Guide to p\TeX\ for developers unfamiliar with Japanese

Japanese \TeX\ Development Community*

version p4.1.0, September 2, 2023

\tex\ and its variants, upTEX, \varepsilon\-p\TeX\ 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 p\LaTeX\ when running on p\TeX\/\varepsilon\-p\TeX\, and called up\LaTeX\ when running on up\TeX\/\varepsilon\-up\TeX\.

**Purpose of this document**

This document is written for developers of \TeX\/\LaTeX, who aim to support p\TeX\/p\LaTeX\ and its variants up\TeX\/up\LaTeX. Knowledge of the followings are assumed:

- Basic knowledge of Western \TeX\ (Knuthian \TeX, \varepsilon\-\TeX\ and pdf\TeX),
- ... 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 p\TeX\ family smoothly.

Note: This English guide (ptex-guide-en.pdf) is not meant to be a complete translation of Japanese manual (ptex-manual.pdf). For example, this document does not cover issues regarding Japanese typesettings. If you are interested in typesetting conventions of Japanese text, please also refer to japanese.pdf distributed with babel-japanese.

This document is maintained at: https://github.com/texjporg/ptex-manual/

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

1 Breif introduction ........................................................................... 4
1 pTEX and its variants ...................................................................... 4
2 Eminent characteristics of pTEX family ........................................ 4
3 Compatibility with Western T\(\times \) .............................................. 5
4 L\(\text{T\(\times \}}\) on pTEX/upTEX — pL\(\text{T\(\times \}}\) ........................................ 5

II Details .......................................................................................... 7
5 Output format — DVI ................................................................. 7
  5.1 Extensions of DVI format in pTEX family ................................. 8
  5.2 DVI drivers with Japanese support ....................................... 9
    5.2.1 Using dvipdfmx ............................................................ 9
    5.2.2 Using dvips ............................................................... 9
6 Programming on pTEX family ..................................................... 9
  6.1 Number of registers and marks ............................................. 9
  6.2 Number of math families ..................................................... 10
  6.3 Additional primitives and keywords ..................................... 10
    6.3.1 pTEX additions (available in pTEX, upTEX, e-pTEX, e-upTEX) 10
    6.3.2 upTEX additions (available in upTEX, e-upTEX) ............. 12
    6.3.3 e-pTEX additions (available in e-pTEX, e-upTEX) .......... 13
    6.3.4 e-upTEX additions (available in e-upTEX) .................. 14
    6.3.5 Other cross-engine additions .................................... 14
  6.4 Omitted primitives and unsupported features ....................... 15
  6.5 Behavior of Western T\(\times \) primitives ................................. 15
    6.5.1 Primitives with limitations in handling Japanese .......... 15
    6.5.2 Primitives capable of handling Japanese .................... 15
  6.6 Case study ............................................................................. 16
    6.6.1 Detecting pTEX ........................................................ 16
    6.6.2 Detecting upTEX ....................................................... 16
    6.6.3 Defining large integer constants ................................ 17
    6.6.4 Creating a Japanese character token with a specified code 18
  6.7 Difference from pdfT\(\times \) in DVI mode ................................. 20
6.8 Recommendation for file encoding ........................................ 20
6.9 Input handling ............................................................ 22
6.10 Japanese tokens .......................................................... 22

7 Basic introduction to Japanese typesetting .......................... 22
7.1 Automatic insertion of glue and penalties ......................... 22
7.2 Japanese fonts ............................................................ 22

8 Other strange beasts ....................................................... 23
8.1 Internal Japanese encodings ......................................... 23
Part I

Brief introduction

1 pTeX and its variants

The figure below shows the relationship between engines.

\[
\begin{array}{c}
pdf\TeX \\
\varepsilon-\TeX \\
\varepsilon-p\TeX \\
p\TeX \\
\varepsilon-p\TeX \\
\varepsilon-up\TeX \\
up\TeX \\
\end{array}
\]

pTeX is an old Japanese-specific extension of T\TeX 82, which aims to support proper type-setting of Japanese text but only supports a limited character set, JIS X 0208 (6879 characters).

upTeX is developed as an extension of pTeX to support full Unicode characters. It also includes modifications and extensions to overcome the difficulties of pTeX in processing 8-bit Latin characters due to conflicts with legacy multibyte Japanese encodings.

\varepsilon-p\TeX and \varepsilon-up\TeX are \varepsilon-\TeX extensions of pTeX and upTeX respectively. In the current release, some extensions derived from pdf\TeX and \Omega are also available.

