Guide to p\TeX for developers unfamiliar with Japanese

Japanese \TeX Development Community*

version p3.8.3, December 13, 2020

p\TeX and its variants, up\TeX, \varepsilon-p\TeX and \varepsilon-up\TeX, 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 Brief introduction 4

1 pT EX and its variants 4

2 Eminent characteristics of pT EX family 4

3 Compatibility with Western T EX 5

4 L A TEX on pT EX/upT EX — pL A TEX/upL A TEX 5

II Details 6

5 Output format — DVI 6

5.1 Extensions of DVI format in pT EX family 7

5.2 DVI drivers with Japanese support 8

5.2.1 Using dvipdfmx 8

5.2.2 Using dvips 8

6 Programming on pT EX 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 SynclpX additions (available in plpX, uplpX, e- qlpX, e- uplpX) 9

6.3.2 qlpX additions (available in qlpX, upqlpX, e-qlpX, e-upqlpX) 10

6.3.3 uplpX additions (available in uplpX, e-uplpX) 11

6.3.4 e-qlpX additions (available in e-qlpX, e-upqlpX) 12

6.3.5 e-upqlpX additions (available in e-upqlpX) 13

6.4 Omitted primitives and unsupported features 13

6.5 Behavior of Western T EX 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 plpX 14

6.6.2 Detecting uplpX 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 pdfT EX in DVI mode 18
Part I
Brief introduction

1 p\TeX and its variants

The figure below shows the relationship between engines.

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

p\TeX 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).

up\TeX is developed as an extension of p\TeX to support full Unicode characters. It also includes extensions to overcome the difficulties of p\TeX 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 p\TeX and up\TeX respectively. In the current release, some extensions derived from pdfp\TeX and \Omega are also available.

2 Eminent characteristics of p\TeX family

The most important characteristics of p\TeX 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; 横組) 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.

$\varepsilon-p\TeX/\varepsilon-up\TeX$ 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 `etex` only as “DVI mode of pdf$\TeX$.” You should note that $\varepsilon-p\TeX/\varepsilon-up\TeX$ 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 $\varepsilon-p\TeX/\varepsilon-up\TeX$, so we focus mainly on $\varepsilon-p\TeX/\varepsilon-up\TeX$.

4 \LaTeX{} on $p\TeX/\up\TeX$ — $p\LaTeX{}/\up\LaTeX{}$

Formats based on $\LaTeX{}$ is called $p\LaTeX{}$ when running on $p\TeX/\varepsilon-p\TeX$, and called $\up\LaTeX{}$ when running on $\up\TeX/\varepsilon-up\TeX$. In recent versions (around 2011) of $\TeX$ Live and its derivatives, the default engines of $p\LaTeX{}$ and $\up\LaTeX{}$ are $\varepsilon-p\TeX$ and $\varepsilon-up\TeX$. That is, the command `platex` starts $\varepsilon-p\TeX$ (not $p\TeX$) with preloaded format `platex.fmt`.

In the kernel level (`platex.ltx` and `uplatex.ltx`), $p\LaTeX{}$ and $\up\LaTeX{}$ adds some additional commands related to the followings:

- Selection of Japanese fonts,
- Crop marks (called `tombow`; トombow) 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 $p\LaTeX{}/\up\LaTeX{}$, `(bottom float)` is placed above `(footnote)`. That is, the complete order is `(top float) → (body text) → (bottom float) → (footnote)`.

For developers, additional care may be needed, for changes in the kernel macros and/or the absence of pdf$\TeX$ features.
Part II
Details

5 Output format — DVI

The output of \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 \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 dvips (e.g. 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 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.}

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

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

The DVI format output by \texttt{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{pandoc} but never used in \TeX82, can come out.

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

• \texttt{set3 (130)}: Used to typeset a Japanese character with 3-byte code (\texttt{up\TeX} only).

When \texttt{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 \texttt{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, \texttt{p\TeX}/\texttt{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 \texttt{p\TeX}/\texttt{up\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 *pTeX* 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 *pTeX* 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 *TeX* Development Community.

6 Programming on *pTeX* family

We focus on programming aspects of *pTeX* and its variants.

