control sequence, \TeX considers it to have character code 256 and category code 16, unless the current equivalent of that control sequence has been \let equal to a non-active character token.

  However, this includes a lie; in the real implementation of \textweb, a control sequence is considered to have a category code 0.

6.6 Case study

Here we provide some code examples which may be useful for package developers.

6.6.1 Detecting p\TeX

Since the primitive \ptexversion is rather new (added in 2018), the safer solution for detecting p\TeX is to test if a primitive \kanjiskip is defined.

```
\ifx\kanjiskip\undefined
  \else
    \% p\TeX / up\TeX / e-p\TeX / e-up\TeX
  \fi
```

14
6.6.2 Detecting up\TeX

\upTeX/-up\TeX are almost upward compatible with p\TeX/-p\TeX respectively, however, there are two major differences:

1. Improvements in the \texttt{\textbackslash{}kcatcode} business, mainly for better handling of Latin-1 characters and CJK tokens.

2. Unicode as the default internal kanji encoding, for direct use of its huge character set.

The first difference can be detected by checking if \texttt{\textbackslash{}enablecjktoken} primitive is defined.

\begin{verbatim}
\ifx\enablecjktoken\undefined
  \else
    \% up\TeX/e-up\TeX
  \fi
\end{verbatim}

The second difference can be detected by checking if the character 0x2121 (fullwidth space in JIS encoding) is stored as "3000 internally.

\begin{verbatim}
\ifx\kanjiskip\undefined
  \else
    \ifnum\jis"2121="3000
      \% up\TeX/e-up\TeX with internal Unicode
    \else
      \% p\TeX/e-p\TeX
      \% or, up\TeX/e-up\TeX with internal EUC-JP or Shift-JIS
    \fi
  \fi
\end{verbatim}

Please note that the format-build setting of \texttt{-kanji-internal=(sjis|euc)} with up\TeX makes it effectively p\TeX regarding the character set, which means that only JIS X 0208 character set is supported.

6.6.3 Defining large integer constants

According to \texttt{[2]} (Section 3.3),

A control sequence that has been defined with \texttt{\chardef} command can also be used as a \texttt{(number)}. This fact is used in allocation commands such as \texttt{\newbox}. Tokens defined with \texttt{\mathchardef} can also be used this way.

Here is the list of primitives which can be used for this purpose in p\TeX family:
\chardef (control sequence) = (character code)
Defines a control sequence to be a synonym for \char (character code).

\kchardef (control sequence) = (character code) (for up\TeX/\e-up\TeX)
Defines a control sequence to be a synonym for \kchar (character code).

\mathchardef (control sequence) = (15-bit number)
Defines a control sequence to be a synonym for \mathchar (15-bit number).

\omathchardef (control sequence) = (27-bit number) (for \e-p\TeX/\e-up\TeX)
Defines a control sequence to be a synonym for \omathchar (27-bit number).

The first two (\chardef and \kchardef) are usable only when the integer being defined is in the range of valid character codes, which is not necessarily continuous (see Section 8.1). The most efficient and convenient way of defining integer constants is as follows:

- 0–255: \chardef
- 256–32767: \mathchardef
- 32768–134217727: \omathchardef (only for \e-p\TeX/\e-up\TeX)
- (optional) 256–2147483647: \chardef (only for up\TeX/\e-up\TeX)

6.6.4 Creating a Japanese character token with a specified code

Short version:

- With \e-p\TeX 191112 or later (\TeX Live 2020), you can use expandable primitives \Uchar and \Ucharcat.
- Otherwise, use the "\kansuji trick".

The "\kansuji trick"
This is a modified version of the "\lowercase trick" available in p\TeX family.

Short note on the "\lowercase trick": to create a character token with a specified code value between 0–255 with Knuthian \TeX, the "\lowercase trick" can be used; for example,

\begingroup
\lccode`\? = \mycount
\lowercase{\endgroup \def\X{?}}
defines \X which expands to a character number \mycount while the \catcode of ? (12) is preserved. However, the trick cannot be applied to Japanese characters, since \pif\X family does not support \lccode outside 0–255.

\kansuji is an expandable primitive like \number or \romannumeral, and it converts an integer into its corresponding kanji notation called “kansuji” (漢数字). The important point here is that the number-kanji mapping can be altered by \kansujichar.

Example 1: equivalent to \def\X{あ} (JIS code 0x2422 is “あ”):
\begingroup
\kansujichar1=\jis"2422 \xdef\X{\kansuji1}
\endgroup

Example 2: equivalent to \def\日本{Japan}.
\begingroup
\kansujichar5=\jis"467C\relax
\kansujichar6=\jis"4B5C\relax
\expandafter\gdef\csname\kansuji56\endcsname{Japan}
\endgroup

Since \kansujichar accepts only Japanese character code, the “\kansuji trick” and the “\lowercase trick” should be used complementarily.