2 Eminent characteristics of pTeX family

The most important characteristics of pTeX family can be summarized as follows:

- Japanese characters are interpreted and handled completely apart from Western characters. If a pair of two or more 8-bit codes in the input matches the pattern of Japanese character codes, it is regarded as one Japanese character and given a different \texttt{\catcode} value.

- There are two text directions; horizontal (yoko-gumi; 橫組) and vertical (tate-gumi; 縦組). Two directions can be mixed even within a single document.
3 Compatibility with Western \TeX

p\TeX/\up\TeX are almost upward compatible with Knuthian \TeX, however, they do not pass the TRIP test. The most important difference lies in the handling of 8-bit code inputs; some 8-bit Latin characters may be subject to the encoding conversion. There is no difference in handling 8-bit TFM font.

\varepsilon\TeX/\varepsilon\up\TeX are almost upward compatible with \varepsilon\TeX, however, input handling is similar to \p\TeX/\up\TeX. It does not pass the e-TRIP test. That said, please note that “raw \varepsilon\TeX” is unavailable anymore in \TeX Live and derived distributions; they provide a command \texttt{etex} only as “DVI mode of pdf\TeX.” You should note that \varepsilon\TeX/\varepsilon\up\TeX are not upward compatible with DVI mode of pdf\TeX, which will be discussed later in Section 6.7.

There is no advantage to choose \p\TeX/\up\TeX over \varepsilon\TeX/\varepsilon\up\TeX, so we focus mainly on \varepsilon\TeX/\varepsilon\up\TeX.

4 \LaTeX on \p\TeX/\up\TeX --- \LaTeX/\up\LaTeX

Format based on \LaTeX is called p\LaTeX when running on \p\TeX, and called up\LaTeX when running on up\TeX. When building the format, \texttt{platex.ltx} (p\LaTeX) or \texttt{uplatex.ltx} (up\LaTeX) loads \texttt{latex.ltx} first and adds some additional commands related to the followings:

- Selection of Japanese fonts,
- Crop marks (called tombow; トンボ) for printings,
- Adjustment for mixing horizontal and vertical texts.

For authors, p\LaTeX/\up\LaTeX are almost upward compatible with original \LaTeX, except for the followings:

- Order of float objects; in p\LaTeX/\up\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 the engine difference (Japanese handling, absence of pdf\TeX features, etc). In recent versions of \TeX Live and its derivatives, the default engines of p\LaTeX and up\LaTeX are as follows:

<table>
<thead>
<tr>
<th>Date</th>
<th>p\LaTeX</th>
<th>up\LaTeX</th>
</tr>
</thead>
<tbody>
<tr>
<td>\TeX Live 2010</td>
<td>\p\TeX</td>
<td>—</td>
</tr>
<tr>
<td>\TeX Live 2011</td>
<td>\varepsilon\p\TeX</td>
<td>—</td>
</tr>
<tr>
<td>\TeX Live 2012–2022, 2023 initial</td>
<td>\varepsilon\p\TeX</td>
<td>\varepsilon\up\TeX</td>
</tr>
<tr>
<td>Since 2023-06-01</td>
<td>\varepsilon\up\TeX in legacy (see below)</td>
<td>\varepsilon\up\TeX</td>
</tr>
</tbody>
</table>
The command \texttt{platex} started \texttt{$\varepsilon$-p\TeX} (not \texttt{p\TeX}) with preloaded format \texttt{platex.fmt} in \TeX\ Live 2022. Since 2023-06-01, \texttt{p\LaTeX} has switched its engine from \texttt{$\varepsilon$-p\TeX} to \texttt{$\varepsilon$-up\TeX} in “legacy-encoding-compatibility mode.” It means that additional primitives of \texttt{(\varepsilon-)}\texttt{up\TeX} is available also on \texttt{p\LaTeX}, but the internal code of Japanese characters is still non-Unicode to keep the backward compatibility. Further information can be found in Section 5.6.2.
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 (graphicx, color, 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:

     \begin{verbatim}
     \documentclass{article}
     \usepackage[dvipdfmx]{graphicx}
     \usepackage[dvipdfmx]{color}
     \end{verbatim}

  2. To have a driver option as global:

     \begin{verbatim}
     \documentclass[dvipdfmx]{article}
     \usepackage{graphicx}
     \usepackage{color}
     \end{verbatim}

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.

- 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:

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

5.1 Extensions of DVI format in p\TeX family

The DVI format output by p\TeX family is fully compatible with Knuthian \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{pTeX} but never used in \TeX82, can come out.