6.1 Number of registers and marks

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

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

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

6.2 Number of math families

In \ueTeX and \ueupTeX, the number of math fonts is restricted to 16, each of which can contain 256 characters (same as Knuthian \TeX). In ε-\ueTeX and ε-\ueupTeX, a change file fam256.ch, which is derived from Ω, extends the upper limit to 256. As a consequence, ε-\ueTeX and ε-\ueupTeX 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 \ue\ueTeX users to use more than 16 math fonts, it is necessary to use macros which exploit Ω-derived primitives such as \omathchar. Recent (u)p\ueTeX (since 2016/11/29) partially supports this, and the maximum number of math alphabets that can be defined by \DeclareMathAlphabet is extended to 256 (\epsilonmathgroup@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 \mathfam256\footnote{\url{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 \ue\ueTeX family in alphabetical order. The features of each primitive can be found in Japanese edition.

6.3.1 Sync\ueTeX additions (available in \ue\ueTeX, \ueupTeX, ε-\ueTeX, ε-\ueupTeX)

In the standard build of \ueTeX Live, Sync\ueTeX extension is unavailable in Knuthian \ueTeX; however, it is enabled in \ue\ueTeX family.
6.3.2 \texttt{p\LaTeX} additions (available in \texttt{p\LaTeX}, \texttt{up\LaTeX}, \texttt{\varepsilon-p\LaTeX}, \texttt{\varepsilon-up\LaTeX})

- \texttt{\synctex}

- \texttt{\autospacing}

- \texttt{\autoxspacing}

- \texttt{\disinhibitglue} — New primitive since p3.8.2 (\TeX\ Live 2019)

- \texttt{\dtou}

- \texttt{\euc}

- \texttt{\ifdbox} — New primitive since p3.2 (\TeX\ Live 2011)

- \texttt{\ifddir} — New primitive since p3.2 (\TeX\ Live 2011)

- \texttt{\iffbox} — New primitive since p3.8.3 (\TeX\ Live 2020)

- \texttt{\iftbox} — New primitive since p3.7.1 (\TeX\ Live 2017)

- \texttt{\iftbox}

- \texttt{\ifdir}

- \texttt{\iftbox} — New primitive since p3.8.3 (\TeX\ Live 2020)

- \texttt{\ifybox}

- \texttt{\ifydir}

- \texttt{\inhibitglue}

- \texttt{\inhibitxspcode}

- \texttt{\jcharwidowpenalty}

- \texttt{\jfam}

- \texttt{\jfont}

- \texttt{\jis}

- \texttt{\kanjiskip}

- \texttt{\kansuji}

- \texttt{\kansujichar}

- \texttt{\kcatcode}

- \texttt{\kuten}

- \texttt{\noautospacing}
\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 \texttt{up\TeX} additions (available in \texttt{up\TeX}, \texttt{\varepsilon-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 \textit{\varepsilon\-\textsf{\LaTeX}} additions (available in \textit{\varepsilon\-\textsf{\LaTeX}}, \textit{\varepsilon\-\textsf{up\LaTeX}})

- \texttt{\currentspacingmode} — New primitive since 191112 (\textsf{\LaTeX} Live 2020)
- \texttt{\currentxspacingmode} — New primitive since 191112 (\textsf{\LaTeX} Live 2020)
- \texttt{\epTeXinputencoding} — New primitive since 160201 (\textsf{\LaTeX} Live 2016)
- \texttt{\epTeXversion} — New primitive since 180121 (\textsf{\LaTeX} Live 2018)
- \texttt{\expanded} — New primitive since 180518 (\textsf{\LaTeX} Live 2019)
- \texttt{\hfi}
- \texttt{\ifincsname} — New primitive since 190709 (\textsf{\LaTeX} Live 2020)
- \texttt{\ifpdfprimitive} — New primitive since 150805 (\textsf{\LaTeX} Live 2016)
- \texttt{\lastnodechar} — New primitive since 141108 (\textsf{\LaTeX} Live 2015)
- \texttt{\lastnodesubtype} — New primitive since 180226 (\textsf{\LaTeX} Live 2018)
- \texttt{\odelcode}
- \texttt{\odelimiter}
- \texttt{\omathaccent}
- \texttt{\omathchar}
- \texttt{\omathchardef}
- \texttt{\omathcode}
- \texttt{\oradical}
- \texttt{\pagefistretch}
- \texttt{\pdffcreationdate} — New primitive since 130605 (\textsf{\LaTeX} Live 2014)
- \texttt{\pdffelapsedtime} — New primitive since 161114 (\textsf{\LaTeX} Live 2017)
- \texttt{\pdffiledump} — New primitive since 140506 (\textsf{\LaTeX} Live 2015)
- \texttt{\pdffilename} — New primitive since 130605 (\textsf{\LaTeX} Live 2014)
- \texttt{\pdffilesize} — New primitive since 130605 (\textsf{\LaTeX} Live 2014)
- \texttt{\pdfflastxpos}
- \texttt{\pdfflastypos}
- \texttt{\pdfmdfivesum} — New primitive since 150702 (\textsf{\LaTeX} Live 2016)
- \texttt{\pdfnormaldeviate} — New primitive since 161114 (\textsf{\LaTeX} Live 2017)
- \texttt{\pdfpageheight}
- \texttt{\pdfpagewidth}
\newcommand{\currentcjktoken}{New primitive since 191112 (\TeX Live 2020)}