\Uchar, \Ucharcat

The “\kansuji trick” above includes an assignment of \kansujichar which is unexpandable. \dpif\X 191112 or later (\pifX Live 2020) provides expandable primitives \Uchar and \Ucharcat, which are derived from \X\X\X. Regardless of their names, and unlike \X\X\X or \L\a\X, these primitives do not necessarily take a Unicode value as an argument. These primitives in \dpif\X and \up\dpif\X take a valid character code (see \S\ref{sec-\Uchar}) based on the internal kanji encoding.

\Uchar \langle character code \rangle

Expands to a character token with specified slot \langle character code \rangle.

- When an 8-bit number (0–255) is given, it expands to a Latin character token with category code 12, except for a space character (32) which has category code 10.
- When a Japanese character code greater than 255 is given, it expands to a Japanese character token with its current category code; 16–18 for \dpif\X, 16–19 for \up\dpif\X.

\Ucharcat \langle character code \rangle \langle category code \rangle

Expands to a character token with slot \langle character code \rangle and \langle category code \rangle specified.

- With \dpif\X:
– Only 8-bit number (0–255) are allowed for ⟨character code⟩; that is, only Latin characters can be generated.
– The values allowed for ⟨category code⟩ are 1–4, 6–8, 10–13.
• With ε-upTEX:
  – When ⟨character code⟩ is between 0–127, only Latin characters can be generated. Thus, the values allowed for ⟨category code⟩ are 1–4, 6–8, 10–13.
  – When ⟨character code⟩ is between 128–255, both Latin and Japanese characters can be generated depending on the specified ⟨category code⟩; 1–4, 6–8, 10–13: Latin character, 16–19: Japanese character.
  – When ⟨character code⟩ is greater than 255, only Japanese characters can be generated. Thus, the values allowed for ⟨category code⟩ are 16–19.

6.7 Difference from pdfTEX in DVI mode
As stated above, ε-pTEX/ε-upTEX are not upward compatible with DVI mode of pdfTEX.

6.8 Recommendation for file encoding

6.9 Input handling
For simplicity, first we introduce of input handling of ε-upTEX.

6.10 Japanese tokens

7 Basic introduction to Japanese typesetting
This section does not aim to explain Japanese typesetting completely; here we provide a minimum requirement for “getting away” with Japanese.

7.1 Automatic insertion of glue and penalties
Sometimes pTEX family automatically inserts glue and penalties between characters.

7.2 Japanese fonts
pTEX family can have 3 different “current” fonts at the same time; a Latin font, a Japanese font for horizontal writing (“yoko-gumi”), and a Japanese font for vertical writing (“tate-gumi”). The first one is the same as in the Knuthian TEX, which is defined in a standard TFM format.
The latter two are specific to pT EX family, which are defined in a JFM (Japanese T EX font metric) format.\(^6\) While typesetting, pT EX family automatically switches between these 3 fonts, depending on the character code and the writing direction:

- For typesetting Latin characters, the current Latin font shown by \texttt{\the\font} is selected.
- For typesetting Japanese characters, the current Japanese font suitable for the current writing direction is selected. It is shown by \texttt{\the\jfont} for horizontal writing and \texttt{\the\tfont} for vertical writing.

In Knuthian T EX, the primitive \texttt{\nullfont} refers to an “empty font” in which all characters are undefined. However in pT EX family, this is regarded as a Latin font and there is no equivalent to “Japanese \nullfont” by design. To elaborate, it is possible only when no Japanese font is set globally, i.e. in iniT EX mode. Once a valid Japanese font is selected, there is no way of selecting “Japanese \nullfont” to discard all characters.

Moreover, pT EX and friends assume that each Japanese font (except “Japanese \nullfont” in iniT EX mode) contains all valid Japanese character code. In other words, all Japanese fonts share the same character set corresponding to the whole valid Japanese character code range.

\section{Other strange beasts}

\subsection{Internal kanji encodings}

The \langle character code \rangle is a union of the following two:

- Range of numbers between 0–255, and
- Numbers allowed for internal code of Japanese characters.