- \texttt{set2} (129): Used to typeset a Japanese character with 2-byte code (both p\TeX and up\TeX).
- \texttt{set3} (130): Used to typeset a Japanese character with 3-byte code (up\TeX only).

When p\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 up\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, p\TeX/\up\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 \TeX82. On the other hand, the DVI ID in the postamble can be special. Normally it is set to 2, as with \TeX82; however, when \texttt{dir} (255) appears at least once in a single p\TeX/\up\TeX DVI, the post_post table of postamble contains \texttt{ID = 3}.

\footnote{The fact that there had been a mismatch in option names ([dvipdfm] vs. [dvipdfmx]) between packages may also have been part of it; \texttt{geometry} did not understand \texttt{[dvipdfmx]} option until 2018!}

\footnote{Since \LaTeX 2ε 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.}
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 p\TeX{} can be converted directly to a PDF file using dvipdfmx. For Japanese fonts to be used in the output PDF, dvipdfmx refers to \texttt{kanjix.map} generated by the command \texttt{updmap}. You can use the script \texttt{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 p\TeX{} can be converted to a PostScript file using dvips. For Japanese fonts to be used in the output PostScript, dvips refers to \texttt{psfonts.map} generated by the command \texttt{updmap}. You can use the script \texttt{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 (\texttt{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 \texttt{cjk-gs-integrate} developed by Japanese \TeX{} Development Community.

6 Programming on p\TeX{} family

We focus on programming aspects of p\TeX{} and its variants.

6.1 Number of registers and marks

p\TeX{} and up\TeX{} have exactly the same number (\(= 256\)) of registers (count, dimen, skip, muskip, box, and token) as Knuthian \TeX{}. \(\varepsilon\)-p\TeX{} and \(\varepsilon\)-up\TeX{} in extended mode have more registers; there are 65536, which is twice as many as 32768 of \(\varepsilon\)-\TeX{}. Similarly \(\varepsilon\)-p\TeX{} and \(\varepsilon\)-up\TeX{} have 65536 mark classes, which is twice as many as 32768 of \(\varepsilon\)-\TeX{}.

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

\begin{verbatim}
\if\eTeXversion\undefined
% Knuthian TeX, pTeX, upTeX:
\% 256 registers, 1 mark
\end{verbatim}
Here a primitive \omathchar, which is derived from Ω, is used as a marker of a change file fam256.ch.\footnote{There is another \fontfamily{ptex}{\ttfamily ptex-ng} (or Asiatic \fontfamily{ptex}{\ttfamily ptex}) \linkurl{https://github.com/clerkma/ptex-ng} it is based on \fontfamily{ptex}{\ttfamily ptex} and up\fontfamily{ptex}{\ttfamily ptex}, but currently does not adopt fam256.ch so it has the same number of registers and mark classes as \fontfamily{ptex}{\ttfamily ptex}.}

### 6.2 Number of math families

In \fontfamily{ptex}{\ttfamily ptex} and up\fontfamily{ptex}{\ttfamily ptex}, the number of math fonts is restricted to 16, each of which can contain 256 characters (same as Knuthian \fontfamily{ptex}{\ttfamily ptex}). In \fontfamily{ptex}{\ttfamily ptex} and up\fontfamily{ptex}{\ttfamily ptex}, a change file fam256.ch, which is derived from Ω, extends the upper limit to 256. As a consequence, \fontfamily{ptex}{\ttfamily ptex} and up\fontfamily{ptex}{\ttfamily ptex} allows 256 math fonts, each of which can contain 256 characters.\footnote{Ω allows 256 math fonts, each of which can contain 65536 characters.}

For \fontfamily{ptex}{\ttfamily ptex}/up\fontfamily{ptex}{\ttfamily ptex} users to use more than 16 math fonts, it is necessary to use macros which exploit Ω-derived primitives such as \omathchar. Recent (u)\fontfamily{ptex}{\ttfamily ptex} (since 2016/11/29) partially supports this, and the maximum number of math alphabets that can be defined by \DeclareMathAlphabet is extended to 256 \fontfamily{ptex}{\ttfamily \LaTeX} without needing any extension package. However, symbol fonts are restricted to 16 as \DeclareMathSymbol etc still use the standard \char etc. A simple solution to use more symbol fonts as well as math alphabets is to load a package mathfam256\footnote{\linkurl{https://www.ctan.org/pkg/mathfam256}} though it’s still preliminary.