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

\newcommand{\currentcjktoken}{New primitive since 191112 (\TeX Live 2020)}

6.4 Omitted primitives and unsupported features

Compared to Knuthian \TeX and \textit{\varepsilon}\text-\textit{\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{\varepsilon}\text-\textit{\TeX} primitives when used within \pTeX 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.”

\newcommand{\catcode, \scode, \mcode, \dcode, \lcode, \ucode}{...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 \langle character code \rangle, \chardef \langle control sequence \rangle = \langle character code \rangle

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

▶ \if \langle token \rangle \langle token \rangle, \ifcat \langle token \rangle \langle token \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.

\hfill \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 \texttt{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
% \TeX / up\TeX / e-\TeX / e-up\TeX
\fi
6.6.2 Detecting up\TeX\n
up\TeX/\texttt{\textasciitilde-up\TeX} are almost upward compatible with p\TeX/\texttt{\textasciitilde-p\TeX} respectively, however, there are two major differences:

1. Improvements in the \texttt{\textbackslash catcode} 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
    \ifs\jis"2121="3000
      \% up\TeX/e-up\TeX with internal Unicode
    \else
      \% p\TeX/e-p\TeX
    \fi
    \% 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 \cite{2} (Section 3.3),

A control sequence that has been defined with a \texttt{\textbackslash chardef} command can also be used as a \texttt{\langle number\rangle}. This fact is used in allocation commands such as \texttt{\textbackslash 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:

15
\chardef \langle control sequence⟩ = \langle character code⟩
Defines a control sequence to be a synonym for \char \langle character code⟩.

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

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

\omathchardef \langle control sequence⟩ = \langle 27-bit number⟩ (for \varepsilon-p\TeX/\varepsilon-up\TeX)
Defines a control sequence to be a synonym for \omathchar \langle 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 \texttt{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 \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".

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{?}}

16
defines \textbackslash X which expands to a character number \textbackslash mycount while the \textbackslash catcode of ? (12) is preserved. However, the trick cannot be applied to Japanese characters, since \texttt{p\TeX} family does not support \textbackslash lccode outside 0–255.

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

Example 1: equivalent to \texttt{\def\X{あ}} (JIS code \texttt{0x2422} is “あ”):

\begin{verbatim}
\begingroup
  \kansujichar1=\jis"2422 \xdef\X{\kansuji\texttt{1}}
\endgroup
\end{verbatim}

Example 2: equivalent to \texttt{\def\\texttt{೔ຊ}{Japan}}.

\begin{verbatim}
\begingroup
  \kansujichar5=\jis"467C\relax
  \kansujichar6=\jis"4B5C\relax
  \expandafter\gdef\csname\kansuji\texttt{56}\endcsname{Japan}
\endgroup
\end{verbatim}

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

\texttt{\Uchar}, \texttt{\Ucharcat}

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

\begin{itemize}
  \item \texttt{\Uchar \langle character code \rangle}

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

  \begin{itemize}
    \item 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.
    \item 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 \texttt{\e-p\TeX}, 16–19 for \texttt{\e-up\TeX}.
  \end{itemize}

  \item \texttt{\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.

  \begin{itemize}
    \item With \texttt{\e-p\TeX}:
  \end{itemize}
\end{itemize}
– Only 8-bit number (0–255) are allowed for \textit{character code}; that is, only Latin characters can be generated.
– The values allowed for \textit{category code} are 1–4, 6–8, 10–13.