The former is the same as Knuthian T EX, but the latter is a problem. In upT EX/\varepsilon-upT EX with -kanji-internal=uptex (default on), the range is very simple:

\[ c \geq 0 \]

However in pT EX/\varepsilon-pT EX, only legacy encodings (EUC-JP as euc, or Shift-JIS as sjis) are available for -kanji-internal1. In this case, the range can be represented as follows:

\[ c = 256c_1 + c_2 \quad (c_i \in C_i) \]

\footnote{A JFM is a modified version of the standard TFM. It can be created by (u)pPtoTF, and decoded by (u)pTFtoPL. Please also refer to the man pages of these programs (ppltotf.man1.pdf and ptftopl.man1.pdf).}
where
\[
\begin{align*}
C_1 &= C_2 = \{\text{"a1}, \ldots, \text{"fe}\} \quad (\text{euc}), \\
C_1 &= \{\text{"81}, \ldots, \text{"9f}\} \cup \{\text{"e0}, \ldots, \text{"fc}\},
C_2 = \{\text{"40}, \ldots, \text{"7e}\} \cup \{\text{"80}, \ldots, \text{"fc}\} \quad (\text{jis}).
\end{align*}
\]

Therefore, the overall range of character code is not continuous.

To check whether an integer is a valid Japanese character code or not, you can use \texttt{\iffontchar} with \texttt{\\$\$-\texttt{p\TeX} 190709 or later (\TeX\ Live 2020). Suppose a count register \texttt{\mycount} stores an integer, you can do it as follows:

\begin{verbatim}
\iffontchar\jfont\mycount
% \mycount is a valid Japanese character code
\fi
\end{verbatim}

Here the primitive \texttt{\jfont} is used merely as a representative non-empty\footnote{This assumption is always safe after one of the standard \TeX \ formats (e.g. plain \TeX, \LaTeX) is loaded.} Japanese font containing all valid Japanese character code (see $\square$).
References

   https://ctan.org/pkg/dvistd

   https://www.eijkhout.net/texbytopic/texbytopic.html
## Index

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\accent</td>
<td>\inhibitxspcode</td>
</tr>
<tr>
<td>\autospacing</td>
<td>\jcharwidowpenalty</td>
</tr>
<tr>
<td>\autoxspacing</td>
<td>\jmath</td>
</tr>
<tr>
<td>\char</td>
<td>\jkern</td>
</tr>
<tr>
<td>\chardef</td>
<td>\kanjiskip</td>
</tr>
<tr>
<td>\currentcjk</td>
<td>\kansui</td>
</tr>
<tr>
<td>\currentxspacing</td>
<td>\kansujchar</td>
</tr>
<tr>
<td>\disablecjk</td>
<td>\kcatcode</td>
</tr>
<tr>
<td>\disinhibitglue</td>
<td>\kchar</td>
</tr>
<tr>
<td>\dtou</td>
<td>\kchardef</td>
</tr>
<tr>
<td>\enablecjk</td>
<td>\kuten</td>
</tr>
<tr>
<td>\epTeXinputencoding</td>
<td>\lastnodechar</td>
</tr>
<tr>
<td>\epTeXversion</td>
<td>\lastnodesubtype</td>
</tr>
<tr>
<td>\euc</td>
<td>\mathchardef</td>
</tr>
<tr>
<td>\expanded</td>
<td>\noautospacing</td>
</tr>
<tr>
<td>\font</td>
<td>\noautoxspacing</td>
</tr>
<tr>
<td>\fontdimen</td>
<td>\odelimiter</td>
</tr>
<tr>
<td>\fontname</td>
<td>\omathaccent</td>
</tr>
<tr>
<td>\forcecjk</td>
<td>\omathchar</td>
</tr>
<tr>
<td>\hfi</td>
<td>\omathchardef</td>
</tr>
<tr>
<td>\if</td>
<td>\omathcode</td>
</tr>
<tr>
<td>\ifcat</td>
<td>\oradical</td>
</tr>
<tr>
<td>\ifdirbox</td>
<td>\pagefistretch</td>
</tr>
<tr>
<td>\ifdir</td>
<td>\pagedevice</td>
</tr>
<tr>
<td>\ifincsname</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\ifjfont</td>
<td>\pdfelapsedtime</td>
</tr>
<tr>
<td>\ifmbox</td>
<td>\pdffiledump</td>
</tr>
<tr>
<td>\ifmbox</td>
<td>\pdffilename</td>
</tr>
<tr>
<td>\ifpdfprimitive</td>
<td>\pdffilesize</td>
</tr>
<tr>
<td>\iftbox</td>
<td>\pdflastxpos</td>
</tr>
<tr>
<td>\iftdir</td>
<td>\pdflastypos</td>
</tr>
<tr>
<td>\iftfont</td>
<td>\pdfmdfivesum</td>
</tr>
<tr>
<td>\inhibitxspcode</td>
<td>\pdfnormaldeviate</td>
</tr>
<tr>
<td>\ifybox</td>
<td>\pdfpageheight</td>
</tr>
<tr>
<td>\ifydir</td>
<td>\pdfpagewidth</td>
</tr>
<tr>
<td>\inhibitglue</td>
<td>\pdfprimitive</td>
</tr>
</tbody>
</table>

22