### 6.3 Additional primitives and keywords

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

#### 6.3.1 \fontfamily{ptex}{\ttfamily ptex} additions (available in \fontfamily{ptex}{\ttfamily ptex}, up\fontfamily{ptex}{\ttfamily ptex}, \fontfamily{ptex}{\ttfamily \LaTeX}, up\fontfamily{ptex}{\ttfamily \LaTeX})

- \\texttt{\autospacing}
- \\texttt{\autoxspacing}
\disinhibitglue — New primitive since p3.8.2 (\TeX Live 2019)
\dtou
\euc
\ifdbox — New primitive since p3.2 (\TeX Live 2011)
\ifddir — New primitive since p3.2 (\TeX Live 2011)
\ifjfont — New primitive since p3.8.3 (\TeX Live 2020)
\ifmbox — New primitive since p3.7.1 (\TeX Live 2017)
\ifmdir
\iftbox
\iftdir
\iftfont — New primitive since p3.8.3 (\TeX Live 2020)
\ifybox
\ifydir
\inhibitglue
\inhibitxspcode
\jcharwidowpenalty
\jfam
\jfont
\jis
\kanjiskip
\kansuji
\kansujichar
\kcatcode
\kuten
\noautospacing
\noautoxspacing
\postbreakpenalty
\prebreakpenalty
\ptexfontname — New primitive since p4.1.0 (\TeX Live 2023)
\ptexlineendmode — New primitive since p4.0.0 (\TeX Live 2022)
\ptexversion — New primitive since p3.8.0 (\TeX Live 2018)
\ptexrevision — New primitive since p3.8.0 (\TeX Live 2018)
\ptextracingfonts — New primitive since p4.1.0 (\TeX Live 2023)
\ptexversion — New primitive since p3.8.0 (\TeX Live 2018)
\scriptbaselineshiftfactor — New primitive since p3.7 (\TeX Live 2016)
\scriptscriptradial — New primitive since p3.7 (\TeX Live 2016)
\textbaselineshift — New primitive since p3.7 (\TeX Live 2016)
\showmode
\#jish
\tate
\tbaselineshift
\textbaselineshift — New primitive since p3.7 (\TeX Live 2016)
\textfont
\tojis — New primitive since p4.1.0 (\TeX Live 2023)
\toucs — New primitive since p3.10.0 (\TeX Live 2022)
\ucs — Imported from up\TeX, since p3.10.0 (\TeX Live 2022)\(^6\)
\xkanjiskip
\xspcode
\ybaselineshift
\yoko
\H
\Q
\zh
\zw

6.3.2 up\TeX additions (available in up\TeX, \varepsilon-up\TeX)
\disablecjktoken
\enablecjktoken
\forcecjktoken

\(^6\)The primitive \texttt{\ucs} was part of "up\TeX additions" until \TeX Live 2021.
\kchar
\kchardef
\uptexrevision — New primitive since u1.23 (\TeX\ Live 2018)
\uptexversion — New primitive since u1.23 (\TeX\ Live 2018)

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

\currentspacingmode — New primitive since 191112 (\TeX\ Live 2020)
\currentxspacingmode — New primitive since 191112 (\TeX\ Live 2020)
\epTeXinputencoding — New primitive since 160201 (\TeX\ Live 2016)
\epTeXversion — New primitive since 180121 (\TeX\ Live 2018)
\expanded — New primitive since 180518 (\TeX\ Live 2019)
\hfi
\ifincsname — New primitive since 190709 (\TeX\ Live 2020)
\ifpdfprimitive — New primitive since 150805 (\TeX\ Live 2016)
\lastnodechar — New primitive since 141108 (\TeX\ Live 2015)
\lastnodefont — New primitive since 220214 (\TeX\ Live 2022)
\lastnodesubtype — New primitive since 180226 (\TeX\ Live 2018)
\odelcode
\odelimiter
\omathaccent
\omathchar
\omathchardef
\omathcode
\oradical
\pagefistretch
\pdffield — New primitive since 140506 (\TeX\ Live 2015)
\pdffield — New primitive since 130605 (\TeX\ Live 2014)
\pdffield — New primitive since 130605 (\TeX\ Live 2014)
\pdflastxpos
\pdflastypos
\pdfmdfivesum — New primitive since 150702 (\TeX\ Live 2016)
\pdfnormaldeviate — New primitive since 161114 (\TeX\ Live 2017)
\pdfpageheight
\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)
\suppresslongerror — New primitive since 211207 (\TeX\ Live 2022)
\suppressmathparerror — New primitive since 211207 (\TeX\ Live 2022)
\suppressoutererror — New primitive since 211207 (\TeX\ Live 2022)
\Uchar — New primitive since 191112 (\TeX\ Live 2020)
\Ucharcat — New primitive since 191112 (\TeX\ Live 2020)
\vadjust pre — New keyword since 210701 (\TeX\ Live 2022)
\vfi
fi