\begin{itemize}
\item With $\varepsilon$-up$\TeX$:
  \begin{itemize}
  \item When \textit{character code} is between 0–127, only Latin characters can be generated. Thus, the values allowed for \textit{category code} are 1–4, 6–8, 10–13.
  \item When \textit{character code} is between 128–255, both Latin and Japanese characters can be generated depending on the specified \textit{category code}; 1–4, 6–8, 10–13: Latin character, 16–19: Japanese character.
  \item When \textit{character code} is greater than 255, only Japanese characters can be generated. Thus, the values allowed for \textit{category code} are 16–19.
  \end{itemize}
\end{itemize}

6.7 Difference from pdf$\TeX$ in DVI mode

As stated in Section 3, $\varepsilon$-p$\TeX$/ $\varepsilon$-up$\TeX$ are \textit{not} upward compatible with DVI mode of pdf$\TeX$, which is available as the \texttt{etex} command in $\TeX$ Live. Here we list some important differences:

First, some pdf$\TeX$-specific primitives are absent. Examples:

\begin{itemize}
\item All primitives specific to PDF output: \texttt{\pdfoutput}, \texttt{\pdfinfo}, \texttt{\pdfobj} etc.\footnote{$\varepsilon$-p$\TeX$/ $\varepsilon$-up$\TeX$ 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$\TeX$. Also note that $\varepsilon$-p$\TeX$/ $\varepsilon$-up$\TeX$ does not have \texttt{\pdfhorigin} and \texttt{\pdfvorigin}.}
\item All primitives related to micro-typography: \texttt{\pdffontexpand}, \texttt{\pdfprotrudechars}, etc.
\item Some primitives related to handling of strings: \texttt{\pdfescapestring}, \texttt{\pdfescapehex} etc.
\end{itemize}

6.8 Recommendation for file encoding

6.9 Input handling

For simplicity, first we introduce of input handling of $\varepsilon$-up$\TeX$.

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 p\TeX family automatically inserts glue and penalties between characters.

7.2 Japanese fonts

p\TeX 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 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 (u)p\text{PLtoTF}, and decoded by (u)p\text{TFtoPL}. Please also refer to the man pages of these programs (ppltotf.man1.pdf and 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 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.

8 Other strange beasts

8.1 Internal kanji 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/\varepsilon-up\TeX with -kanji-internal=uptex (default on), the range is very simple:

\[ c \geq 0 \]

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

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

where

\[
\begin{align*}
C_1 &= C_2 = \{"a1,\ldots,\text{"fe}\} \quad \text{(euc),} \\
C_1 &= \{"81,\ldots,\text{"9f}\} \cup \{"e0,\ldots,\text{"fc}\}, \quad C_2 = \{"40,\ldots,\text{"7e}\} \cup \{"80,\ldots,\text{"fc}\} \quad \text{(sjis).}
\end{align*}
\]

Therefore, the overall range of \textit{(character code)} is not continuous.

To check whether an integer is a valid Japanese character code or not, you can use \texttt{\iffontchar} with \varepsilon-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\textsuperscript{8} Japanese font containing all valid Japanese character code (see \texttt{\iffontchar}).

\textsuperscript{8}This assumption is always safe after one of the standard p\TeX formats (e.g. plain p\TeX, p\LaTeX) is loaded.
References