6.3.4 ε-up\TeX\ additions (available in ε-up\TeX)
\currentcjktoken — New primitive since 191112 (\TeX\ Live 2020)

6.3.5 Other cross-engine additions
Sync\TeX\ extension (available in p\TeX, up\TeX, ε-p\TeX, ε-up\TeX):
\synctex
\TeX{} Live additions (available in p\TeX{}, up\TeX{}, $\varepsilon$-p\TeX{}, $\varepsilon$-up\TeX{}):

- \texttt{\partokencontext} — New primitive since \TeX{} Live 2022
- \texttt{\partokenname} — New primitive since \TeX{} Live 2022
- \texttt{\showstream} — New primitive since \TeX{} Live 2022
- \texttt{\special shipout} — New keyword since 230214 (\TeX{} Live 2023)
- \texttt{\tracingstacklevels} — New primitive since \TeX{} Live 2021

6.4 Omitted primitives and unsupported features

Compared to Knuthian \TeX{} and $\varepsilon$-\TeX{}, some primitives and extensions are omitted due to conflict with Japanese handling.

- The enc\TeX{} extension, including the primitives \texttt{\mubyte} etc., is unavailable.
- The ML\TeX{} extension, such as \texttt{\charsubdef}, is not enabled by default. It becomes available with the command-line option \texttt{-mltex}, but not well-tested.

6.5 Behavior of Western \TeX{} primitives

Here we provide some notes on behavior of Knuthian \TeX{} and $\varepsilon$-\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}, \texttt{\sfcode}, \texttt{\mathcode}, \texttt{\delcode}, \texttt{\lccode}, \texttt{\uccode}.

Each of the following primivies has \texttt{\ldots char} in its name, however, the effective values are restricted to 0–255.

\texttt{\endlinechar}, \texttt{\newlinechar}, \texttt{\escapechar}, \texttt{\defaulthyphenchar}, \texttt{\defaultskewchar}.

6.5.2 Primitives capable of handling Japanese

The following primitives are extended to support Japanese characters:
\char \langle character code \rangle, \chardef \langle control sequence \rangle = \langle character code \rangle

In addition to 0–255, internal codes of Japanese characters (see 8.1) are allowed. For putting Japanese characters, a Japanese font (see 7.2) is chosen. Further information can be found in 6.6.3.

\font, \fontname, \fontdimen

\accent \langle character code \rangle = \langle character \rangle

\if \langle token_1 \rangle \langle token_2 \rangle, \ifcat \langle token_1 \rangle \langle token_2 \rangle

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 tex.web, 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

6.6.2 Detecting up\TeX

up\TeX is almost upward compatible with p\TeX, however, there are two major differences:

1. Improvements in the \kcatcode business, mainly for better handling of Latin-1 characters and CJK tokens.
2. Unicode as the default internal Japanese encoding (see \[8.1\]), for direct use of its huge character set.

The first difference can be detected by checking if \(\ldots cjktoken\) primitive is defined.

\[
\texttt{\ifx\enablecjktoken\undefined}\texttt{\else}\% upTe\,X/e-upTe\,X\\\texttt{\fi}
\]

This can be called “engine detection” of upTeX.

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

\[
\texttt{\ifx\kanjiskip\undefined}\texttt{\else}\texttt{\ifnum\jis"2121="3000}\% upTe\,X/e-upTe\,X with internal Unicode\\\texttt{\else}\% pTe\,X/e-pTe\,X\\% or, upTe\,X/e-upTe\,X with internal EUC-JP or Shift-JIS\\\texttt{\fi}\\\texttt{\fi}
\]

This can be called “encoding detection” of upTeX.

Please note that the format-build setting of \texttt{-kanji-internal=(sjis|euc)} with upTeX makes it effectively pTeX regarding the character set, which means that only JIS X 0208 character set is supported. This can be called “legacy-encoding-compatibility mode” of upTeX, where the \texttt{\kcatcode} difference remains but the internal encoding difference disappears. This method is used in building \texttt{platex.fmt} on \texttt{-upTeX}, since 2023-06-01. Therefore, to distinguish upTeX from \texttt{pTeX}, “engine detection” is not enough; you should use “encoding detection.”