   https://ctan.org/pkg/dvistd

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

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>\accent</td>
<td>14</td>
</tr>
<tr>
<td>\autospacing</td>
<td>14</td>
</tr>
<tr>
<td>\autoxspacing</td>
<td>14</td>
</tr>
<tr>
<td>\char</td>
<td>14</td>
</tr>
<tr>
<td>\chardef</td>
<td>14</td>
</tr>
<tr>
<td>\currentcjktoken</td>
<td>14</td>
</tr>
<tr>
<td>\currentxspacingmode</td>
<td>14</td>
</tr>
<tr>
<td>\disablecjktoken</td>
<td>14</td>
</tr>
<tr>
<td>\disinhibitglue</td>
<td>14</td>
</tr>
<tr>
<td>\dtou</td>
<td>14</td>
</tr>
<tr>
<td>\enablecjktoken</td>
<td>14</td>
</tr>
<tr>
<td>\epTeXinputencoding</td>
<td>14</td>
</tr>
<tr>
<td>\epTeXversion</td>
<td>14</td>
</tr>
<tr>
<td>\euclid</td>
<td>14</td>
</tr>
<tr>
<td>\expanded</td>
<td>14</td>
</tr>
<tr>
<td>\font</td>
<td>14</td>
</tr>
<tr>
<td>\fontdimen</td>
<td>14</td>
</tr>
<tr>
<td>\fontname</td>
<td>14</td>
</tr>
<tr>
<td>\forcecjktoken</td>
<td>14</td>
</tr>
<tr>
<td>\hfi</td>
<td>14</td>
</tr>
<tr>
<td>\if</td>
<td>14</td>
</tr>
<tr>
<td>\ifcat</td>
<td>14</td>
</tr>
<tr>
<td>\ifddir</td>
<td>14</td>
</tr>
<tr>
<td>\ifdir</td>
<td>14</td>
</tr>
<tr>
<td>\ifincsname</td>
<td>14</td>
</tr>
<tr>
<td>\ifjfont</td>
<td>14</td>
</tr>
<tr>
<td>\ifmbox</td>
<td>14</td>
</tr>
<tr>
<td>\ifpdfprimitive</td>
<td>14</td>
</tr>
<tr>
<td>\iftbox</td>
<td>14</td>
</tr>
<tr>
<td>\iftdir</td>
<td>14</td>
</tr>
<tr>
<td>\iftfont</td>
<td>14</td>
</tr>
<tr>
<td>\ifybox</td>
<td>14</td>
</tr>
<tr>
<td>\ifydir</td>
<td>14</td>
</tr>
<tr>
<td>\inhibitglue</td>
<td>14</td>
</tr>
<tr>
<td>\inhibitxspcode</td>
<td>14</td>
</tr>
<tr>
<td>\jcharwidowpenalty</td>
<td>14</td>
</tr>
<tr>
<td>\jdir</td>
<td>14</td>
</tr>
<tr>
<td>\jfont</td>
<td>14</td>
</tr>
<tr>
<td>\jis</td>
<td>14</td>
</tr>
<tr>
<td>\kcatcode</td>
<td>14</td>
</tr>
<tr>
<td>\kchar</td>
<td>14</td>
</tr>
<tr>
<td>\kchardef</td>
<td>14</td>
</tr>
<tr>
<td>\kensui</td>
<td>14</td>
</tr>
<tr>
<td>\kensujichar</td>
<td>14</td>
</tr>
<tr>
<td>\klastnodesubtype</td>
<td>14</td>
</tr>
<tr>
<td>\kmathaccent</td>
<td>14</td>
</tr>
<tr>
<td>\kmathchar</td>
<td>14</td>
</tr>
<tr>
<td>\kmathchardef</td>
<td>14</td>
</tr>
<tr>
<td>\kmathcode</td>
<td>14</td>
</tr>
<tr>
<td>\kmathchardef</td>
<td>14</td>
</tr>
<tr>
<td>\kmathcode</td>
<td>14</td>
</tr>
<tr>
<td>\kmathdef</td>
<td>14</td>
</tr>
<tr>
<td>\kradical</td>
<td>14</td>
</tr>
<tr>
<td>\pagedfprimitive</td>
<td>14</td>
</tr>
<tr>
<td>\pdfcreationdate</td>
<td>14</td>
</tr>
<tr>
<td>\pdfelapsedtime</td>
<td>14</td>
</tr>
<tr>
<td>\pdfchardump</td>
<td>14</td>
</tr>
<tr>
<td>\pdffiledate</td>
<td>14</td>
</tr>
<tr>
<td>\pdffilemoddate</td>
<td>14</td>
</tr>
<tr>
<td>\pdffilesize</td>
<td>14</td>
</tr>
<tr>
<td>\pdflastxpos</td>
<td>14</td>
</tr>
<tr>
<td>\pdflastypos</td>
<td>14</td>
</tr>
<tr>
<td>\pdfmdfivesum</td>
<td>14</td>
</tr>
<tr>
<td>\pdfnormaldeviate</td>
<td>14</td>
</tr>
<tr>
<td>\pdfpageheight</td>
<td>14</td>
</tr>
<tr>
<td>\pdfpagewidth</td>
<td>14</td>
</tr>
<tr>
<td>\pdfprimitive</td>
<td>14</td>
</tr>
</tbody>
</table>

22