### 6.6.3 Defining large integer constants

According to \[2\] (Section 3.3),

A control sequence that has been defined with a \texttt{\chardef} command can also be used as a \texttt{\langle number\rangle}. 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 \texttt{pTeX} 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/\varepsilon-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 \varepsilon-p\TeX/\varepsilon-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 \S1). The most efficient and convenient way of defining integer constants is as follows:

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

6.6.4 Creating a Japanese character token with a specified code

Short version:

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

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

\lowercase trick
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
\lcodename?=\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 p\TeX 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. \e-\TeX 191112 or later (\TeX Live 2020) provides expandable primitives \Uchar and \Ucharcat, which are derived from \Xe\TeX. Regardless of their names, and unlike \Xe\TeX or \Lu\TeX, these primitives do not necessarily take a Unicode value as an argument. These primitives in \e-\TeX and \e-up\TeX take a valid character code (see \S\S\S) based on the internal Japanese encoding.

\Uchar ⟨character code⟩

Expands to a character token with specified slot ⟨character code⟩.

• 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 \e-\TeX, 16–19 for \e-up\TeX.

\Ucharcat ⟨character code⟩ ⟨category code⟩

Expands to a character token with slot ⟨character code⟩ and ⟨category code⟩ specified.

• With \e-\TeX:
– 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 in Section 3, 𝜀-TeX/𝜀-upTeX are not upward compatible with DVI mode of pdfTeX, which is available as the \texttt{etex} command in TeX Live. Here we list some important differences:

First, some pdfTeX-specific primitives are absent. Examples:

• All primitives specific to PDF output: \texttt{pdffontwidth}, \texttt{pdffontwidth}, \texttt{pdfobj} etc.

• All primitives related to micro-typography: \texttt{pdfprotrudechars}, etc.

• Some primitives related to handling of strings: \texttt{pdfescapestring}, \texttt{pdfescapehex} etc.

6.8 Recommendation for file encoding

Due to historical reasons, multiple encodings are commonly used for Japanese text. Sometimes user documents and distribution files (classes, packages) may have different encodings. Among those, the universal UTF-8 and three major legacy encodings (ISO-2022-JP, EUC-JP, Shift-JIS) are accepted as input to \TeX family, depending on the configuration and runtime options. To make this possible, \TeX family does code conversion in input and output.

The details are too complicated, so here we propose the optimum solution for Japanese file encoding for package/class developers who aim to support \TeX family:

• If you want to distribute files on CTAN/\TeX Live, please use UTF-8.

UTF-8 files are almost always safe enough for recent \TeX/\upTeX (2018–), and the same files will have no problem when read by Western \TeX. To secure this “almost always” to “always”, please add below at the beginning of your individual UTF-8 files:

\begin{verbatim}
\epsilon-\texfamily/\epsilon-up\texfamily has primitives \texttt{pdfpagewidth} and \texttt{pdfpageheight}; this is just because they were convenient for implementing \texttt{pdfsavepos}, and their behavior is somewhat different from that of pdf\texfamily. Also note that \epsilon-\texfamily/\epsilon-up\texfamily does not have \texttt{pdfhorigin} and \texttt{pdfvorigin}.
\end{verbatim}
it will help \TeX family to read forcibly in UTF-8, so it becomes \textit{always} safe for 2016–.

- If you aim to support broader legacy environment of \TeX specifically, ...
  
  Extra care is required for missing features and different configurations which can lead to failure of reading UTF-8. There is no such thing as a perfect solution; instead, you should choose between encoding in ISO-2022-JP or writing in ASCII characters.

  - Encoding in ISO-2022-JP:
    
    All historical versions of \TeX family can always read ISO-2022-JP properly because it’s a 7-bit encoding safely distinguished from others. This is why many old packages/classes widely used in Japan are particularly encoded in ISO-2022-JP.
    
    On the other hand, ISO-2022-JP is unsupported in Western \TeX; also, CTAN/\TeX Live requires some special handling of uploaded files.

  - Writing in ASCII characters:
    
    Safe for Western \TeX and CTAN/\TeX Live, but often requires lots of \TeXniques and hard-to-read programmings (e.g. generating Japanese tokens as in Section 6.6.4, or encoding into hex dump as in bxjalipsum.sty, ...)
    
    ... Annoying? Please forget that legacy environment ;-) 

For your information, here is the behavior of the common default configuration available in the latest \TeX Live distribution (since 2023).

- up\TeX default: always properly reads UTF-8 and ISO-2022-JP.

- \TeX default: always properly reads ISO-2022-JP. It also properly reads UTF-8 almost always\textsuperscript{8}, and successful results of “guess-input-enc” conversion of Shift-JIS and EUC-JP.

- When the command-line option -kanji=(sjis|euc) is specified: UTF-8 above is replaced with the given encoding.

Note on older versions:

- The “guess-input-enc” conversion status above is relatively new:
  
  - In \TeX Live 2023, it is available for all platform of \TeX Live. Default on for (-)\TeX, off for (-)up\TeX, but also controlled by runtime option -(no-)guess-input-enc.
  
  - In \TeX Live 2022 and older, it was limited for Windows only; also, default on for all of (-)up\TeX. For Unix it was not implemented yet.

- In \TeX Live 2017 and older, the default input encoding of (-)\TeX was Shift-JIS for Windows, UTF-8 for Unix.

\textsuperscript{8}The exception of “almost always” comes from a failure of guessing; at the expense of properly reading a certain amount of Shift-JIS and EUC-JP, there are occasional misreading of UTF-8.
• The primitive `\epTeXinputencoding` was added to `\vTeX/\vupTeX` in `\TeX` Live 2016. Older versions do not have it.
• Very old distributions by ASCII Corporation (~2009) supported only legacy encodings; UTF-8 was not allowed.

6.9 Input handling

For simplicity, first we introduce how `upTeX` handles the input when all files are UTF-8.

1. An input line is stored into the internal buffer. (No effective code conversion here for `upTeX`.)
2. The input processor reads the buffer. Here Japanese character tokens are distinguished from ordinary 8-bit character tokens.
3. ... [TODO]

The situation is more complicated in `pTeX`. As described in Section 6.8, it accepts UTF-8 input; however, `pTeX` uses a legacy encoding as the internal Japanese encoding (default: Shift-JIS on Windows, and EUC-JP otherwise). This means that `pTeX` does code conversion in input and output. ... [TODO]

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

---

9To be precise, it passes through a code conversion; however, this is an identity conversion which has no effect because `upTeX` defaults to internal Unicode.
latter two are specific to p\TeX family, which are defined in a JFM (Japanese \TeX font metric) format.\footnote{A JFM is a modified version of the standard TFM. It can be created by \(\text{(u)pP}l\text{toTF},\) and decoded by \(\text{(u)pTF}t\text{oPL}.\) Please also refer to the man pages of these programs (\text{ppltotf.man1.pdf} and \text{ptftopl.man1.pdf}).}

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

Moreover, p\TeX and friends assume that each Japanese font (except “Japanese \texttt{\nullfont}” in ini\TeX 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 Japanese encodings}

The \texttt{⟨character code⟩} 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 \TeX, but the latter is a problem.

In up\TeX (default internal Unicode mode), the range is very simple:

\[ c \geq 0 \]

However in p\TeX, only limited encodings are available; Shift-JIS as \texttt{sjis} (default for \TeX Live Windows), or EUC-JP as \texttt{euc} (otherwise). The range can be represented as follows:

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

\[ 0 \leq c_1 < 256 \quad 0 \leq c_2 < 128 \]
where

\[
\begin{align*}
  C_1 &= C_2 = \{a_1, \ldots, fe\} \quad \text{(euc),} \\
  C_1 &= \{81, \ldots, 9f\} \cup \{e0, \ldots, fc\}, \\
  C_2 &= \{40, \ldots, 7e\} \cup \{80, \ldots, fc\} \quad \text{(jis).}
\end{align*}
\]

Therefore, the overall range of \textit{character code} is not continuous. This is similar for “legacy-encoding-compatibility mode” of up\TeX{}.

To check whether an integer is a valid Japanese character code or not, you can use \texttt{\iffontchar} with \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\(^{11}\) Japanese font containing all valid Japanese character code (see \texttt{\textasciitilde{\textcircled{\texttt{\textasciitilde}}}}).

Note that \texttt{p\TeX{}} (not including \texttt{up\TeX{}} with internal Unicode) does not support typesetting characters outside JIS X 0208, which is a subset of accepted range of \textit{character code} described above. To check if an integer is in the range of JIS X 0208, you can use \texttt{\toucs} with \texttt{p\TeX{} p3.10.0} or later (\TeX{} Live 2022):

\begin{verbatim}
\ifnum\toucs\mycount>0
  \% \mycount is in the range of JIS X 0208
\fi
\end{verbatim}

The primitive \texttt{\toucs} converts an integer value from an internal Japanese code to a Unicode. This conversion is based on JIS-Unicode mapping table,\(^{12}\) and returns \(-1\) if no mapping is available for the input integer.

---

\(^{11}\)This assumption is always safe after one of the standard \texttt{p\TeX{}} formats (e.g. plain \texttt{p\TeX{}}, \texttt{pl\TeX{}}) is loaded.

\(^{12}\)Defined in \texttt{jisx0208.h} of \texttt{ptexenc} library.
References

https://ctan.org/pkg/dvistd

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

<table>
<thead>
<tr>
<th>Symbols</th>
<th>\inhibitxspcode</th>
<th>\jcharwidowpenalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>\accent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\autospacing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\autooxspacing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\char</td>
<td>\kansujichar</td>
<td></td>
</tr>
<tr>
<td>\chardef</td>
<td>\kansuji</td>
<td></td>
</tr>
<tr>
<td>\currentcjktoken</td>
<td>\kansujichar</td>
<td></td>
</tr>
<tr>
<td>\currentxspacingmode</td>
<td>\kcatcode</td>
<td></td>
</tr>
<tr>
<td>\disablecjktoken</td>
<td>\kchar</td>
<td></td>
</tr>
<tr>
<td>\disinhibitglue</td>
<td>\kchardef</td>
<td>\pdfnormaldeviate</td>
</tr>
<tr>
<td>\dtou</td>
<td>\kuten</td>
<td></td>
</tr>
<tr>
<td>\enablecjktoken</td>
<td>\lastnodechar</td>
<td>\pdfmdfivesum</td>
</tr>
<tr>
<td>\epTeXinputencoding</td>
<td>\lastnodefont</td>
<td>\pdflastypos</td>
</tr>
<tr>
<td>\epTeXversion</td>
<td>\lastnodesubtype</td>
<td>\pdflastxpos</td>
</tr>
<tr>
<td>\eucl</td>
<td>\mathchardef</td>
<td>\pdffilesize</td>
</tr>
<tr>
<td>\expanded</td>
<td>\noautospacing</td>
<td>\pdffilemoddate</td>
</tr>
<tr>
<td>\font</td>
<td>\noautoxspacing</td>
<td>\pdffiledump</td>
</tr>
<tr>
<td>\fontdimen</td>
<td>\odelcode</td>
<td>\pdfelapsedtime</td>
</tr>
<tr>
<td>\fontname</td>
<td>\odelimiter</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\forcenodecjspace</td>
<td>\omathaccent</td>
<td>\pdffilesize</td>
</tr>
<tr>
<td>\hfi</td>
<td>\omathchar</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\if</td>
<td>\omathchardef</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\ifcat</td>
<td>\omathcode</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\ifdbx</td>
<td>\oradical</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\ifddir</td>
<td>\pagefistretch</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\ifincsname</td>
<td>\partokencontext</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\ifjfont</td>
<td>\partokenname</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\ifmbox</td>
<td>\pdfcreationdate</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\ifpdfdir</td>
<td>\pdfcreationdate</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\ifpdfprimitive</td>
<td>\pdfcreationdate</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\iftbox</td>
<td>\pdfcreationdate</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\iftfont</td>
<td>\pdfcreationdate</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\ifybox</td>
<td>\pdfcreationdate</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\ifydir</td>
<td>\pdfcreationdate</td>
<td>\pdfcreationdate</td>
</tr>
<tr>
<td>\inhibitglue</td>
<td>\pdfcreationdate</td>
<td>\pdfcreationdate</td>
</tr>
</tbody>
</table>
\pdfpageheight \pdfpagewidth \pdfprimitive \pdfrandomseed \pdfresettimer \pdfsavepos \pdfsetrandomseed \pdfshellescape \pdfstrcmp \pdfuniformdeviate \postbreakpenalty \prebreakpenalty \ptexfontname \ptexlineendmode \ptxminorversion \ptexrevision \ptextracingfonts \readpapersizespecial \scriptbaselineshiftfactor \scriptscriptrabaselineshiftfactor \showmode \showstream \sjis \special \suppresslongerror \suppressmathparerror \suppressoutererror \synctex \tate \tbaselineshift \textbaselineshiftfactor \tfont \tojis \toucs \tracingstacklevels \Uchar \Ucharcat \ucs \uptexrevision

F

fi

H

H

Q

Q

Z

zh

zw

27